Міністерство освіти і науки України
Луцький національний технічний університет

ПРИКЛАДНЕ ТА WEB-ПРОГРАМУВАННЯ

Конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Професійна освіта (комп'ютерні технології)» галузі знань А Освіта
спеціальності А5.39 Професійна освіта (Цифрові технології)
денної та заочної форм навчання

Луцьк 2026

УДК 004.432:378.14 (07)
П 75

До друку

Голова вченої ради факультету цифрових, освітніх та соціальних технологій _____ Г.А. Герасимчук.

Затверджено вченою радою факультету цифрових, освітніх та соціальних технологій ЛНТУ, № ___ від «____»_____ 2026 року.

Електронна копія друкованого видання передана для внесення в репозитарій ЛНТУ

Директор бібліотеки _____ Н.П. Поліщук.

Рекомендовано до видання на засіданні кафедри цифрових освітніх технологій ЛНТУ, протокол № ___ від «___» _____2026 року.

Завідувач кафедри цифрових освітніх технологій _____ В.В. Кабак.

Укладач: _____ В.В. Кабак, кандидат педагогічних наук, доцент кафедри цифрових освітніх технологій ЛНТУ.

Рецензент: _____ О.І. Редько, кандидат технічних наук, доцент кафедри цифрових освітніх технологій.

Відповідальний за випуск: _____ В.В. Кабак, кандидат педагогічних наук, доцент, завідувач кафедри цифрових освітніх технологій ЛНТУ.

П 75 **Прикладне та Web-програмування**: конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Професійна освіта (комп'ютерні технології)» галузі знань А Освіта спеціальності А5.39 Професійна освіта (Цифрові технології) денної та заочної форм навчання / уклад. В. В. Кабак. Луцьк: ЛНТУ, 2026. 156 с.

Конкспект лекцій розроблено відповідно до діючої програми навчального курсу «Прикладне та Web-програмування». Він концентрує ґрунтовний базис теоретичних знань, який надає здобувачам освіти можливість швидкого та лаконічного опанування дидактичного матеріалу курсу з мінімальними часовими витратами. В методичному виданні подано основні теоретичні відомості з тем навчального курсу та перелік необхідних літературних джерел для забезпечення подальшої самопідготовки майбутніх фахівців.

ЗМІСТ

1. АНОТАЦІЯ КУРСУ

Актуальність навчального курсу «Прикладне та Web-програмування» обумовлена зростаючими вимогами до підготовки майбутніх фахівців професійної освіти, здатних застосовувати сучасні технології програмування у процесі створення прикладного програмного забезпечення. Опанування змісту дисципліни забезпечує формування у здобувачів вищої освіти системного бачення теоретичних засад розроблення прикладних програмних продуктів із використанням топових на сьогодні мов програмування Java та PHP, а також сприяє набуттю практичних умінь і навичок, необхідних для проєктування та реалізації сучасних програмних застосунків. Особливу увагу в межах дидактичного курсу приділено оволодінню технологіями створення прикладних програм для web-середовища, а також мідлетів, сервлетів і програмних рішень із використанням елементів графічного інтерфейсу користувача (GUI), що забезпечує практичну спрямованість навчання та підготовку до подальшої професійної діяльності.

Метою вивчення курсу «Прикладне та Web-програмування» є надання здобувачам вищої освіти вмінь і навичок програмування в об'єктно-орієнтовних середовищах Java та PHP; засвоєння сучасних методів та способів створення програм різної складності з поступовим набуттям практичних вмінь використання мов програмування Java та PHP для розробки прикладних комп'ютерних програм.

Завдання курсу – засвоєння студентами теоретичних основ програмування в Java та PHP; опанування майбутніми фахівцями методів створення прикладних програм за допомогою сучасних засобів комп'ютерної техніки; розвиток вмінь створювати сучасні програми як для середовища Windows, так і безпосередньо для мережі Інтернет у середовищах мов програмування в Java та PHP; вироблення навичок розробляти та застосовувати компоненти в Java та PHP при здійсненні програмування складних об'єктів з використанням баз даних.

Зміст курсу «Прикладне та Web-програмування» для підготовки студентів першого (бакалаврського) рівня вищої освіти спеціальності А5.39 (015.39) Професійна освіта (Цифрові технології) денної та заочної форм навчання передбачає теоретичну складову у вигляді блоку лекцій, лабораторні заняття та самостійну роботу здобувачів вищої освіти. Навчальний курс читається для майбутніх фахівців професійної освіти в 6-му та 7-му семестрах.

Формою підсумкового контролю здобувачів освіти у кінці 6-го семестру є залік, а у 7-му семестрі – екзамен.

У процесі вивчення курсу «Прикладне та Web-програмування» для підготовки здобувачів першого (бакалаврського) рівня вищої освіти спеціальності А5.39 (015.39) Професійна освіта (Цифрові технології) враховується взаємозв'язок з такими навчальними дисциплінами освітньо-професійної програми «Професійна освіта (комп'ютерні технології)», як: алгоритмізація і програмування з методикою навчання; фізичні основи інформаційних систем; Web Technologies and Web Design; організація баз даних та знань.

.

ЛЕКЦІЯ 1

Тема: *Вступ у прикладне та web-програмування.*

План:

1. Історія створення мови Java.
2. Платформа Java.
3. Основні версії та продукти Java.

І. Історія створення мови Java.

Поштовхом до створення мови програмування Java стала ініціатива Патріка Нотона, який у корпорації Sun Microsystems зарекомендував себе як один із найсильніших програмістів свого часу. Після наміру перейти до компанії NeXT він на прохання Скотта Мак Нілі підготував аналітичний документ, у якому окреслив основні технологічні проблеми Sun та запропонував шляхи їх подолання. У своїх пропозиціях Патрік Нотон критично оцінив архітектуру NEWS, над якою тоді працювала компанія, і водночас наголосив на доцільності переходу до більш цілісної моделі програмної розробки, що передбачала використання єдиного інструментального середовища та вдосконалення користувацьких інтерфейсів. Його ідеї знайшли підтримку серед провідних спеціалістів корпорації, насамперед Біла Джоя та Джеймса Гослінга, які мали значний авторитет у сфері системного програмування.

Подальший розвиток цієї ініціативи привів до формування окремої команди, яка розпочала роботу над новим перспективним напрямом. Основна концепція полягала у створенні апаратно-програмної платформи, орієнтованої на масового користувача, із простим інтерфейсом і широкими можливостями застосування. Для реалізації цієї ідеї у 1991 році Патрік Нотон, Джеймс Гослінг і Майк Шерідан започаткували проєкт «Green», який функціонував автономно від основних підрозділів Sun. Його метою було визначення майбутніх тенденцій розвитку комп'ютерної індустрії та створення продукту, здатного відповідати новим технологічним потребам.

У межах проєкту Джеймс Гослінг розпочав розроблення нової мови програмування як альтернативи C++, орієнтованої на підвищення надійності, безпечності та переносимості програмного коду. Початкова назва мови – Oak – була пов'язана з дубом, який ріс за вікном його робочого кабінету. Нове середовище програмування від самого початку створювалося з урахуванням можливості використання в різних електронних пристроях, зокрема в побутовій техніці, системах керування та інтерактивних платформах.

На початковому етапі результати проєкту були спрямовані на сферу інтерактивного телебачення та побутової електроніки. Для комерціалізації технології була створена компанія FirstPerson, однак висока вартість технічної реалізації та недостатня готовність ринку до таких рішень не дозволили швидко впровадити продукт у виробництво. Водночас стрімкий розвиток World Wide Web відкрив нові можливості для застосування Oak, оскільки його архітектура добре відповідала потребам мережевого середовища.

Після переорієнтації на інтернет-технології Патрік Нотон створив браузер WebRunner, який став першим середовищем для демонстрації можливостей нової платформи. Його головною особливістю була підтримка аплетів – невеликих програм, що передавалися мережею та виконувалися безпосередньо в браузері. Саме це стало принципово новим підходом, який значно розширив функціональні можливості Web-сторінок порівняно з традиційним використанням HTML.

Публічна демонстрація WebRunner засвідчила високий потенціал нової технології. Особливе враження на професійну аудиторію справили приклади інтерактивної графіки та анімації, які продемонстрували можливість виконання складних динамічних операцій безпосередньо у Web-середовищі. Це дало підстави розглядати Oak як одну з ключових технологій для подальшого розвитку мережевих застосунків.

У 1995 році, після проведення маркетингового аналізу, Oak було перейменовано на Java, а браузер WebRunner отримав назву HotJava. Офіційне представлення нової технології відбулося у травні 1995 року, коли Sun повідомила про підтримку Java у браузері Netscape Navigator. Це рішення стало важливим етапом у розвитку вебтехнологій, оскільки Java фактично інтегрувалася у глобальний інформаційний простір і заклала підґрунтя для створення платформонезалежного програмного забезпечення.

ІІ. Платформа Java.

Платформа Java є програмно-технологічним середовищем, призначеним для розроблення, компіляції, виконання та супроводу програмних застосунків на основі мови Java. Її ключова особливість полягає в забезпеченні платформної незалежності: програма, створена один раз, може виконуватися на різних операційних системах без зміни вихідного коду. Цей принцип реалізується завдяки концепції «Write Once, Run Anywhere» («написав один раз – запускай будь-де»).

Основу платформи становить віртуальна машина Java – *Java Virtual Machine*, яка виконує байт-код, отриманий після компіляції вихідної програми.

Під час компіляції Java-програма не перетворюється безпосередньо в машинний код конкретної операційної системи, а компілюється у проміжний формат – *байт-код*, який є універсальним для різних платформ. Саме JVM інтерпретує або компілює цей байт-код у машинні інструкції конкретного пристрою.

Вихідний код будь-якої програми на мові Java представляється звичайними текстовими файлами, які можуть бути створені в будь-якому текстовому редакторові, або спеціалізованому засобі розробки і мають розширення *.java*. Результат роботи компілятора зберігається в бінарних файлах з розширенням *.class*. Java-програма, що складається з таких файлів, подається на вхід віртуальної машини, яка починає їх виконувати, або інтерпретувати, оскільки сама є програмою.

До складу платформи Java входить також стандартна бібліотека класів, що містить великий набір готових компонентів для роботи з текстом, файлами, мережами, графічним інтерфейсом, базами даних та багатьма іншими ресурсами. Завдяки цьому розробник отримує можливість використовувати вже реалізовані програмні механізми без необхідності створювати їх з нуля.

У Java застосовується сувора типізація. Це означає, що будь-яка змінна і будь-який вираз має тип, відомий вже на момент компіляції. Такий підхід застосований для спрощення виявлення проблем, адже компілятор відразу повідомляє про помилки, і вказує їх розташування в коді. Пошук же виняткових ситуацій (exceptions) під час виконання програми (runtime) потребує складного тестування, при якому причина дефекту може виявитися зовсім в іншому класі.

Важливою складовою є *Java Development Kit (JDK)* – комплект засобів розробника, що включає компілятор, бібліотеки, налагоджувальні інструменти та службові програми для створення застосунків . Для запуску готових програм використовується Java Runtime Environment (JRE), яка містить JVM і базові бібліотеки виконання.

Однією з важливих переваг Java є автоматичне керування пам’яттю через механізм збирача сміття *(Garbage Collector)*, який автоматично звільняє пам’ять від об’єктів, що більше не використовуються. Це підвищує надійність програм і зменшує кількість помилок, пов’язаних із керуванням ресурсами.

Збирач сміття – це фоновий потік виконання, який регулярно переглядає існуючі об’єкти і видаляє не потрібні.

Java-платформа володіє наступними перевагами: кросплатформенність; об’єктна орієнтованість (створена ефективна об’єктна модель); звичний синтаксис C/C++; вбудована і прозора модель безпеки; орієнтація на Internet-задачі, мережеві розподілені застосування; динамічність, легкість розвитку і додавання нових можливостей; простота освоєння.

Таким чином, платформа Java являє собою цілісне програмне середовище, що поєднує мову програмування, механізми виконання, бібліотеки класів та інструменти розроблення, забезпечуючи універсальність, безпечність і високу переносимість програмних рішень.

III. Основні версії та продукти Java.

У травні 1995 року відбулося офіційне представлення платформи Java, що стало важливим етапом у розвитку сучасних програмних технологій. На момент першого оголошення користувачам були доступні бета-версії основних компонентів платформи, серед яких ключове місце займала специфікація мови Java (*Java Language Specification, JLS*). Цей документ визначав синтаксичні правила мови, систему типів даних, принципи побудови програмних конструкцій та загальні засади функціонування мови. Високий рівень опрацювання специфікації забезпечив її довготривалу актуальність: попри численні оновлення, уточнення й розширення, базові концептуальні положення залишилися практично незмінними.

Другим важливим елементом стала специфікація віртуальної машини Java – *Java Virtual Machine*, яка регламентувала правила виконання байт-коду, організацію пам’яті, завантаження класів і механізми взаємодії між програмою та середовищем виконання. Цей документ орієнтований насамперед на розробників віртуальних машин і реалізаторів платформи, тому в практичній діяльності програмістів використовується переважно опосередковано.

Основним інструментом для створення програм став Java Development Kit (JDK) – комплект засобів розроблення, який тривалий час залишався базовим середовищем роботи програміста. До його складу входили компілятор, стандартні бібліотеки класів, службові утиліти та демонстраційні приклади. Особливістю JDK було те, що він не містив власного текстового редактора, а передбачав використання вже створених вихідних файлів мовою Java. Ключовим компонентом JDK є утиліта *javac*, яка здійснює компіляцію вихідного коду у байт-код, придатний для виконання у JVM. Для запуску програм використовується команда java, що забезпечує виконання байт-коду через віртуальну машину. Важливим допоміжним засобом став також javadoc – інструмент автоматичного створення документації на основі коментарів у вихідному коді, що значно полегшувало супровід програмних проєктів.

Таким чином, початкова архітектура платформи Java вже на етапі першого представлення містила всі основні компоненти, які заклали основу її подальшого розвитку як універсального середовища програмування. Зокрема, 1-ша версія містила 8 стандартних бібліотек:

- *java.lang* – базові класи, які необхідні для роботи будь-якого застосунку;
- *java.util* – бібліотека, яка містить значну кількість корисних допоміжних класів;
- *java.applet* – класи для створення аплетів;
- *java.awt* – бібліотека мови Java для створення графічного інтерфейсу користувача (GUI);
- *java.awt.image* – додаткові класи для роботи із зображеннями;
- *java.io* – засоби мови програмування Java для роботи з потоками даних та файлами;
- *java.net* – складові мови програмування Java для роботи в мережі Internet.

Фінальна версія JDK 1.0 була випущена 23 січня 1996 року. Основні недоліки даної версії: продуктивність (перша віртуальна машина працювала дуже повільно, що було пов'язано з тим, що JVM, по суті, є інтерпретатор, який працює завжди повільніше, ніж код, що відкомпілювався); наголошувалося на досить незначних можливостях AWT, відсутності в ній роботи з сучасними базами даних тощо.

У лютому 1997 року вийшла фінальна версія JDK 1.1. Особлива увага в ній приділена продуктивності виконання програмного коду. Багато частин віртуальної машини було оптимізовано і переписано з використанням Assembler. Крім того, з жовтня 1996 року Sun розвиває новий продукт – *Just-In-Time компілятор*, завдання якого – транслювати Java байт-код програми в «рідний» код операційної системи. Було додано також багато нових важливих можливостей.

JavaBeans – технологія, оголошена ще в 1996 році, дозволяє створювати візуальні компоненти, які легко інтегруються у візуальні засоби розробки. JDBC (Java DataBase Connectivity) забезпечує доступ до баз даних. *RMI (Remote Method Invocation)* дозволяє легко створювати розподілені застосування. Було вдосконалено підтримку національних мов і систему безпеки.

Вихід наступної версії Java 1.2 багато раз відкладався, але у результаті вона настільки перевершила попередню, що її та всі подальші версії почали називати платформою Java 2. Фінальна версія була випущена 8 грудня 1998 року, і за перші вісім місяців її було завантажено більше мільйона разів.

15 червня 1999 року, на конференції розробників JavaOne компанія Sun оголосила про розділення розвитку платформи Java 2 на три напрями:

- ***Java 2 Platform, Standard Edition (J2SE)*** – призначена для використання на робочих станціях і персональних комп'ютерах. Standard Edition –

основа технології Java і прямий розвиток JDK (засіб розробника був перейменований в j2sdk);

- ***Java 2 Platform, Enterprise Edition (J2EE)*** – містить засоби для створення складних, високонадійних, розподілених серверних застосувань;
- ***Java 2 Platform, Micro Edition (J2ME)*** – усічена версія Standard Edition, призначена для задоволення жорстких апаратних вимог невеликих пристроїв, таких як кишенькові комп'ютери і стільникові телефони.

Вихід наступних версій Java подамо у хронологічному порядку їх появи:

- 8 травня 2000 р. – вийшла J2SE 1.3 (*Kestrel*).
- 13 лютого 2002 р. – вийшла J2SE 1.4.0 (*Merlin*).
- 29 вересня 2004 р. – вийшла J2SE 5.0 (1.5.0) (*Tiger*).
- 11 грудня 2006 р. – вийшла Java SE 6 (1.6.0) (*Mustang*).
- 28 липня 2011р. – вийшла Java SE 7 (1.7.0) (*Dolphin*).
- 18 березня 2014 р. - Java SE 8 (1.8.0) (неформальна назва Spider (в деяких джерелах Octopus)) – додано підтримку лямбда-виразів.

У зв'язку зі складнощами в модулярізації (проект Jigsaw) реліз версії Java SE 9 кілька разів відкладався: спочатку дата була перенесена на 23 березня 2017 року, потім – на 27 липня 2017 року, а потім – на 21 вересня 2017 року.

Open JDK 10 був випущений 20 березня 2018 року, в новий випуск увійшло 12 нововведень. Серед іншого:

- JEP 286: виведення типів для локальних змінних;
- JEP 317: експериментальний JIT компілятор на Java;
- JEP 310: можливість спільного використання класів різними додатками для Java.

Відтепер компанія планує випускати LTS реліз раз на три роки, демонстрації нових функцій (англ. feature release) кожні шість місяців, а оновлення — щокварталу. Після релізу Java 11 у вересні 2018 року наступним LTS релізом став Java 17, випуск якої відбувся 14 вересня 2021 року.

Актуальними версіями Java на 2025-2026 роки є версія 21 (LTS) та 17 (LTS), які є основними для розробки. Також широко використовуються, підтримуються LTS-версії 8 та 11, що забезпечують стабільність, а новіші, наприклад 20-22, виходять для впровадження функцій.

Основні LTS (Long-Term Support) версії Java, що зараз використовуються:

- Java 21 (LTS) – наразі найновіша довгострокова версія (остання 21.0.2).
- Java 8 (LTS) – версія мови Java, яка підтримується до грудня 2030 року.

Java активно використовується для розробки веб-додатків (BackEnd), Android-додатків, банківських систем та IoT.

ЛЕКЦІЯ 2

Тема: *Синтаксис Java.*

План

1. Структура програми на мові Java.
2. Коментарі.
3. Константи.

I. Структура програми на мові Java.

Структура програми мовою Java має чітко визначену організацію, що забезпечує зрозумілість коду, його модульність і можливість подальшого розширення. Кожна Java-програма складається з одного або кількох класів, оскільки основною одиницею побудови в цій мові є клас. Навіть найпростіша програма повинна бути розміщена всередині класу.

Типова структура програми включає такі складові:

- оголошення пакета (package);
- імпорт необхідних бібліотек (import);
- оголошення класу;
- головний метод main();
- оператори та інструкції виконання програми.

Наведемо класичний приклад (програма виводить на екран рядок *Hello, world*):

```
class POIF{
public static void main(String[] args){
System.out.println("Hello, World!");
}}
```

Приклад відображає ряд суттєвих особливостей мови Java:

1. Початок класу позначається службовим словом *class*, за яким іде *ім'я класу*. Все, що міститься в класі, записується в фігурних дужках і складає *тіло класу (class body)*.

2. Всі дії виконуються за допомогою методів обробки інформації. Методи розрізняються по іменах. Один із методів обов'язково повинен називатися *main*, з нього починається виконання програми. В нашій програмі тільки один метод, тому ім'я йому *main*.

Метод завжди видає в результат (частіше говорять, повертає (returns) тільки одне значення, тип якого обов'язково вказується перед іменем методу. Метод може і не повертати ніякого значення, виконуючи роль процедури, як у нашому випадку. Тоді замість типу значення записується слово *void*.

3. Після імені методу в дужках, через кому, перечислюються аргументи – або параметри методу. Для кожного аргумента вказується його тип і, через пробіл, ім'я. В прикладі тільки один аргумент, його тип – масив, що складається з рядків символів. Рядок символів – це вбудований в Java API тип String, а квадратні дужки – ознака масиву. Ім'я масиву може бути довільним, в прикладі вибрано ім'я *args*.

4. Перед типом значення, що повертається методом, можуть бути записані *модифікатори*. В прикладі їх два: слово *public* – означає, що цей метод доступний звідусіль; слово *static* – забезпечує можливість виклику метода *main()* на початку виконання програми. Модифікатори взагалі необов'язкові, але для методу *main()* вони необхідні.

Єдина дія, яку виконує метод *main()* в прикладі, полягає у виклику іншого методу зі складним іменем *System.out.println(),* і передачі йому на опрацювання одного аргументу, текстової константи "Hello, World!".

Складне ім'я *System.out.println()* означає, що в класі *System*, який входить в *Java API*, визначається змінна з іменем *out*, котра містить екземпляр одного із класів *Java API*, класу *PrintStream*. В ньому є метод *println()*. Дія методу *println()* полягає у виведенні свого аргументу в вихідний потік, пов'язаний, як правило, з виведенням інформації на екран текстового терміналу.

Після виведення курсор переходить на початок наступного рядка екрану, на це вказує закінчення *ln*, а слово *println* – скорочення слів *print line*. У складі об'єкта *out* є також метод *print()*, що залишає курсор в кінці рядка.

Мова Java розрізняє прописні і заглавні літери. Свої імена можна записувати по різному (наприклад, helloworld чи Helloworld), але між Java-програмістами заключено договір під іменем «Code Conventions for the Java Programming Language». Відмітимо деякі пункти з нього:

- імена класів починаються з великої літери; якщо ім'я містить декілька слів, то кожне слово починається із великої літери;
- імена методів і змінних починаються із прописної літери; якщо ім'я містить декілька слів, то кожне наступне слово теж починається з прописної літери;
- імена констант записуються повністю усіма великими літерами; якщо ім'я константи утворює сукупність декількох слів, то між ними ставиться знак підкреслення.

Серед обов'язкових правил написання програмного коду можна виокремити наступні:

- кожна команда завершується символом ';' ;
- код чутливий до регістру;

– фігурні дужки визначають межі блоків;
– кожен файл містить один public-клас.

Таким чином, структура Java-програми базується на класах, методах і чітко визначених синтаксичних правилах. Такий підхід забезпечує надійність, переносимість і масштабованість програмних рішень.

ІІ. Коментарі.

Під час написання програмного коду важливе значення мають *коментарі* – текстові пояснення, які не впливають на виконання програми, оскільки не обробляються компілятором. Їх використання дає змогу зробити програму зрозумілішою для розробника, полегшує аналіз логіки алгоритму та спрощує подальше супроводження коду. Особливо корисними коментарі є в процесі налагодження, коли окремі оператори тимчасово виключаються з виконання шляхом їх «закоментування».

У мові Java застосовуються два основні типи коментарів.

Однорядковий коментар починається двома похилими рисками // і поширюється до кінця поточного рядка. Такий тип використовується для коротких пояснень або тимчасового вимкнення окремої команди:

```
System.out.println("Hello"); // Виведення повідомлення
```

Багаторядковий коментар починається символами /* і завершується символами */. Усередині такого блоку можна розміщувати текст на кількох рядках, що зручно для детального опису окремих частин програми:

```
/*
Це багаторядковий коментар.
Він використовується для пояснення
роботи кількох операторів програми.
*/
```

Правильне використання коментарів сприяє підвищенню якості програмного коду, оскільки робить його більш читабельним, структурованим і придатним до колективної розробки. Програму з хорошими коментарями називають *самодокументованою.*

У мову Java введені коментарі третього типу, а в склад *JDK* – програму *javadoc*, що поміщає ці коментарі в окремі файли формату *HTML* і створює гіперпосилання між ними: похилою рискою і двома зірочками підряд, без пробілів, /** починається коментар, котрий може займати декілька рядків до зірочки з однією похилою рискою */ і опрацьовується програмою javadoc. В

такий коментар можна вставить вказівки програмі javadoc, котрі починаються символом @. Саме так створюється документація в JDK.

Приклад усіх описаних вище типів коментарів демонструє наступний лістинг Java-коду:

```
class HelloPOIF{
/**
* Пояснення змісту і особливостей програми...
* @author Ім'я Прізвище (автора)
* @version 1.0 (версія програми)
*/
// HelloPOIF — це ім'я
// Наступний метод починає виконання програми
public static void main(String[] args){
/* Наступний метод просто виводить свій
аргумент  на екран дисплея */
System.out.println("Перша програма гр.ПО та ІФ!");
//System.out.println виведе текст: Перша програма
гр.ПО та ІФ!
}}
```

III. Константи

Константи – це дані, які зафіксовані під час виконання програми і не можуть бути змінені.

Вони використовуються для збереження фіксованих даних, що повинні залишатися незмінними протягом усього часу роботи застосунку. Використання констант підвищує надійність програмного коду, робить його зрозумілішим і полегшує супровід програмних проєктів.

Для створення константи в Java застосовується ключове слово final. Якщо змінна оголошена як final, після присвоєння значення його вже не можна змінити.

```
final double PI = 3.14159;
```

У цьому прикладі константа PI зберігає значення числа π і не може бути змінена в подальшому.

Основні правила оголошення констант:

- використовується ключове слово final;
- значення задається один раз;
- повторне присвоєння неможливе;
- назви констант зазвичай записуються великими літерами через символ підкреслення.

```
final int MAX_SIZE = 100;
final String UNIVERSITY_NAME = "ЛНТУ";
```

Константи можуть належати до будь-якого примітивного типу:

```
final int A = 10;
final double RATE = 5.5;
final char LETTER = 'A';
final boolean FLAG = true;
```

Символьні та рядкові константи:

```
final char SYMBOL = '*';
final String MESSAGE = "Java programming";
```

Рядкові константи широко використовуються для фіксованих текстових повідомлень.

Друковані символи можна записувати в апострофах: 'a', 'N', '?'.

Керуючі символи записуються в апострофах з оберненою похилою рискою:

'\\' – обернена похила риска;

'\"' – лапка;

'\'' – апостроф.

Код будь-якого символу з десятковим кодуванням від *0* до *255* можна задати, записавши його не більше ніж трьома цифрами у вісімковій системі числення в апострофах після оберненої похилої риски: *'\123'* – буква *S*, *'\346'* – буква *ц*. Не рекомендується використовувати цю форму запису для друкованих і керуючих символів, перечислених у попередньому пункті, оскільки компілятор відразу переведе вісімковий запис у вказану вище форму. Найбільший код *'\377'* – десяткове число *255*.

Код будь-якого символу в кодуванні *Unicode* набирається в апострофах після оберненої похилої риски і латинської літери *u* рівно чотирма шістнадцятковими цифрами: *'\u0053'* – буква *S*, *'\u0416'* – знак питання *?*.

Символи зберігаються в форматі типу *char*.

Якщо константа повинна бути спільною для всіх об’єктів класу, застосовується комбінація static final:

```
public static final double GRAVITY = 9.81;
```

Такі константи належать класу, а не окремому об’єкту.

Вбудовані константи Java. Java має готові константи у стандартних класах **`Math.PI`** і **`Math.E`**. Наприклад:

```
System.out.println(Math.PI);
```

Окрім оголошених final-констант, у Java існують літерали – значення, безпосередньо записані в коді:

Керуючі символи та коди записуються в рядках так само як і з оберненою похилою рискою, але без апострофів, та викликають ті ж самі дії. Рядки можуть розташовуватися лише в одній стрічці вихідного коду. Не можна також відкриваючі лапки ставити в одному рядку, а закриваючі – в наступному, наприклад:

```
"Цей рядок \n з переносом"
"\"Волинь\" — Чемпіон!"
```

Для рядкових констант визначена також операція з'єднання, яка позначається плюсом (+), наприклад: **"З'єднання"+"рядків"** дає в результаті один цілісний рядок з двох слів, а саме: **З'єднання рядків**.

ЛЕКЦІЯ 3

Тема: *Примітивні типи даних та операції над ними.*

План:

1. Класифікація примітивних типів даних.
2. Операції над типами даних та виразами.

І. Класифікація примітивних типів даних.

Всі типи вихідних даних, вбудованих в мову програмування Java, діляться на дві групи: примітивні типи та посилкові типи. Посилкові типи діляться на масиви (arrays), класи (classes) та інтерфейси (interfaces).

Основні відмінності між примітивними та посилковими типами даних включають таке:

- примітивні типи даних займають фіксовану кількість пам'яті, визначену для кожного типу, і їхні значення зберігаються в стеку пам'яті;
- посилкові типи даних вимагають більше пам'яті, оскільки вони містять додаткові дані, такі як заголовок об'єкта, посилання на методи тощо;
- значення посилкових типів даних зберігаються в пам'яті, а змінні містять тільки посилання на ці значення;
- примітивні типи даних передаються за значенням, тобто копіюються цілком під час передачі в метод або присвоєння іншій змінній;
- посилкові типи даних передаються за посиланням, тобто копіюється тільки посилання на об'єкт, а не сам об'єкт.

Важливо розуміти відмінності між примітивними і посилковими типами даних, оскільки це впливає на спосіб роботи з ними, передачу аргументів у методи і зберігання значень у пам'яті.

В цілому, примітивні типи даних відіграють важливу роль у програмуванні на Java. Вони надають основу для зберігання і маніпулювання різними видами даних, такими як числа, символи і логічні значення. Примітиви в Java мають простий синтаксис і ефективні у використанні завдяки своїй безпосередній природі, що зберігається.

Примітивних типів всього вісім. Їх можна розділити на логічний тип boolean і числовий (numeric). До числових типів відносяться цілі (integer) і дійсні (floating-point) типи. Цілих типів п'ять: byte, short, int, long, char.

Примітиви в Java є основою мови і надають простий спосіб зберігання та оперування даними. Вони можуть бути використані в багатьох аспектах програмування, від обчислень до керування логікою програми.

Символи можна використовувати скрізь, де вживається тип int, тому *JLS* причисляє їх до цілих типів. Наприклад, їх можна використовувати в арифметичних обчисленнях, скажімо, можна написати 2 + 'b', тоді до двійки буде доданий ASCII код 98 літери 'b' і в результаті додавання одержимо 100.

Дійсних типів два: float і double. Оскільки по імені змінної неможливо визначити її тип, всі змінні обов'язково повинні бути описані перед їх використанням. Опис полягає в тому, що записується ім'я типу, потім, через пробіл, список імен змінних, розділених комою. Для всіх або деяких змінних можна вказати початкові значення після знаку рівності, котрими можуть бути будь-які константні вирази того ж типу. Описи кожного типу закінчується крапкою з комою. В програмі може бути скільки завгодно описів кожного типу.

Значення логічого типу boolean виникають у результаті різних порівнянь, і використовуються в основному в умовних операторах і операторах циклів. Логічних значень всього два: true (істина) і false (хиба). Це службові слова Java. Опис змінних цього типу виглядає так:

```
boolean b = true, bb = false, bool2;
```

Над логічними даними можна виконувати операції присвоювання, наприклад, bool2 = true, в тому числі й сумісні з логічними операціями; порівняння на рівність b == bb і на нерівність b != bb, а також логічні операції:

- заперечення (NOT) ! (позначається знаком оклику);
- кон'юнкція (AND) & (амперсанд);
- диз'юнкція (OR) | (вертикальна риска);
- виключне АБО (XOR) ^ (каре).

Вони виконуються над логічними даними, їх результатом буде також

логічне значення true або false. Нагадаємо таблицю логічних операцій.

Таблиця 3.1 – Логічні операції

b1	b2	!b1	b1&b2	b1|b2	b1^b2
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

Ці правила можна виразити так:

- заперечення змінює значення істинності;
- кон'юнкція істинна, тільки якщо обидва операнди істинні;
- диз'юнкція хибна, тільки якщо обидва операнди хибні;
- виключне АБО істинне, тільки якщо значення операндів різні.

Специфікація мови Java визначає розрядність (кількість байт, що відводяться для зберігання значень типу в оперативній пам'яті) і діапазон значень кожного типу. Для цілих типів вони наведені в таблиці 3.2.

Таблиця 3.2 – Цілі типи

Тип	Розрядність (байт)	Діапазон
byte	1	от -128 до 127
short	2	от -32768 до 32767
int	4	от -2147483648 до 2147483647
long	8	от -9223372036854775808 до 9223372036854775807
char	2	від '\u0000' до '\uFFFF' , в десятковій формі від 0 до 65535

Хоча тип *char* займає два байти, в арифметичних обчисленнях він працює як тип *int*, йому виділяється 4 байти, два старших байти заповнюються нулями.

Приклади визначення змінних цілих типів:

```
byte b1 = 50, b2 = -99, b3;
short det = 0, ind = 1;
int i = -100, j = 100, k = 9999;
long big = 50, veryBig = 2147483648L;
char c1 = 'A', c2 = '?', newLine = '\n';
```

У Java існують класи-обгортки (wrapper classes), які представляють обгортку над примітивними типами даних. Класи-обгортки дають змогу використовувати примітивні типи даних у контексті об’єктів. Вони також надають методи для перетворення значень, виконання арифметичних операцій, роботи з бітами та інших операцій.

Одна з головних переваг класів-обгорток полягає в їхній можливості роботи з колекціями (наприклад, списками, наборами та відображеннями), які вимагають об’єктів як елементів. Також вони можуть бути корисними під час використання узагальнень (generics) у Java.

Однією з особливостей класів-обгорток є автопакування (autoboxing) та авторозпакування (unboxing). Автопакування дає змогу автоматично перетворювати примітивні типи у відповідні класи-обгортки, а авторозпакування виконує зворотну операцію – перетворення об’єктів класів-обгорток у примітивні типи. Завдяки цим механізмам, можна використовувати примітивні типи в контексті об’єктів без явного перетворення.

Приклад автопакування та авторозпакування:

```
```java
Integer number = 10; // автоупаковка: примітивний
тип int перетворюється на об'єкт Integer
int value = number; // авторопакування: об'єкт
Integer перетворюється на примітивний тип int
```
```

Автопакування та авторозпакування полегшують роботу з примітивними типами даних, спрощують код і роблять його більш читабельним. Вони автоматично виконуються компілятором Java, що дає змогу програмісту використовувати примітиви та класи-обгортки взаємозамінно в більшості випадків.

II. Операції над типами даних та виразами.

Над цілими типами можна виконувати масу операцій. Їх набір визначився в мові С, він виявився зручним і переходить з мови в мову майже без змін. Особливості застосування цих операцій в мові Java показані на прикладах.

Всі операції, котрі виконуються над цілими числами, можна розділити на наступні групи:

1. Арифметичні операції:

- додавання + (плюс);
- віднімання - (дефіс);
- множення * (зірочка);
- ділення / (похила риска – слеш);

- залишок від ділення % (процент);
- інкремент (збільшення на одиницю) ++;
- декремент (зменшення на одиницю) - -.

Між спареним плюсами і мінусами не можна залишати пробіли. Додавання, віднімання і множення цілих значень виконуються в звичному вигляді, а ділення цілих значень в результаті дає знову ціле (так зване «ціле ділення»), наприклад, 5/2 дасть в результаті 2, а не 2.5, а 5/(-3) дасть -1.

Операція ділення по модулю визначається так: a%b = a - (a / b) * b; наприклад, 5%2 дасть в результаті 1, а 5% (-3) дасть 2, тому що 5 = (-3)*(-1)+2, але (-5)%3 дасть -2, оскільки -5 = 3*(-1)-2.

Операції інкременту та декременту означають збільшення або зменшення значення змінної на одиницю і застосовуються тільки для змінних, а не для констант чи виразів. Не можна, наприклад, написати 5++ або (a + b)++.

Ці операції можна записати і перед змінною: ++i, -- j. При першій формі запису *(постфіксній)* у виразі приймає участь старе значення змінної, і лише потім відбувається збільшення чи зменшення її значення. При другій формі запису *(префіксній)* спочатку зміниться змінна і її нове значення буде приймати участь у виразі.

Наприклад, якщо k=9999, то вираз (k++) + 5 дасть в результаті 10004, а змінна k прийме значення 10000. Але в тій же ситуації (++k) + 5 дасть 10005, а змінна k стане рівною 10000.

2. Операції порівняння (більше >; менше <; більше або дорівнює >=; менше або дорівнює <=; рівно ==; не рівно !=)**.**

Спарені символи записуються без пробілів, їх не можна переставляти місцями. Так запис => буде невірним. Результат операції порівняння – логічне значення: true, в результаті, наприклад, порівняння 3 != 5; або false, наприклад, в результаті порівняння 3 == 5.

Запис a < x < b в мові Java приведе до повідомлення про помилку, оскільки перше порівняння, a < x, дасть true або false, а Java не знає, більше це, ніж b, чи менше. Слід написати вираз (a < x) && (x < b), причому тут дужки можна опустити, написати просто a < x && x < b.

3. Побітові операції. Інколи потрібно змінити значення окремих бітів в цілих даних. Це виконується за допомогою побітових (bitwise) операцій шляхом накладання маски. В мові Java є чотири побітові операції:

- доповнення (complement) ~ (тильда);
- побітова кон’юнкція (bitwise AND) &;
- побітова диз’юнкція (bitwise OR) |;

– побітове виключне АБО (bitwise XOR) ^.

4. Операції присвоювання. Проста операція присвоєння записується знаком рівності =, зліва від котрого стоїть змінна, а справа вираз, сумісний з типом змінної. Крім простої операції присвоювання є ще 11 *складних* операцій присвоювання: +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=; >>>=. Символи записуються без пробілів, а також не можна переставляти їх місцями.

5. Умовна операція. Вона має три операнди: спочатку записується довільний логічний вираз, потім ставиться знак питання, потім два довільних вирази, розділених двокрапкою:

```
x < 0 ? 0 : x
x > y ? x- y : x + y
```

Спочатку обчислюється логічний вираз. Якщо одержано значення true, то обчислюється перший вираз після знака ? і його значення буде результатом всієї операції. Останній вираз при цьому не обчислюється. Якщо ж отримане значення false, то обчислюється тільки останній вираз, а його значення буде результатом операції.

Із констант і змінних, операцій над ними, викликів методів і дужок складаються *вирази* (expressions).

При обчисленні виразу виконуються чотири правила:

1. Операції одного пріоритету обчислюються зліва направо: x+y+z обчислюється як (x+y)+z. Виняток: операції присвоювання виконуються справа наліво: x = y = z обчислюється як x = (y = z).
2. Лівий операнд обчислюється раніше правого.
3. Операнди повністю обчислюються перед виконанням операції.
4. Перед виконанням складної операції присвоювання значення лівої частини зберігається для використання в правій частині.

Нехай **int a = 3, b = 5.** Тоді результатом виразу **b + (b = 3)** буде число **8**; але результатом виразу **(b = 3) + b** буде число 6. Вираз b += (b = 3) дасть в результаті 8, тому що обчислюється як перший із наведених вище виразів.

Розуміння примітивних типів даних у Java важливе для розробника, оскільки це дає змогу ефективно використовувати ресурси пам’яті, керувати даними та створювати ефективні алгоритми. Знання примітивів також необхідне для роботи з методами та класами, які оперують цими типами даних.

ЛЕКЦІЯ 4

Тема: *Оператори мови Java.*

План:

1. Класифікація операторів мови Java.
2. Умовний оператор if.
3. Оператори циклу while, do-while та for.
4. Оператор варіанту switch.
5. Оператори переходу break, continue і return.

І. Класифікація операторів мови Java.

Мова програмування Java широко застосовується у сфері розробки програмного забезпечення. Важливою складовою цієї мови є оператори, які надають розробникам інструменти для управління потоком виконання програми.

Оператор – це символ або поєднання символів, який виконує певну дію. Оператори дають змогу виконувати арифметичні операції, ухвалювати рішення на основі умов і багато іншого. У Java існує широкий набір операторів, кожен з яких виконує свою специфічну функцію. Набір операторів мови Java включає:

- оператори описування змінних і інших об’єктів;
- оператори-вирази;
- оператори присвоювання;
- умовний оператор if;
- три оператори циклу while, do-while, for;
- оператор варіанту switch;
- оператори переходу break, continue і return;
- блок {};
- пустий оператор (крапка з комою).

Будь-який оператор закінчується крапкою з комою. Можна поставити крапку з комою в кінці будь-якого виразу, тоді він стане оператором. Але це має зміст тільки для операцій присвоювання, інкременту, декременту і виклику методів. В решті випадків це не має змісту, тому що обчислене значення виразу буде втрачено.

Крапка з комою в Java не розділяє оператори, а являється частиною оператора. Лінійне виконання алгоритму забезпечується послідовним записом операторів.

Оператор ***блок*** містить в собі нуль або декілька операторів з метою використання їх як один оператор у тих місцях, де за правилами мови можна

записати тільки один оператор. Наприклад, {x = 5; y = 2;}. Можна записати і пустий блок, просто пару фігурних дужок {}.

Блоки операторів часто використовуються для обмеження області дії змінних і просто для легшого читання тексту програми.

Крапка з комою в кінці будь-якої операції присвоювання перетворює її в оператор присвоювання. Різниця між операцією і оператором присвоювання носить лише теоретичний характер. Присвоювання частіше застосовується як оператор, а не як операція.

Перевантаження операторів – це можливість визначення поведінки оператора для користувацьких типів даних. На відміну від інших мов, таких як C++, Java не дозволяє перевантажувати оператори довільно. У Java існує кілька операторів, для яких можна зробити перевантаження. Це арифметичні оператори (+, -, *, /), оператори порівняння (==, !=, <, >) та оператори присвоювання (=, +=, -= тощо).

Перевантаження операторів дає змогу програмістам створювати більш інтуїтивний і зручний інтерфейс для роботи з користувацькими типами даних. Наприклад, якщо у вас є клас «Vector», ви можете перевантажити оператор додавання (+), щоб об’єднати два вектори.

Однак, важливо пам’ятати, що перевантаження операторів має бути виправданим і слідувати певним правилам. У Java існують обмеження і правила для перевантаження операторів, і не всі оператори можуть бути перевантажені.

Наприклад, існує міф, що перевантажити можна абсолютно будь-який оператор Java. Насправді, в Java перевантаження доступне тільки для обмеженого набору операторів, більшість з яких перераховані вище.

II. Умовний оператор if.

Умовний оператор *(if-then-else statement)* у мові Java записується так:

```
if (логічний вираз) оператор_1 else оператор_2
```

Спочатку обчислюється логічний вираз. Якщо його результат *true*, то діє *оператор_1* і на цьому дія умовного оператора завершується. *Оператор_2* не діє, а далі виконується наступний за *if* оператор. Якщо результат false, то діє *оператор_2,* при цьому *оператор_1* взагалі не виконується.

Умовний оператор може бути скороченим (*if-then statement*):

```
if (логічний вираз) оператор_1
```

Тут і у випадку, коли значення логічного виразу рівне *false* – не виконується нічого.

Синтаксис мови не дозволяє записувати декілька операторів ні у гілці *then*, ні у гілці *else*. При необхідності створюється блок операторів у фігурних дужках. «Code Conventions» рекомендує завжди використовувати фігурні дужки і розташовувати оператор в декількох рядках з відтступами, як в наступному прикладі:

```
if (a < x) {
x = a + b; } else {
x = a - b;
}
```

Це полегшує додавання операторові в кожну гілку при зміні алгоритму. Часто одним із операторів є знову умовний оператор, наприклад:

```
if (n == 0){
sign = 0;
} else if (n<0){
sign = -1;
} else {
sign = 1;
}
```

Взагалі не варто використовувати складні вкладені умовні оператори. Перевірка умов займає багато часу. По можливості краще скористатися логічними операціями, наприклад, попередній вираз можна записати так:

```
if (ind >= 10 && ind <= 20) x = 0; else x = 1;
```

III. Оператори циклу while, do-while та for.

Основний оператор циклу – *оператор while* – виглядає так:

```
while (логічний вираз) оператор
```

Спочатку обчислюється *логічний вираз*. Якщо його значення *true*, то виконується оператор, що утворює цикл. Потім знову обчислюється логічний вираз і діє оператор, і так до тих пір, поки не набуде значення *false*. Якщо логічний вираз з самого початку рівний *false*, то *оператор* не буде виконуватися ні разу. Попередня перевірка забезпечує безпеку виконання циклу, дозволяє уникнути переповнення, ділення на нуль і інші неприємності. Тому оператор *while* являється основним, а в деяких мовах і єдиним оператором циклу.

Оператор в циклі може бути і пустим, наприклад, наступний фрагмент коду обчислює суму членів гармонічного ряду до тих пір, поки вона досягне значення 10.

```
int i = 0;
double s = 0.0;
```

```
while((s += 1.0/++i)<10);
```

Можна організувати і нескінчений цикл:

```
while (true) оператор
```

Звичайно, з такого циклу потрібно передбачити якийсь вихід, наприклад, за дпопомогою оператора *break*. Якщо в цикл треба включити декілька операторів, то слід створити блок операторів {}.

Другий з операторів циклу – *оператор do-while* – має вигляд:

```
do оператор while (логічний вираз)
```

Спочатку виконується *оператор*, а потім відбувається обчислення *логічного виразу*. Цикл виконується, поки *логічний вираз* залишається рівним *true*.

В циклі *do-while* перевіряється умова продовження, а не закінчення циклу. Суттєва різниця між цими двома операторами циклу тільки в тому, що в циклі *do-while* оператор обов'язково виконується хоча б один раз.

Синтаксис оператора for виглядає так:

```
for(список виразів_1;логічний вираз;список виразів_2)оператор
```

Перед виконанням циклу обчислюється *список виразів_1*. Це нуль або декілька виразів, перечислених через кому. Вони обчислюються зліва направо, і в наступному виразі уже можна використовувати результат попереднього виразу. Як правило задаються початкові значення змінних циклу.

Потім обчислюється *логічний вираз*. Якщо він істинний, *true*, то діє *оператор*. Далі обчислюються зліва направо вирази із *списку виразів_2*. Потім знову перевіряється *логічний вираз*. Якщо він істинний, то виконується *оператор* і *список виразів_2* і т. д. Як тільки *логічний вираз* стане рівним *false*, виконання циклу закінчується.

Будь-яка частина оператора *for* може бути відсутня: цикл може бути пустим, вираз в заголовку теж, при цьому крапки з комою зберігаються. Так можна задати нескінчений цикл:

```
for (;;) оператор
```

Хоча в операторі *for* закладені великі можливості, використовується він, головним чином, для обчислень, коли їх число заздалегідь відоме.

IV. Оператор варіанту switch.

Оператор варіанту *switch* організує розгалуження за декількома напрямками. Кожна гілка відмічається константою або константним виразом

якого-небудь цілого типу (крім long) і вибирається, якщо значення певного виразу співпадає з цією константою. Вся конструкція виглядає так:

```
switch (цілВир){
case констВир1: оператор1
case констВир2: оператор2
case констВирN: операторN
default: операторDef
}
```

Вираз в дужках *цілВир* може бути типу *byte, short, int, char*, але не *long.* Цілі числа або цілочисельні вирази, складені із констант, *констВир* теж не повинні мати тип *long*.

Оператор варіанту виконується так. Всі константні вирази обчислюються заздалегідь, на етапі компіляції, і повинні мати відмінні один від одного значення. Спочатку обчислюється цілочисельний вираз *цілВир*. Якщо він співпадає з однією із констант, то виконується оператор, помічений цією константою. Потім виконуються всі наступні оператори, включаючи і *операторDef,* після чого робота оператора варіанту закінчується.

Якщо ж жодна константа не рівна значенню виразу, то виконується *операторDef* і всі наступні оператори. Тому гілка *default* повинна записуватися останньою. Гілка *default* може бути відсутня, тоді в цій ситуації оператор варіанту взагалі нічого не робить.

Таким чином, константи у варіантах *case* відіграють роль лише міток, своєрідних точок входу в оператор варіанту, а далі виконуються всі інші оператори в порядку їх запису.

Після виконання одного варіанта оператор switch продовжує иконувати всі інші варіанти. Найчастіше потрібно «пройти» тільки одну гілку операторів. В такому випадку використовується оператор *break*, який відразу зупиняє виконання оператора *switch*.

Часто потрібно виконати один і той же оператор в різних гілках *case*. В цьому випадку ставимо декілька міток *case* підряд, наприклад:

```
switch(dayOfWeek){
case 1: case 2: case 3: case 4:case 5:
System.out.println("Week-day"); break;
case 6: case 7:
System.out.println("Week-end"); break;
default:
System.out.printlnt("Unknown day");
}
```

V. Оператори переходу break, continue і return.

У тих випадках, коли необхідно пропустити виконання деяких операторів програми, в Java використовуються оператор переривання *break* і оператор продовження *continue*.

Оператор ***continue*** використовується тільки в операторах циклу. Він має дві форми. Перша форма складається тільки із слова *continue* і здійснює негайний перехід до наступної ітерації циклу. В черговому фрагменті коду оператор *continue* дозволяє обійти ділення на нуль:

```
for (int i = 0; i < N; i++){
if (i == j) continue;
s += 1.0 / (i - j);
}
```

Друга форма містить мітку:

```
continue мітка
```

Мітка записується, як і всі ідентифікатори Java, із літер, цифр і знаку підкреслення, але не вимагає ніякого опису. Мітка ставиться перед оператором або відкриваючою фігурною дужкою і відокремлюється від них двокрапкою. Так виходить помічений оператор або помічений блок.

Якщо мітка опущена, оператор *continue* передає управління в самий кінець тіла циклу (після останнього його оператора) і, якщо контрольний вираз, обчислений після того, як черговий прохід тіла циклу був примусово завершений оператором **continue**, буде рівний **true**, виконання циклу продовжиться.

Оператор ***break*** використовується в операторах циклу і операторові варіанту для негайного виходу із цих конструкцій. Синтаксис оператора:

```
break мітка
```

Мітка є звичайним ідентифікатором Java, за яким слідує двокрапка. Мітка може бути опущена – в цьому випадку управління передається за межі циклу або оператора вибору, що містить даний оператор *break*.

Наступна схема пояснює конструкцію:

```
M1: { // Зовнішній блок
M2: { // Вкладений блок — другий рівень
M3: { // Третій рівень вкладеності...
if (щось трапилось) break M2;
// Якщо true, то тут нічого не виконується
}
// Тут теж нічого не виконується
}
```

```
// Сюди передається управління
}
```

Оператор ***return*** призначений для повернення управління з методу, що не викликається, в той, що викликається. Якщо в послідовності операторів виконується *return*, то управління негайно (якщо це не обумовлено особливо) передає управління в метод, що викликається.

Синтаксис оператора *return*:

return *вираз*

Оператор *return* може мати, а може й не мати аргументів. Якщо аргументи відсутні, то цей оператор може бути використаний для повернення управління тільки в методах з кваліфікатором *void*.

Якщо при оголошенні методу використаний тип *void*, то *return* не може мати аргументів, інакше, буде отримана помилка часу компіляції.

Якщо як аргумент *return* використаний вираз:

```
return (x*y +10) /11;
```

то, спочатку буде виконано вираз, а потім результат його виконання буде переданий в метод, що викликається. У випадку якщо вираз буде завершений не природним чином, то і оператор *return* буде аналогічно завершений. Наприклад, якщо під час виконання виразу в операторові *return* виникне виключна ситуація то і *return* не буде виконаний так, як це очікувалося. Тобто виняткова ситуація буде оброблена чи буде викликана в самому методі.

ЛЕКЦІЯ 5

Тема: *Робота з масивами в Java.*

План:

1. Опис масивів.
2. Одновимірні масиви в мові програмування Java.
2. Багатовимірні масиви.

І. Опис масивів

У мові Java ***масив*** – це структурований тип даних, призначений для збереження впорядкованої сукупності елементів одного типу під спільним ім’ям. Кожен елемент масиву має власний порядковий номер – індекс, за допомогою якого здійснюється доступ до його значення. Індексація в Java починається з нуля, тобто перший елемент має індекс 0, другий – 1 і т.д.

Особливості масивів у Java

– усі елементи одного типу;
– розмір задається один раз;
– елементи автоматично ініціалізуються значеннями за замовчуванням;
– масив є об'єктом.

Опис масивів проходить в три етапи:

Перший етап – *оголошення* (declaration). На цьому етапі визначається тільки змінна типу *посилання* (reference) *на масив,* і містить тип масиву. Для цього записується ім'я типу елементів масиву, квадратними дужками вказується, що оголошується посилання на масив і перечислюються імена змінних, наприклад:

```
double[] a, b;
```

Тут визначені дві змінні – посилання *a* і *b* на масиви типу *double*.

Можна поставити квадратні дужки і безпосередньо після імені. Це зручно робити серед визначень звичайних змінних:

```
int I = 0, ar[], k = -1;
```

В прикладі визначені дві змінні цілого типу *i* і *k*, а також оголошено посилання на цілочисельний масив *ar*.

Другий етап – *визначення* (installation). На цьому етапі вказується кількість елементів масиву, це називається його *довжиною,* виділяється місце для масиву в оперативній пам'яті, змінна-посилання одержує адресу масиву. Всі ці дії виконуються ще однією операцією мови Java – операцією *new тип,* що виділяє місце в оперативній пам'яті для об'єкта, вказаного в операції типу і повертає в якості результату адресу цього місця. Наприклад:

```
a = new double[5];
b = new double[100];
ar = new int[50];
```

Індекси масивів завжди починаються з 0. Масив *a* складається із п'яти змінних a[0], a[1], …, a[4]. Індекси можна задавати будь-якими цілочисельними виразами, крім типу *long*, наприклад, a[i+j], a[i%5], a[++i]. Виконуюча система Java слідкує за тим, щоб значення цих виразів не виходили за межі довжини масиву.

Третій етап – *ініціалізація* (initialization). На цьому етапі елементи масиву отримують початкові значення. Наприклад:

```
a[0] = 0.01; a[1] = -3.4; a[2] = 2.89; a[3] = 4.5; a[4] = -6.7;
for (int i = 0; i < 100; i++) b[i] = 1.0 /i;
for (int i = 0; i < 50; i++) ar[i] = 2 * i + 1;
```

Перші два етапи можна сумістити:

```
a = new double[5], b = new double[100];
```

```
int i = 0, ar[] = new int[50], k = -1;
```

Можна відразу задати і початкові значення, записавши їх в фігурних дужках через кому у вигляді констант або константних виразів. При цьому навіть необов'язково вказувати кількість елементів масиву, вона буде рівною кількості початкових значень:

```
double[] a = {0.01, -3.4, 2.89, 4.5, -6.7};
```

Можна поєднати другий і третій етап:

```
a = new double[] {0.1, 0.2, -0.3, 0.45, -0.02};
```

Можна навіть створити безіменний масив, відразу ж використовуючи результат операції *new*, наприклад, так:

```
System.out.println(new char[] {'H', 'e', 'l', 'l', 'o'});
```

II. Одновимірні масиви в мові програмування Java.

Масив вважається ***одновимірним***, якщо для визначення місцеположення елементу в масиві потрібно вказати значення одного індексу.

При виділенні пам'яті елементи масиву заповнюються:

- нульовими значеннями для числових типів;
- значеннями false для логічного типу boolean;
- пустими значеннями null для посилальних типів.

Посилання на масив не являється частиною описаного масиву, його можна перекинути на другий масив того ж типу операцією присвоювання. Наприклад, після присвоювання *a = b* обидва посилання *a* і *b* вказують на один і той же масив із 100 дійсних змінних типу *double* і містять одну і ту ж адресу.

Посиланню можна присвоїти «пусте» значення null, що не вказує на жодну адресу оперативної пам'яті:

```
ar = null;
```

Крім простої операції присвоювання, з посиланнями можна виконувати ще також перевірку на рівність, наприклад, a = b, і нерівність, a != b. При цьому співставляються адреси, що містяться в посиланнях.

Масиви в Java завжди визначаються динамічно, хоча посилання на них задаються статично. Крім посилання на масив, для кожного масиву автоматично визначається ціла константа з одним і тим же іменем *length*. Вона рівна довжині масиву. Для кожного масиву ім'я цієї константи уточнюється іменем масиву через крапку. Так константа *a.length* рівна 5, константа *b.length* рівна 100, а *ar.length* рівна 50. Останній елемент масиву *a* можна записати так: *a [a.length - 1]*, передостанній – *a [a.length - 2]* і т.д.

ІІІ. Багатовимірні масиви.

Багатовимірні масиви будуються за принципом «масив масивів». Масив, як відомо, є об'єктом. Двовимірний масив – це масив посилань на об'єкти-масиви. Тривимірний масив – це масив посилань на масиви, які у свою чергу, є масивами посилань на масиви.

Існують скорочені варіанти створення масиву, які дозволяють відразу розмістити всі необхідні масиви посилань. Можна зробити наступне оголошення масиву:

`char[] [] c;` що еквівалентно **`chart[] c[];`** або **`char c[][];`**

Потім визначаємо зовнішній масив:

```
c = new char[3][];
```

Стає зрозумілим, що *c* — масив, який складається з трьох елементів-масивів. Визначимо його елементи-масиви:

```
c[0] = new char[2];
c[1] = new char[4];
c[2] = new char[3];
```

Після цих визначень змінна *c.length* рівна 3, *c[0].length* рівна 2, *c[l].length* рівна 4 і *c[2].length* рівна 3.

Насамкінець, задаються початкові значення с [0] [0] = 'a', с[0][1] = 'r', с[1][0] = 'r',с[1][1] = 'a',с[1][2] = 'y' і т.д.

Двомірний масив у Java не зобов'язаний бути прямокутним. Опис, наведений вище, можна скоротити: **`int[] [] d = new int[3] [4].`**

А початкові значення можна задати таким чином: **`int[][]d ={{I,2, 3},{4, 5, 6}}.`**

У мові Java для ефективної роботи з масивами передбачено спеціальні засоби стандартної бібліотеки, основним із яких є клас Arrays, розміщений у пакеті java.util. Цей клас містить набір статичних методів, що дозволяють виконувати сортування, пошук, копіювання, порівняння та перетворення масивів без написання додаткових алгоритмів вручну.

Для використання цих методів необхідно підключити бібліотеку:

```
import java.util.Arrays;
```

Метод *sort()* використовується для впорядкування елементів масиву у зростаючому порядку.

```
int[] a = {5, 2, 8, 1, 3};
Arrays.sort(a);
System.out.println(Arrays.toString(a));
```

```
Результат:
[1, 2, 3, 5, 8]
```

Для числових типів сортування виконується автоматично за зростанням.

Метод *binarySearch()* виконує пошук елемента у відсортованому масиві.

```
int[] a = {1, 3, 5, 7, 9};
int index = Arrays.binarySearch(a, 5);
System.out.println(index);

Результат:
2
```

Метод повертає індекс знайденого елемента.

Метод *equals()* перевіряє, чи однакові два масиви за довжиною та значеннями.

```
int[] a = {1, 2, 3};
int[] b = {1, 2, 3};

System.out.println(Arrays.equals(a, b));

Результат:
True
```

Метод *fill()* заповнює всі елементи однаковим значенням.

```
int[] a = new int[5];
Arrays.fill(a, 7);
System.out.println(Arrays.toString(a));

Результат:
[7, 7, 7, 7, 7]
```

Часто цей метод використовується для початкової ініціалізації.

Метод *copyOf()* створює копію масиву заданої довжини.

```
int[] a = {1, 2, 3};
int[] b = Arrays.copyOf(a, 5);
System.out.println(Arrays.toString(b));

Результат:
[1, 2, 3, 0, 0]
```

Метод *copyOfRange()* – забезпечує копіювання частини масиву:

```
int[] a = {10, 20, 30, 40, 50};
int[] b = Arrays.copyOfRange(a, 1, 4);
System.out.println(Arrays.toString(b));

Результат:
[20, 30, 40]
```

Метод *toString()* – перетворення масиву в рядок. Для зручного виведення масиву використовується:

```
int[] a = {1, 2, 3};
System.out.println(Arrays.toString(a));

Результат:
[1, 2, 3]
```

Без цього виведення масиву відображає лише адресу об’єкта.

Метод *deepToString()* – виконує подібну дію для багатовимірних масивів:

```
int[][] a = {{1, 2}, {3, 4}};
System.out.println(Arrays.deepToString(a));

Результат:
[[1, 2], [3, 4]]
```

Методи класу Arrays значно спрощують роботу з масивами, оскільки дозволяють виконувати типові операції швидко, надійно й без розроблення додаткових алгоритмів. Їх використання є необхідною складовою сучасного програмування мовою Java.

ЛЕКЦІЯ 6

Тема: *Бібліотеки класів Java.*

План:

1. Вбудовані класи.
2. Особливості використання заміщуючих класів в Java.
3. Бібліотеки класів, що підключаються.

І. Вбудовані класи.

Клас – це шаблон для створення об’єкту.

Клас визначає структуру об’єкту і його *методи,* створюючи функціональний інтерфейс. В процесі виконання Java-програми система використовує визначення класів для створення представників класів. Представники класів є реальними *об’єктами.*

Методи – це підпрограми, приєднані до конкретних визначень класів.

Вони описуються усередині визначення класу на тому ж рівні, що і змінні об’єктів. При оголошенні методу задаються тип поверненого ним результату і список параметрів.

Конструктор – це метод класу, який ініціалізував новий об’єкт після його створення.

Ім’я конструктора завжди *співпадає* з ім’ям класу, в якому він розташований. У конструкторів немає типу поверненого результату, навіть ***void***. Наприклад:

```
class Point { int x, y;
Point(int x, int y){
this.x = x;
this.y = y;
} }
```

У Java відсутнє множинне спадкоємство. Кожен клас може мати тільки один батьківський клас. Таким чином, ми можемо прослідкувати ланцюжок спадкоємства від будь-якого класу піднімаючись все вище.

Існує клас, на якому такий ланцюжок завжди закінчується – це клас Object. Саме від нього успадковуються всі класи, в оголошенні яких явно не вказаний інший батьківський клас. А значить, будь-який клас безпосередньо або через свій батьківський клас є спадкоємцем Object. Звідси витікає, що методи цього класу є у будь-якого об’єкту (поля в Object відсутні).

Однією з важливих переваг платформи Java є наявність великої кількості ***вбудованих класів***, які входять до стандартної бібліотеки мови і забезпечують готові засоби для виконання типових програмних операцій. Такі класи реалізують роботу з числами, текстом, масивами, файлами, датами, колекціями, потоками введення-виведення та іншими об’єктами, що значно прискорює розроблення програмного забезпечення.

Вбудовані класи розміщуються у стандартних пакетах Java, основними з яких є: java.lang; java.util; java.io; java.time.

Особливістю пакета java.lang є те, що його класи підключаються автоматично, без використання оператора import.

Клас *String* використовується для роботи з текстовими даними. Він є одним із найуживаніших у Java.

```
String text = "Java";
System.out.println(text.length());
```

Основні методи:

- length() – довжина рядка;
- charAt() – символ за індексом;
- substring() – підрядок;
- toUpperCase() – перетворення у верхній регістр:

```
System.out.println(text.toUpperCase())
```

Клас *Math* містить математичні функції.

```
System.out.println(Math.sqrt(25));
```

Основні методи:

- sqrt() – квадратний корінь;
- pow() – піднесення до степеня;
- abs() – модуль числа;
- random() – випадкове число.

```
System.out.println(Math.pow(2, 3));
```

ІІ. Особливості використання заміщуючих класів в Java.

У Java під ***заміщуючими класами*** зазвичай розуміють класи-обгортки (wrapper classes), тобто спеціальні вбудовані класи, призначені для представлення примітивних типів даних у вигляді об’єктів. Їх поява зумовлена тим, що примітивні типи (int, double, char, boolean тощо) не є об’єктами, а тому не можуть безпосередньо використовуватися в багатьох механізмах об’єктно-орієнтованого програмування, зокрема у колекціях, узагальненнях та стандартних бібліотеках Java .

Приведемо назви цих класів і назви базових типів даних, які вони заміщають:

Базовий тип даних	Заміщуючий клас
boolean	Boolean
char	Character
int	Integer
long	Long
float	Float
double	Double

Для перетворення базових типів даних в об’єкти заміщуючого класу і назад не можна застосовувати оператор привласнення. Замість цього необхідно використовувати відповідні конструктори і методи заміщуючих класів.

Головна функція заміщуючих класів полягає у перетворенні примітивного значення в об’єкт, що дозволяє:

- використовувати примітивні значення в колекціях;
- викликати методи для обробки чисел;
- виконувати перетворення типів;
- працювати з узагальненими структурами даних.

Приклад створення об’єктів заміщуючих класів:

```
Integer a = Integer.valueOf(10);
```

```
Double b = Double.valueOf(5.5);
```

У сучасних версіях Java можливий скорочений запис:

```
Integer a = 10;
Double b = 5.5;
```

Це називається автоматичне пакування (autoboxing). При цьому Java автоматично перетворює примітивний тип у відповідний клас:

```
Integer number = 25;
```

Тут int автоматично перетворюється в Integer.

Зворотне перетворення також виконується автоматично:

```
Integer number = 30;
int x = number;
```

Об’єкт Integer автоматично перетворюється в int.

Передбачені також класи для виконання запуску процесів і потоків, управління системою безпеки, а також для вирішення інших системних завдань. Бібліотека вбудованих класів містить дуже важливі класи для роботи з виключеннями. Ці класи потрібні для обробки помилкових ситуацій, які можуть виникнути при роботі застосувань або аплетів Java.

III. Бібліотеки класів, що підключаються.

Однією з важливих особливостей Java є використання бібліотек класів, які містять готові програмні компоненти для виконання типових операцій. Такі бібліотеки входять до складу стандартної платформи Java і дозволяють програмісту не створювати багато функцій самостійно, а використовувати вже реалізовані класи, методи й інтерфейси.

У Java класи об’єднуються в пакети (packages) – логічно організовані групи класів за функціональним призначенням. Для використання класів із певного пакета їх необхідно підключити за допомогою оператора import.

Загальний синтаксис:

```
import ім'я_пакета.ім'я_класу;
```

або для підключення всіх класів пакета:

```
import ім'я_пакета.*;
```

Символ * означає імпорт усіх класів пакета.

Бібліотека класів *java.util* корисна при складанні застосувань, тому що в ній є класи для створення таких структур, як динамічні масиви, стеки і словники.

Є класи для роботи з генератором псевдовипадкових чисел, для розбору рядків на елементи, що становлять, для роботи з календарною датою і часом.

Клас *Scanner* використовується для введення даних із клавіатури.

```
import java.util.Scanner;

Scanner input = new Scanner(System.in);
int x = input.nextInt();
```

Основні методи: nextInt(); nextDouble(); nextLine().

Клас *Arrays* забезпечує роботу з масивами.

```
import java.util.Arrays;
int[] a = {3, 1, 2};
Arrays.sort(a);
```

Клас *System* забезпечує взаємодію із системним середовищем.

```
System.out.println("Hello");
```

Найчастіше використовується:

- System.out.println() – виведення;
- System.currentTimeMillis() – системний час.

Клас *Random* генерує випадкові числа.

```
import java.util.Random;

Random r = new Random();
System.out.println(r.nextInt(10));
Клас Date / сучасний java.time
```

У сучасних версіях Java використовується пакет *java.time*.

```
import java.time.LocalDate;

LocalDate today = LocalDate.now();
System.out.println(today);
```

Цей пакет дає змогу працювати з датою та часом.

У бібліотеці класів *java.io* зібрані класи, що мають відношення до введення і виведення даних через потоки. З використанням цих класів можна працювати не тільки з потоками байт, але також і з потоками даних інших типів, наприклад числами *int* або текстовими рядками.

Бібліотека класів java.net призначена для роботи в мережі. Вона містить класи, за допомогою яких можна працювати з універсальними мережевими адресами URL, передавати дані з використанням сокетів TCP і UDP, виконувати різні операції з адресами IP. Ця бібліотека містить також класи для виконання перетворень двійкових даних в текстовий формат.

Як приклад застосування, можна привести гру «Color Lines» (рис. 6.1).

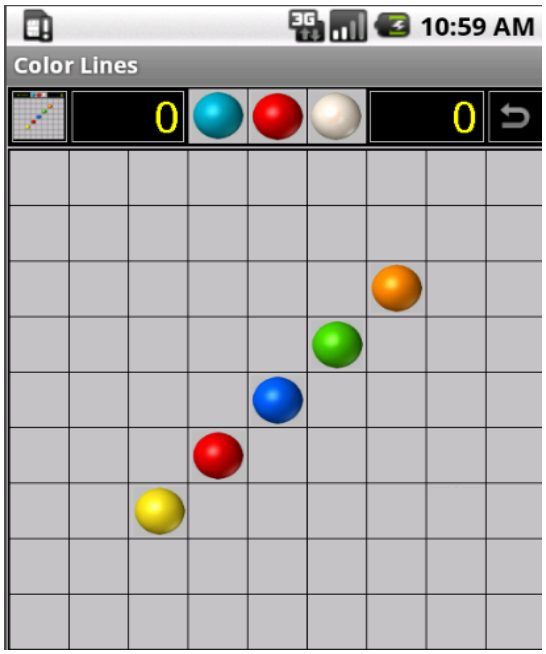

Рисунок 6.1 – Гра «Color Lines»

Для створення призначеного користувачам інтерфейсу аплети Java можуть і повинні використовувати бібліотеку класів *java.awt*. AWT – це скорочення від Abstract Window Toolkit – інструментарій для роботи з абстрактними вікнами.

Класи, що входять до складу бібліотеки *java.awt*, надають можливість створення призначеного для користувача інтерфейсу способом, не залежним від платформи, на якій виконується аплет Java. Користувач може створювати звичайні вікна і діалогові панелі, кнопки, перемикачі, списки, меню, смуги перегляду, однорядкові і багаторядкові поля для введення текстової інформації.

В Java, малювання і обробка графічних зображень виконується набагато простіше, оскільки нам доступна спеціально призначена для цього бібліотека класів *java.awt.image*. Крім широкої різноманітності і зручності визначених в ній класів і методів, здатність цієї бібліотеки полягає в можливості працювати з графічними зображеннями у форматі GIF. Цей формат широко використовується в Internet, оскільки він дозволяє стискати файли графічних зображень без втрати якості.

Бібліотека класів *java.awt.peer* служить для підключення компонент AWT (наприклад, кнопок, списків, полів редагування текстової інформації тощо) до реалізацій, залежних від платформи, в процесі створення цих компонент.

Бібліотека класів *java.applet* інкапсулює поведінку аплетів Java. При створенні аплетів потрібний клас *Applet*, розташований в цій бібліотеці класів.

ЛЕКЦІЯ 7

Тема: *Графічні примітиви.*

План:

1. Методи класу Graphics.
2. Клас Polygon.
3. Клас FontMetrics.
4. Можливості Java 2D.
5. Перетворення координат.

I. Методи класу Graphics.

Клас Graphics є складовою пакета java.awt і призначений для виконання графічних операцій у Java-додатках. За його допомогою здійснюється побудова графічних примітивів, відображення тексту, малювання геометричних фігур, зображень та керування кольором у віконних програмах. Найчастіше об’єкт класу Graphics використовується в методі paint(), який викликається під час відображення графічного компонента.

При створенні компонента, тобто об’єкта класу *Component*, автоматично формується його *графічний контекст*. В контексті розміщується область малювання і виведення тексту та зображень. Контекст містить поточний і альтернативний колір малювання і колір фону – об’єкти класу *Color*, поточний шрифт для виведення тексту – об’єкт класу *Font*. В контексті визначена система координат, початок якої має координати (0, 0) розташований у верхньому лівому куті області малювання, вісь Ох направлена вправо, вісь Оу – вниз. Точки координат знаходяться між пікселями. Керує контекстом *клас Graphics* або новий клас *Graphics2D*, введений в *Java 2*. Оскільки графічний контекст сильно залежить від конкретної графічної платформи, ці класи зроблені абстрактними. Тому не можна безпосередньо створити екземпляри класу Graphics або Graphics2D. Однак кожна віртуальна машина Java реалізує методи цих класів, створює їх екземпляри для компонента і представляє об’єкт класу *Graphics* методом *getGraphics()* класу *Component* або як аргумент методів *paint()* і *update()*.

Розглянемо, які методи роботи з графікою і текстом представляє клас Graphics.

При створенні контексту в ньому задається поточний колір для малювання, зазвичай чорний, і колір фону області малювання — білий або сірий. Змінити поточний колір можна методом *setCoIor (Color newCoIor)*, аргумент *newCoIor* котрого – об’єкт класу *Color*. Дізнатися поточний колір можна методом *getColor()*, повертаючим об’єкт класу *Color*.

Колір, як і все в Java – об'єкт певного класу, а саме, класу *Color*. Основу класу складають сім конструкторів кольору. Найпростіший конструктор:

```
Color(int red, int green, int blue)
```

створює колір, одержаний змішуванням червоного *red*, зеленого *green* і синього *blue*. Ця кольорова модель називається *RGB*. Кожна складова змінюється від 0 (відсутність складової) до 255 (повна інтенсивність цієї складової). Наприклад:

```
Color pureRed = new Color(255, 0, 0);
Color pureGreen = new Color(0, 255, 0);
```

визначають чистий яскраво-червоний *pureRed* і чистий яскраво-зелений *pureGreen* кольори.

В наступному конструкторі інтенсивність складової можна змінювати більш гладко дійсними числами від 0.0 (відсутність складової) до 1.0 (повна інтенсивність складової):

```
Color(float red, float green, float blue)
```

Наприклад:

```
Color someColor = new Color(0.05f, 0.4f, 0.95f);
```

Третій конструктор:

```
Color(int rgb)
```

задає всі три складові в одному цілому числі. В бітах 16-23 записується червона складова, в бітах 8-15 – зелена, а в бітах 0-7 – синя складова кольору. Наприклад:

```
Color c = new Color(0xFF8F48FF);
```

Червона складова задана з інтенсивністю 0x8F, зелена – 0x48, синя – 0xFF.

Наступні три конструктори:

```
Color(int red, int green, int blue, int alpha)
Color(float red, float green, float blue, float alpha)
Color(int rgb, boolean hasAlpha)
```

вводять четверту складову кольору, так звану «альфу», що визначає прозорість кольору. Ця складова проявляє себе при накладанні одного кольору на другий. Якщо альфа рівна 255 або 1,0, то колір повністю непрозорий, попередній колір не просвічується крізь нього. Якщо альфа рівна 0 або 0,0, то колір абсолютно прозорий, для кожного пікселя видно тільки попередній колір. Останній із цих конструкторів враховує складову альфа, що знаходиться в бітах 24-31, якщо параметр *hasAIpha* рівний *true*. Якщо ж *hasAIpha* рівне *false*, то складова альфа

вважається рівною 255, незалежно від того, що записано в старших бітах параметра RGB. Перші три конструктори створюють непрозорий колір з альфою, рівною 255 або 1,0.

Сьомий конструктор:

```
Color(ColorSpace cspace, float[] components, float alpha)
```

дозволяє створювати колір не тільки в кольоровій моделі RGB, але і в інших моделях (CMYK, HSB, CIEXYZ), визначених об'єктом класу *ColorSpace*. Для створення кольору в моделі HSB можна скористатися статичним методом:

```
getHSBColor(float hue, float saturation, float brightness).
```

Якщо немає необхідності ретельно підбирати кольори, то можна просто скористатися однією із тринадцати статичних констант типу *color* класу *Color*. Вони записуються рядковими літерами: black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow.

Методи класу *Color* дозволяють одержати складові поточного кольору: *getRed(), getGreen(), getBlue(), getAlpha(), getRGB(), getColorSpace(), getComponents()*.

Два методи створюють більш яскравий *brighter()* і більш темний *darker()* кольори в порівнянні з поточним кольором. Вони корисні, якщо потрібно виділити активний компонент або, навпаки, показати неактивний компонент блідніше решти компонентів.

Два статичних методи повертають колір, перетворений із кольорової моделі *RGB* в *HSB* і навпаки:

```
float[] RGBtoHSB(int red, int green, int blue, float[]
hsb)
int HSBtoRGB(int hue, int saturation, int brightness)
```

Створивши колір, можна малювати ним в графічному контексті. Основний метод малювання:

```
drawLine(int x1, int y1, int x2, int y2)
```

малює поточним кольором відрізок прямої між точками з координатами (x1, y1) і (x2, y2). Одного цього методу достатньо, щоб намалювати будь-яку картину за точками, малюючи кожну точку з координатами (x, y) методом *drawLine (x, y, x, y)* і змінюючи кольори від точки до точки.

Інші графічні примітиви:

– ***drawRect(int x, int y, int width, int height)*** – малює прямокутник зі сторонами, паралельними краям екрану, і задається координатами верхнього лівого кута (x, y), шириною *width* пікселів і висотою *height*

пікселів;

- ***draw3DRect(int x, int y, int width, int height, boolean raised)*** – малює прямокутник, що ніби-то виділяється із площини малювання, якщо аргумент *raised* рівний *true*, або ніби-то вдавлений в площину, якщо аргумент *raised* рівний *false*;
- ***drawOval(int x, int y, int width, int height)*** – малює овал, вписаний в прямокутник, заданий аргументами методу. Якщо *width==height*, то одержимо коло;
- ***drawArc(int x, int y, int width, int height, int startAngle, int arc)*** – малює дугу овала, вписаного в прямокутник, заданий першими чотирма аргументами. Дуга має величину *arc* градусів і відраховується від кута *startAngle*. Кут відраховується в градусах від осі Ох. Додатній кут відраховується проти годиникової стрілки, від'ємний – по годинниковій стрілці;
- ***drawRoundRect (int x, int y, int width, int height, int arcWidth, int arcHeight)*** – малює прямокутник із закругленими краями. Закруглення викреслюються четвертинками овалів, вписаних в прямокутники шириною *arcWidth* і висотою *arcHeight*, побудовані в кутах основного прямокутника;
- ***drawPolyline(int[] xPoints, int[] yPoints, int nPoints)*** – малює ламану з вершинами в точках *xPoints[i], yPoints[i])* і числом вершин *nPoints*;
- ***drawPolygon(int[] xPoints, int[] yPoints, int nPoints)*** – малює замкнену ламану, проводячи замикаючий відрізок прямої між першою і останньої точками;
- ***drawPoIygon(Polygon p)*** – малює замкнену ламану, вершини якої задані об'єктом р класу *Polygon*.

Декілька методів малюють фігури, залиті поточним кольором: *fillRect(), fill3DRect(), fillArco, fiIIOval(), fillPoIygon(), fillRoundRect()*. У них такі ж аргументи, як і у відповідних методів, малюючих незаповнені фігури. Наприклад, якщо потрібно змінити колір фону області малювання, встановлюють новий поточний колір і малюють ним заповнений прямокутник величиною у всю область малювання:

```
public void paint(Graphics g){
 Color initColor = g.getColor(); //Зберігаємо початковий
колір
 g.setColor(new Color(0, 0, 255)); //Встанавлюємо колір
фону
 //Заливаємо область малювання
```

```
    g.fillRect(0, 0, getSize().width - 1, getSize().height -
   1);
    g.setColor(initColor); // Відновлюємо початковий колір
    // Подальші дії}
```

Для виведення тексту в область малювання поточним кольором і шрифтом, починаючи з точки (x, y), в класі *Graphics* є декілька методів:

- ***drawString (String s, int x, int y)*** – виводить рядок *s*;
- ***drawBytes(byte[] b, int offset, int length, int x, int y)*** – виводить *length* елементів масиву байтів *b*, починаючи з індексу *offset*;
- ***drawChars(char[] ch, int offset, int length, int x, int y)*** – виводить *length* елементів масиву символів *ch*, починаючи з індексу *offset*.

Наступний метод виводить текст, занесений в об'єкт класу, реалізуючого інтерфейс *AttributedCharacterIterator*. Це дозволяє задавати свій шрифт для кожного символу:

```
drawString(AttributedCharacterIterator iter, int x, int y)
```

Точка (x, y) – це ліва нижня точка першої літери тексту на базовій лінії виведення шрифту.

Метод ***setFont(Font newFont)*** класу *Graphics* встановлює поточний шрифт для виведення тексту. Метод ***getFont()*** повертає поточний шрифт.

Об'єкти класу *Font* зберігають креслення символів, що утворюють шрифт. Їх можна створити двома конструкторами:

- ***Font (Map attributes)*** – задає шрифт із заданим аргументом *attributes*. Ключі атрибутів і деякі їх значенния задаються константами класу *TextAttribute* із пакету *java.awt.font*. Цей конструктор характерний для *Java 2D*.
- ***Font (String name, int style, int size)*** – задає шрифт по імені *name*, із стилем *style* і розміром *size* типографічних пунктів. Цей конструктор характерний для *JDK 1.1*, але широко використовується і в *Java 2D*.

Типографічний пункт в Україні рівний 0,376 мм, точніше, 1/72 частина французького дюйма. В англо-американській системі мір пункт рівний 1/72 частина англійського дюйма, 0,351 мм. Останній пункт і використовується в комп'ютерній графіці.

Ім'ям шрифту *name* може бути рядок із фізичним іменем шрифту, наприклад, «Courier New», або один із рядків «Dialog», «Dialoglnput», «Monospaced», «Serif», «SansSerif», «Symbol» – так звані *логічні імена шрифтів*.

Стиль шрифту *style* – це одна із констант класу *Font*: *BOLD* – напівжирний; *ITALIC* – курсив; *PLAIN* – звичайний.

Шрифти повинні знаходитися в складі графічної системи тієї машини, на якій виконується додаток. Список імен доступних з них можна продивитися наступними операторами:

```
Font[] fnt = Toolkit.getGraphicsEnvironment.getAHFonts();
for (int i = 0; i< fnt.length; i++)
System.out.println(fnt[i].getFontName());
```

При виведенні тексту логічним іменам шрифтів і стилям співставляються *фізичні імена шрифтів* або і*мена сімейств шрифтів*. Ці імена реальних шрифтів, наявних у графічній підсистемі операційної системи. Наприклад, логічному імені «Serif» може бути співставлене ім'я сімейства шрифтів *Times New Roman*.

Отже, за допомогою класу Graphics можна:

- будувати геометричні фігури;
- створювати прості графічні інтерфейси;
- реалізовувати навчальні графічні моделі;
- створювати початкову комп'ютерну анімацію.

Клас Graphics є базовим інструментом графічного програмування в Java і широко використовується під час створення графічних інтерфейсів, навчальних програм і візуалізації даних.

II. Клас Polygon.

Клас ***Polygon*** входить до пакета java.awt і використовується для створення та обробки багатокутників у графічних програмах Java. Багатокутник у цьому випадку являє собою замкнену фігуру, утворену послідовністю відрізків, що з'єднують задані вершини. Клас Polygon широко застосовується під час побудови графічних об'єктів, схем, діаграм та елементів інтерфейсу.

Для використання класу необхідно підключити бібліотеку: import java.awt.*;

Об'єкт класу Polygon зберігає:

- координати вершин по осі X;
- координати вершин по осі Y;
- кількість вершин багатокутника.

Об'єкти класу ***Polygon*** можна створити двома конструкторами:

- *Polygon ()* – створює пустий об'єкт;
- *Polygon(int[] xPoints, int[] yPoints, int nPoints)* – задаються вершини багатокутника (xPoints[i], yPoints[i]) і їх число nPoints.

Після створення об'єкта в нього можна додавати вершини методом:

```
addPoint(int x, int y)
```

Логічні методи *contains()* дозволяють перевірити, чи не лежить в багатокутнику задана аргументами методу точка, відрізок прямої або цілий прямокутник зі сторонами, паралельними сторонам екрану.

Логічні методи *intersects()* дозволяють перевірити, чи не перетинаються з даним багатокутником відрізок прямої, заданий аргументами методу, або прямокутник зі сторонами, паралельними сторонам екрану. Методи *getBounds()* і *getBounds2D()* – повертають прямокутник, цілком вміщаючий в себе даний многокутник.

Узагальнивши сказане, модна відмітити, що клас Polygon використовується для: побудови складних геометричних фігур; створення графічних схем; реалізації навчальної графіки; моделювання елементів інтерфейсу. Також цей клас є зручним засобом побудови багатокутників у Java-графіці, оскільки дозволяє працювати з довільною кількістю вершин і легко інтегрується з методами класу Graphics.

III. Клас FontMetrics.

Клас *FontMetrics* є абстрактним, тому не можна скористатися його конструктором. Для одержання об'єкту класу *FontMetrics*, що містить набір метричних характеристик шрифту f, треба звернутися до методу *getFontMetrics(f)* класу *Graphics* або класу *Component*. *FontMetrics* дозволяє взнати ширину окремого символу *ch* в пікселях методом *charwidth(ch),* загальну ширину всіх символів масиву або підмасиву символів або байтів методами *getChars()* і *getBytes()*, ширину цілого рядка *str* в пікселях методом *stringwidth(str)*.

Декілька методів повертають в пікселях вертикальні розміри шрифту.

Інтерліньяж (leading) – відстань між нижньою точкою звисаючих елементів таких літер, как *р, у* і верхньою точкою виступаючих елементів таких літер, як *б* і *в,* повернення якої здійснює метод *getLeading()*.

Середня відстань від базової лінії шрифту до верхньої точки прописних літер і виступаючих елементів того ж рядка (ascent) повертає метод *getAscent()*, а максимальне – метод *getMaxAscent()*.

Рисунок 7.1 – Характеристики FontMetrics

Середню відстань звисаючих елементів від базової лінії того ж рядка (descent) повертає метод *getDescent()*, а максимальну – метод *getMaxDescent()*.

Висоту шрифту *(height)* – суму *ascent + descent + leading* – повертає метод *getHeight()*. Висота шрифту рівна відстані між базовими лініями сусідніх рядків. Додаткові характеристики шрифту можно визначити методами класу *LineMetrics* пакету *java.awt.font*. Об'єкт цього класу можна одержати кількома методами *getLineMetrics()* класу *FontMetrics*.

IV. Можливості Java 2D.

Засоби малювання і виведення тексту в класі *Graphics* досить обмежені. Лінії можна проводити тільки суцільні і тільки товщиною в один піксель, текст виводиться тільки горизонтально і зліва направо, не враховуються особливості пристроїв виведення, наприклад, розширення екрану. Ці обмеження можна обійти різними хитрощами: креслити декілька паралельних ліній, притиснутих одна до одної, або вузький заповнений прямокутник, виводити текст по одній літері, одержати розширення екрану методом *getScreenSize()* класу *Java.awt.Toolkit* і використовувати його надалі. Але все це утруднює програмування, лишає його стрункості і природності. *Java 2* в рамках системи *Java 2D*, значно розширений класом Graphics2D. В систему пакетів і класів *Java 2D*, основа якої – клас *Graphics2D* пакету *java.awt*, внесено наступні нові положення:

- крім координатної системи, прийнятої в класі *Graphics* і названої координатним простором користувача (User Space), введена ще система координат пристрою виведення (Device Space): екрану монітора, принтера. Методи класу *Graphics2D* автоматично переводять систему координат користувача в систему координат пристрою;
- перетворення координат користувача в координати пристрою можна задати «вручну», причому перетворенням може бути будь-яке перетворення площини. Воно визначається як об'єкт класу *AffineTransform*. Його можна установити як перетворення по замовчуванню методом *setTransform()*. Можна виконати перетворення «на льоту» методами *transform()* і *translate()* і робити композицію перетворень методом *concatenate()*;
- графічні примітиви: прямокутник, овал, дуга і ін., реалізують новий інтерфейс *shape* пакету *java.awt*. Для їх викреслювання використовується метод *draw()*, аргументом якого може бути будь-який об'єкт, що реалізує інтерфейс *shape*. Введено також метод *fill()*, заповнюючий фігури;

- для креслення ліній введено поняття пера (*pen*). Властивості пера описує інтерфейс *stroke*. Клас *Basicstroke* реалізує цей інтерфейс;
- методи заповнення фігур описані в інтерфейсі *Paint*. Три класи реалізують цей інтерфейс. Клас *Color* реалізує його суцільною (solid) заливкою, клас *GradientPaint* – градієнтним заповненням, при якому колір плавно змінюєтья від однієї заданої точки до другої заданої точки, клас *Texturepaint* – заповненням по зразку (pattern fill);
- літери тексту розуміються як фігури, тобто об’єкти, реалізуючі інтерфейс *shape*, і можуть викреслюватися методом *draw()* з використанням всіх можливостей цього методу. При їх викреслюванні використовується перо та всі методи заповнення і перетворення;
- крім імені, стилю і розміру, шрифт отримав багато додаткових атрибутів, наприклад, перетворення координат, підкреслювання або закреслювання тексту, виведення тексту справа наліво. Колір тексту і його фону являються тепер трибутами самого тексту, а не графічного контексту. Можна задати різну ширину символів шрифту, надрядкові і підрядкові індекси. Атрибути встановлюються константами класу *TextAttribute*;
- процес візуалізації (rendering) регулюється правилами, визначеними константами класу *RenderingHints*.

V. Перетворення координат.

У графічному програмуванні Java важливим є правильне розуміння координатної системи, оскільки всі графічні об’єкти розміщуються на площині відповідно до певних координат. Будь-яка фігура, текст або зображення виводяться через координати, які визначають їх положення у графічному вікні.

У Java графічне середовище використовує екранну координатну систему, у якій початок координат знаходиться у верхньому лівому куті компонента точка (0,0) – верхній лівий кут; вісь X спрямована праворуч; вісь Y спрямована вниз. Це відрізняється від математичної системи координат, де вісь Y спрямована вгору (рис. 7.2).

У Java найчастіше використовують:

- зміщення координат;
- масштабування;
- обертання;
- перехід від математичних координат до екранних.

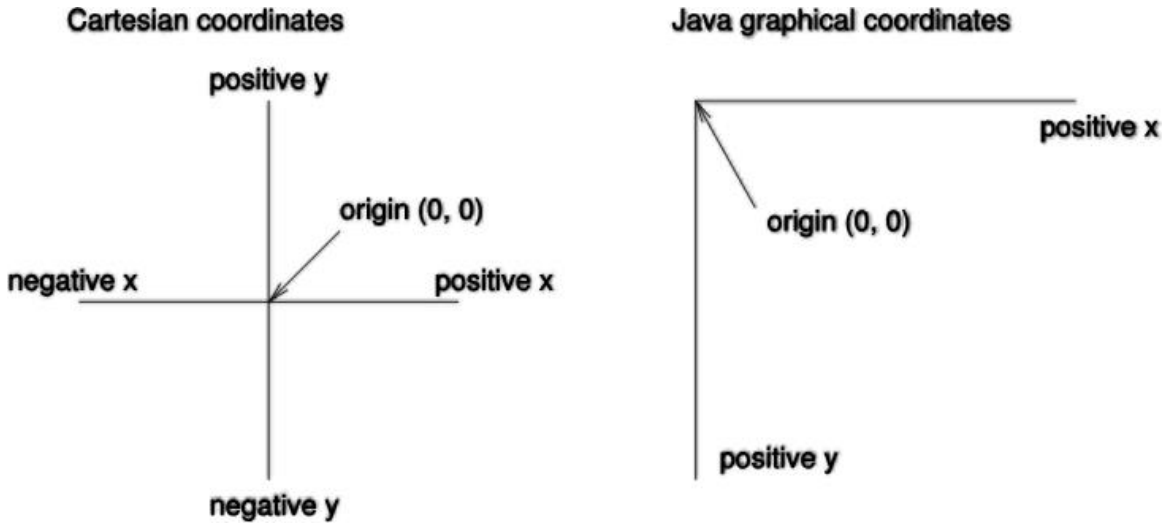

Рисунок 7.2 – Графічне подання координат в Java

Правило перетворення координат користувача в координати графічного пристрою задається автоматично при створенні графічного контексту так же, як колір і шрифт. Надалі його можна змінити методом *setTransform()* так же, як змінюються колір або шрифт. Аргументом цього методу служить об'єкт класу *AffineTransform* із пакету *java.awt.geom.*

Афінне перетворення координат задається двома основними конструкторами класу *AffineTransform*:

```
AffineTransform(double a,double b,double c,double d,double
                                e,double f)
AffineTransform (float a, float b, float c, float d, float e,
                                float f)
```

При цьому точка з координатами (x, y) в просторі користувача перейде в точку з координатами (a*x+c*y+e, b*x+d*y+f) в просторі графічного пристрою. Таке перетворення не викривляє площину – прямі лінії переходять в прямі, кути між лініями зберігаються. Прикладами афінних перетворень служать повороти навколо будь-якої точки на будь-який кут, паралельні зсуви, відображення від осей, стиснення і розтягнення по осях.

Наступні два конструктори використовують в якості аргумента масив {*a, b, c, d, e, f*} або {*a, b, c, d*}, якщо e = f = 0, складений із таких же коефіцієнтів в тому ж порядку:

```
AffineTransform(double[] arr)
AffineTransform(float[] arr)
```

П'ятий конструктор створює новий об'єкт по іншому, уже наявному, об'єкту:

```
AffineTransform(AffineTransform at)
```

Шостий конструктор – конструктор по замовчуванню – створює тотожне перетворення:

```
AffineTransform()
```

В багатьох випадках зручніше створювати перетворення статичними методами, повертаючими об’єкт класу *AffineTransform*:

- ***getRotateInstance (double angle)*** – забезпечує поворот на кут angle, заданий в радіанах, навколо початку координат. Додатній напрям повороту такий, що точки осі Ох повертаються в напряму до осі Оу. Якщо осі координат користувача не змінювалися, то додатне значення *angle* задає поворот по годинниковій стрілці;
- ***getRotateInstance(double angle, double x, double y)*** – такий же поворот навколо точки з координатами (x, y);
- ***getScaleInstance (double sx, double sy)*** – змінює маштаб по осі *Ox* в *sx* разів, по осі *Oy* – в *sy* разів;
- ***getShareInstance (double shx, double shy)*** – перетворює кожну точку *(x, y)* в точку *(x + shx*y, y + shy*x)*;
- ***getTranslateInstance (double tx, double ty)*** – зміщує кожну точку *(x, y)* в точку *(x + tx, y + ty)*.

ЛЕКЦІЯ 8

Тема: *Основні компоненти мови Java*.

План:

1. Ієрархія класів AWT.
2. Клас Component. Клас Container.
3. Компонент Label.
4. Компонент Checkbox та клас CheckboxGroup.
5. Компонент Choice. Компонент List.

І. Ієрархія класів AWT.

Додатки Java повинні працювати в будь-якому або хоча б в багатьох графічних середовищах. Потрібна бібліотека класів, незалежна від конкретної графічної системи. Класи бібліотеки AWT реалізують інтерфейси для створення додатків. Додатки Java використовують дані методи для розміщення і переміщення графічних об’єктів, зміни їх розмірів, взаємодії об’єктів. З іншого боку, для роботи з екраном в конкретному графічному середовищі ці інтерфейси реалізуються в кожному такому середовищі окремо. В графічній оболонці це робиться по-своєму, засобами цієї оболонки за допомогою

графічних бібліотек операційної системи. Такі інтерфейси були названі ***peer-інтерфейсами***.

AWT (Abstract Window Toolkit) – це базова бібліотека Java для створення графічного інтерфейсу користувача. Вона містить систему взаємопов'язаних класів, призначених для побудови вікон, елементів керування, графічних компонентів та обробки подій. Ієрархія класів AWT побудована за принципами об'єктно-орієнтованого програмування: усі графічні елементи мають спільного базового предка й успадковують його властивості

Бібліотека класів Java, заснована на peer-інтерфейсах, отримала назву AWT (Abstract Window Toolkit). При виведенні об'єкта, створеного в додатку Java і заснованого на peer-інтерфейсі, на екран створюється парний йому (peer-to-peer) об'єкт графічної підсистеми операційної системи, котрий і відображається на екрані. Ці об'єкти тісно взаємодіють під час роботи додатку. Тому графічні об'єкти AWT в кожному графічному середовищі мають вигляд, характерний для цього середовища: в MS Windows, Motif, OpenLook, OpenWindows, скрізь вікна, створені в AWT, виглядають як «рідні» вікна. Із-за такої реалізації peer-інтерфейсів і інших «рідних» методів, написаних, головним чином, на мові C++, приходиться для кожної платформи випускати свій варіант JDK.

У версії JDK 1.1 бібліотека AWT була перероблена. В неї додана можливість створення компонентів, повністю написаних на Java і не залежних від peer-інтерфейсів. Такі компоненти стали називати *«легкими»* на відміну від компонентів, реалізованих через peer-інтерфейси, названих *«важкими»*. В Java 2 бібліотека AWT значно розширена доданням нових засобів малювання, виведення текстів і зображень, які одержали назву Java 2D, і засобів, що реалізують переміщення тексту методом DnD (Drag and Drop). Крім того, в Java 2 включені нові методи введення-виведення Input Method Framework і засоби зв'язку з додатковими пристроями введення-виведення, такими як світлове перо або клавіатура Брайля, названі *Accessibility*. Всі ці засоби Java 2 (AWT, Swing, Java 2D, DnD, Input Method Framework і Accessibility) склали бібліотеку графічних засобів Java, названу JFC (Java Foundation Classes).

В AWT компонентом вважається об'єкт класу *Component* або об'єкт будь-якого класу, розширюючий клас *Component*. В даному класі зібрані загальні методи роботи з будь-яким компонентом графічного інтерфейса користувача.

Основу бібліотеки AWT складають готові компоненти: *Button, Canvas, Checkbox, Choice, Container, Label, List, Scrollbar, TextArea, TextField, Menubar, Menu, PopupMenu, MenuItem, CheckboxMenuItem*. Якщо даного набору мало, то від класу *Canvas* можна створити власні «важкі» компоненти, а від класу Component – «легкі» компоненти.

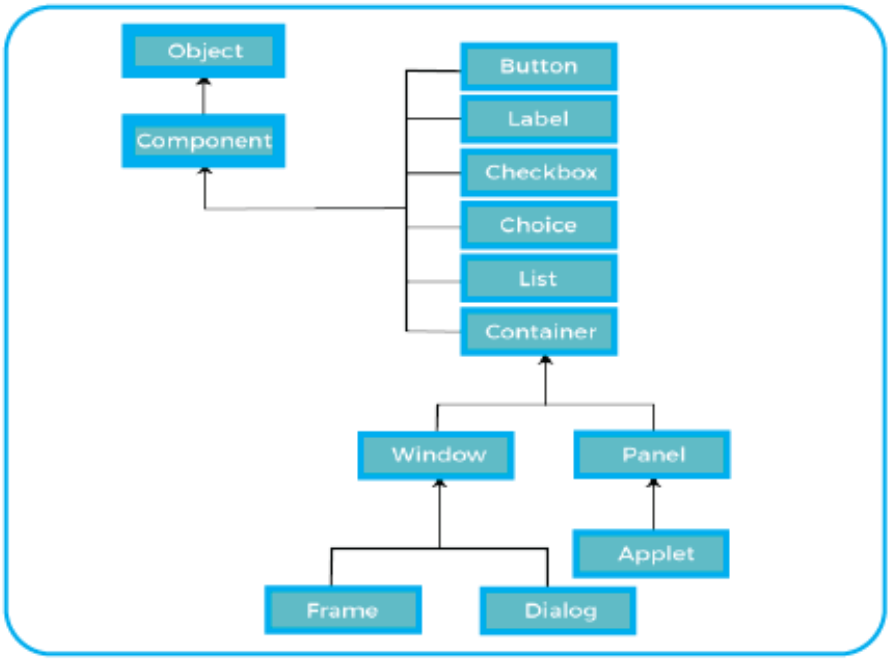

Рисунок 8.1 – Ієрархія класів AWT

Кожен компонент перед виведенням на екран поміщається в контейнер. В мові Java контейнер – це об’єкт класу *Container* або будь-якого його розширення.

Основні контейнери – це класи *Panel, ScrollPane, Window, Frame, Dialog, FileDialog*. Свої «важкі» контейнери можна створити від класу *Panel*, а «легкі» – від класу *Container*. Цілий набір класів допомагає розміщувати компоненти, задавати колір, шрифт і зображення, реагувати на сигнали від миші і клавіатури.

Графічна бібліотека AWT пропонує більше двадцяти готових компонентів. Найбільш часто використовуються підкласи класу Component: класи Button, Canvas, Checkbox, Choice, Container, Label, List, Scrollbar, TextArea, TextField, Panel, ScrollPane, Window, Dialog, FileDialog, Frame.

Ще одна група компонентів – це компоненти меню – класи Menuitem, MenuBar, Menu, PopupMenu, CheckboxMenuItem. Вивчати ці компоненти необхідно від простих компонентів до складних і від найбільш часто використовуваних до використовуваних рідше.

Ієрархія AWT забезпечує:

- повторне використання властивостей;
- уніфіковану систему побудови GUI;
- спрощення створення складних інтерфейсів.

Ієрархія класів AWT формує фундамент графічного програмування Java, на основі якого будуються вікна, кнопки, поля введення та графічні компоненти

ІІ. Клас Component. Клас Container.

Компонент – окремий, повністю визначений елемент, котрий можна використовувати в графічному інтерфейсі незалежно від інших елементів.

Клас Component – центр бібліотеки AWT. В ньому п'ять статичних констант, визначаючих розміщення компонента в середині простору, виділеного для компонента у вміщуючому його контейнері: BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT і близько сотні методів.

Більшість методів даного класу – це методи доступу *getxxx(), isxxx(), setxxx()*. Оскільки клас *Component* абстрактний, він не може використовуватися сам по собі, а використовуються лише його підкласи. Компонент завжди займає прямокутну область зі сторонами, паралельними сторонам екрану і в кожен момент часу має певні розміри, вимірювані в пікселях, які можна взнати методом *getSize()*, повертаючим об'єкт класу *Dimension*, або цілочисельними методами *getHeight()* і *getWidth()*, повертаючими висоту і ширину прямокутника.

Новий розмір компонента можна встановити із програми методами *setSize(Dimension d)* або *setSize(int width, int height)*, якщо це дозволяє менеджер розміщення контейнера, що містить компонент. У компонента є оптимальний розмір, при якому він виглядає найбільш пропорційно. Його можна одержати методом *getPreferredSize()* у вигляді об'єкта *Dimension*.

Компонент має мінімальний і максимальний розміри. Їх повертають методи *getMinimumSize()* і *getMaximumSize()* у вигляді об'єкта *Dimension*. В компоненті є система координат. Її початок точка з координатами (0, 0) – знаходиться в лівому верхньому куті компонента, вісь Ох іде вправо, вісь Оу – вниз, координатні точки розташовані між пікселями. В компоненті зберігаються координати його лівого верхнього кута в системі координат вміщаючого його контейнера. Їх можна взнати методом *getLocation()*, а змінити – методом *setLocation()*.

Можна дізнатися відразу і положення, і розмір прямокутної області компонента методом *getBounds()*, повертаючим об'єкт класу *Rectangle*, і змінити разом і положення, і розмір компонента методами *setBounds()*.

Компонент може бути недоступним для дій користувача, тоді він виділяється на екрані світло-сірим кольором. Доступність компонента можна перевірити логічним методом *isEnabled()*, а змінити – методом *setEnabled(boolean enable)*. Для багатьох компонентів визначається графічний контекст – об'єкт класу *Graphics*, – який керується методом *paint()*, і який можна одержати методом getGraphics().

В контексті є поточний колір і колір фону – об'єкти класу *Color*. Колір фону можна одержати методом *getBackground()*, а змінити – методом *setBackground (Color color)*. Поточний колір можна одержати методом *getForeground()*, а змінити – методом *setForeground (Color color)*.

В контексті є шрифт – об'єкт класу *Font*, що повертається методом *getFont()* і змінюється методом *setFont(Font font)*.

Клас Container – прямий підклас класу *Component*, і наслідує всі його методи.

Крім них основу класу складають методи додавання компонентів у контейнер:

- ***add (Component comp)*** – компонент *comp* додається в кінець контейнера;
- ***add (Component comp, int index)*** – компонент *comp* додається в позицію *index* в контейнері, якщо *index == -1*, то компонент додається в кінець контейнера;
- ***add (Component comp, object constraints)*** – менеджеру розміщення контейнера даються вказівки об'єктом *Constraints*;
- ***add (String name. Component comp)*** – компонент отримує ім'я *name*.
- Два методи видаляють компоненти із контейнера:
- ***remove (Component comp)*** – видаляє компонент з іменем *comp*;
- ***remove (int index)*** – видаляє компонент з індексом *index* в контейнері.

Один із компонентів в контейнері отримує фокус вводу (input focus), на нього направляється введення з клавіатури. Фокус можна переносити з одного компонента на інший клавішами <*Tab*> і <*Shift*>+<*Tab*>. Компонент може запросити фокус методом *requestFocus()* і передати фокус наступному компоненту методом *transferFocus()*.

Компонент може перевірити, чи має він фокус, своїм логічним методом *hasFocus(f)*.

Для полегшення розміщення компонентів в контейнері визначається менеджер розміщення – об'єкт, реалізуючий інтерфейс *LayoutManager* або його підінтерфейс *LayoutManager2*. Кожний менеджер розміщує компоненти в певному своєму порядку. Менеджер визначає зміст слів «додати в кінець контейнера» і «додати в позицію index».

В контейнері в будь-який момент часу може бути встановлений тільки один менеджер розміщення. В кожному контейнері є свій менеджер по замовчуванню, встановлення іншого менеджера виконується методом *setLayout(LayoutManager manager)*. Менеджер по замовчуванню визначається методом *setLayout (null)*.

III. Компонент Label.

Компонент Label є одним із базових елементів графічного інтерфейсу в бібліотеці AWT і призначений для відображення короткого текстового повідомлення у вікні програми. Він використовується тоді, коли необхідно підписати інші елементи інтерфейсу, наприклад поля введення, кнопки або списки, а також для виведення статичної інформації, яка не змінюється користувачем.

Компонент Label відображає текст, який користувач не може редагувати. На відміну від текстових полів, Label виконує лише інформаційну функцію.

Компонент Label – це рядок тексту, оформлений як графічний компонент для розміщення в контейнері. Текст можна поміняти тільки методом доступу *setText(String text)*, а не введенням його користувачем з клавіатури або за допомогою миші. Створюється об'єкт цього класу одним із трьох конструкторів:

- ***Label ()*** – пустий об'єкт без тексту;
- ***Label (String text)*** – об'єкт з текстом *text*, який притискується до лівого краю компонента;
- ***Label (String text, int alignment)*** – об'єкт з текстом *text* і визначеним розміщенням в компоненті тексту, задаваного однією з трьох констант: CENTER, LEFT, RIGHT.

Розміщення можна змінити методом доступу *setAlignment(int alignment)*. Решта методів, крім методів, наслідуваних від класу *Component*, дозволяють одержати текст *getText()* і розміщення *getAlignment()*.

В класі *Label* відбуваються події класів Component: *ComponentEvent, FocusEvent, KeyEvent, MouseEvent*.

IV. Компонент Checkbox та клас CheckboxGroup.

Компонент ***Checkbox*** – це напис справа від невеликого квадратика, в якому в деяких графічних системах появляється галочка після кліку кнопкою миші – компонент переходить в стан *on*. Після наступного кліку галочка пропадає – це стан *off*. В інших графічних системах стан *on* відмічається «вдавлюванням» квадратика. В компоненті *Checkbox* стани *on/off* відмічаються логічними значеннями *true/false* відповідно.

Три конструктори *Checkbox(), Checkbox(String label), Checkbox(String label, boolean state)* створюють компонент без напису, з написом *label* в стані *off*, і в заданому стані. Методи доступу *getLabel(), setLabel (String label), getState(), setState (boolean state)* повертають і змінюють ці параметри

компонента. Компоненти *Checkbox* зручні для швидкого і наочного вибору із списку, цілком розташованого на екрані.

В бібліотеці AWT радіокнопки не створюють окремий компонент. Замість цього декілька компонентів Checkbox об'єднуються в группу за допомогою об'єкту класу *CheckboxGroup*.

Завдання ***CheckboxGroup*** – дати спільне ім'я всім об'єктам Checkbox, що утворюють одну групу. В нього входить один конструктор по замовчуванню CheckboxGroup() і два методи доступу:

- ***getSelectedCheckbox()*** – повертає вибраний об'єкт Checkbox;
- ***setSelectedCheckbox (Checkbox box)*** – здійснює задання вибору.

Щоб організувати групу радіокнопок, потрібно спочатку сформувати об'єкт класу CheckboxGroup, а потім створити кнопки конструкторами: *Checkbox(String label, CheckboxGroupgroup, boolean state), Checkbox(String label, boolean state, CheckboxGroup group).* Ці конструктори ідентичні. Тільки одна радіокнопка в групі може мати стан *state = true.*

V. Компонент Choice. Компонент List.

Компонент ***Choice*** – це список, що розкривається, один, вибраний пункт (item) якого видимий в полі, а інші з'являються при кліку кнопкою миші на невелику кнопку справа від поля компонента.

Спочатку конструктором *Choice()* створюється пустий список. Потім, методом *add (String text)*, в список додаються нові пункти з текстом *text.* Вони розміщуються в порядку написання методів *add()* і нумеруються від нуля. Вставити новий пункт в потрібне місце можна методом *insert (String text, int position).* Вибір пункту можна зробити із програми методом *select (String text)* або *select(int position).* Видалити один пункт із списку можна методом *remove(String text)* або *remove (int position),* а всі пункти — методом *remove().*

Число пунктів у списку можна взнати методом *getItemCount()*. Вияснити, який пункт знаходиться в позиції *pos* можна методом *getItem(int pos),* повертаючим рядок. Нарешті, визначення вибраного пункту виконується методом *getSelectedIndex(),* повертаючим позицію цього пункту, або методом *getSelectedItem()*, повертаючим виділений рядок.

Компонент ***List*** – це список із смугою прокрутки, в якому можна виділити один або декілька пунктів. Кількість видимих на екрані пунктів визначається конструктором списку і розміром компонента.

В класі ***List*** три конструктори:

- ***List()*** – створює пустий список з чотирма видимими пунктами;
- ***List (int rows)*** – створює пустий список з *rows* видимими пунктами;

– ***List (int rows, boolean multiple)*** – створює пустий список в якому можна відмітити декілька пунктів, якщо *multiple == true*.

Після створення об'єкта в список додаються пункти з текстом *item*:

– ***метод add (String item)*** – додає новий пункт в кінець списку;

– ***метод add (String item, int position)*** – додає новий пункт в позицію *position*.

Позиції нумеруються послідовно, починаючи з нуля. Видалити пункт можна методами: *remove (String item), remove (int position), removeAll ()*.

Метод *replaceItem(String newitem, int pos)* дозволяє замінити текст пункту в позиції *pos*. Кількість пунктів у списку повертає метод *getItemCount()*. Виділений пункт можна отримати методом *getSelectedItem()*, а його позицію – методом *getSelectedlndex()*.

ЛЕКЦІЯ 9

Тема: *Контейнери Java.*

План:

1. Контейнер Panel та ScrollPane.
2. Контейнер Window.
3. Контейнер Dialog та FileDialog.

І. Контейнер Panel та ScrollPane.

У бібліотеці AWT важливе місце займають контейнери, тобто компоненти, призначені для розміщення інших графічних елементів інтерфейсу. До найбільш уживаних контейнерів належать Panel і ScrollPane, які забезпечують організацію візуальної структури програми та зручне керування розташуванням компонентів.

Контейнер Panel – це невидимий компонент графічного інтерфейсу, призначений для об'єднання декількох компонентів в один об'єкт типу Panel.

В ньому лише два конструктори:

– ***Panel()*** – створює контейнер з менеджером розміщення по замовчуванню *FlowLayoutJ*.

– ***Panel(LayoutManager layout)*** – створює контейнер з вказаним менеджером розміщення компонентів *layout*.

Після створення контейнера в нього додаються компоненти, наслідувані методом *add()*.

Panel не створює окремого вікна, а розміщується всередині інших контейнерів: Frame, Dialog, Applet.

Створення Panel відображено в наступному коді:

```
Panel p = new Panel();
```

Додавання компонентів до Panel:

```
Panel p = new Panel();
p.add(new Button("OK"));
p.add(new Label("Java"));
```

Усі компоненти автоматично розташовуються відповідно до менеджера розміщення.

Додавання Panel у вікно:

```
Frame f = new Frame();
f.add(p);
```

За замовчуванням Panel використовує FlowLayout, тобто компоненти розташовуються послідовно зліва направо.

```
p.setLayout(new FlowLayout());
```

Контейнер ScrollPane може містить тільки один компонент. Контейнер забезпечує засоби прокрутки для перегляду великого компонента. В контейнері можна встановити смуги прокрутки або постійно, константою *SCROLLBARS_ ALWAYS*, або так, щоб вони появлялись тільки при необхідності (якщо компонент дійсно не поміщується у вікно) константою *SCROLLBARS_AS_NEEDED*. Якщо смуги прокрутки не встановлені (це задає константа *SCROLLBARS_NEVER*), то переміщення компонента для перегляду потрібно забезпечити із програми одним із методів *setScrollPosition()*. В класі два конструктори:

- ***ScrollPane()*** – створює контейнер, в якому смуги прокрутки з'являються по необхідності;
- ***ScrollPane(int scrollbars)*** – створює контейнер, в якому поява лінійок прокрутки задається однією із трьох вказаних вище констант.

Конструктори створюють контейнер розміром 100x100 пікселів, після чого можна змінити розмір *методом setSize(int width, int height)*. Обмеження, що ScrollPane може містити лише один компонент, легко обійти. Завжди можна зробити цим єдиним компонентом об'єкт класу *Panel*, розмістивши на панелі будь-яку кількість компонентів.

Panel використовується для групування елементів інтерфейсу, створення логічних блоків, побудови складних вікон. ScrollPane використовується для великих текстових областей, графічних полотен, великих таблиць.

ІІ. Контейнер Window.

Контейнер Window – це порожнє вікно, без внутрішніх елементів: рамки, рядка заголовку, рядка меню, смуг прокрутки. Це просто прямокутна область на екрані. Вікно типу *Window* самостійне, не міститься ні в якому контейнері, його не потрібно заносити в контейнер методом *add()*. Однак воно не зв'язане з віконним менеджером графічної системи. Тому не можна змінити його розміри та перемістити в інше місце екрану. Воно може бути створене тільки яким-небудь уже існуючим вікном, «власником» (owner) або «батьком» (parent) вікна *Window*. Коли вікно-власник прибирається з екрану, разом з ним прибирається і породжене вікно.

Власник вікна вказується в конструкторі:

- ***Window (Frame f) –*** створює вікно, власник якого – фрейм f;
- ***Window (Window owner)*** – створює вікно, власник якого – уже наявне вікно або підклас класу *Window*.

Створене конструктором вікно не виводиться на екран автоматично. Його належить відобразити методом *show()*. Прибрати вікно з екрану можна методом *hide()*, а перевірити, видивість вікна на екрані – логічним методом *isShowing()*. Вікно типу *Window* можна використовувати для створення спливаючих вікон попередження, повідомлення, підказки. Для створення діалогових вікон є підклас *Dialog*, спливаючих меню – клас *popupMenu*.

Видиме на екрані вікно виводиться на передній план методом *toFront()* або, навпаки, поміщується на задній план методом *toBack()*. Знищити вікно, звільнивши зайняті ним ресурси, можна методом *dispose()*.

ІІІ. Контейнер Dialog та FileDialog.

Контейнер Dialog – це вікно фіксованого розміру, призначене для відповіді на повідомлення додатку. Воно автоматично реєструється у віконному менеджері графічної оболонки, тому його можна переміщати по екрані, змінювати його розміри. Але вікно типу *Dialog*, як і його суперклас – вікно типу *Window,* – обов'язково має власника *owner*, який вказується в конструкторі.

Dialog застосовується для:

- повідомлень;
- підтверджень;
- введення параметрів;
- службових налаштувань.

Вікно типу *Dialog* може бути *модальним* (modal), в якому треба обов'язково виконати всі передбачені дії, інакше із вікна не можна буде вийти. В класі сім конструкторів. З них:

- ***Dialog (Dialog owner)*** – створює немодальне діалогове вікно з пустим рядком заголовка;
- ***Dialog (Dialog owner, string title)*** – створює немодальне діалогове вікно з рядком заголовку *title*;
- ***Dialog(Dialog owner, String title, boolean modal)*** **–** створює діалогове вікно, яке буде модальним, якщо *modal == true*.

Створення Dialog:

```
Frame f = new Frame();
Dialog d = new Dialog(f, "Діалогове вікно");
```

Додавання компонентів у Dialog:

```
d.add(new Label("Введіть дані"));
d.add(new Button("OK"));
```

Контейнер FileDialog – це модальне вікно з власником типу *Frame*, що містить стандартне вікно вибору файлу операційної системи для відкриття (константа LOAD) або збереження (константа SAVE).

Вікна операційної системи створюються і поміщуються в об'єкт класу *FileDialog* автоматично. В контейнері *FileDialog* є три конструктори:

- ***FileDialog (Frame owner)*** – створює вікно з порожнім заголовком для відткриття файлів;
- ***FileDialog (Frame owner, String title)*** **–** створює вікно для відкриття файлів із заголовком *title*;
- ***FileDialog(Frame owner, String title, int mode)*** – створює вікно для відткриття або збереження документа. Аргумент mode має два значення: *FileDialog.LOAD* і *FileDialog.SAVE*.

Методи класу *getDirectory()* і *getFile()* повертають тільки вибраний каталог і ім'я файлу у вигляді рядка *String*. Завантаження або збереження файлу потім потрібно виконувати методами класів введення-виведення. Можна встановити початковий каталог для пошуку файлу і ім'я файлу методами *setDirectory(String dir)* і *setFile(String fileName)*. Замість конкретного імені файлу *fileName* можна написати шаблон, наприклад, **.java*, тоді у вікні будуть видимі тільки імена файлів, що закінчуються крапкою і словом java.

Метод *setFilenameFilter(FilenameFilter filter)* встановлює шаблон *filter* для імені вибраного файла. У вікні будуть видимі тільки імена файлів, що підходять під шаблон. Цей метод не реалізований в SUN JDK на платформі MS Windows.

Контейнер Dialog використовується для: службових повідомлень; введення параметрів; підтвердження операцій. В свою чергу, FileDialog використовується для: відкриття файлів; збереження документів; вибору ресурсів програми.

ЛЕКЦІЯ 10

Тема: *Потоки введення-виведення.*

План:

1. Загальна характеристика потоку в мові Java.
2. Консольне введення-виведення.
3. Файлове введення-виведення.
4. Буферизоване введення-виведення.
5. Прямий доступ до файлу та канали обміну інформацією.
6. Серіалізація об’єктів.
7. Друк в Java 2.

І. Загальна характеристика потоку в мові Java.

У мові Java потік (thread) – це окремий незалежний шлях виконання програми в межах одного процесу. Використання потоків дає змогу організувати паралельне виконання кількох частин програми, що особливо важливо для сучасних застосунків, де необхідно одночасно обробляти декілька задач: виконувати обчислення, працювати з мережею, оновлювати інтерфейс користувача або здійснювати обмін даними.

Під час запуску будь-якої Java-програми автоматично створюється головний потік – main thread, з якого починається виконання методу main(). Однак програміст може створювати додаткові потоки, якщо виникає потреба виконувати декілька операцій незалежно одна від одної.

Таким чином, потік являє собою мінімальну одиницю виконання програми в середовищі JVM.

Потоки в Java мають такі характеристики:

- виконуються в межах одного процесу;
- використовують спільну область пам’яті;
- мають власний стек виконання;
- можуть працювати одночасно або чергуватися в часі.

Основним засобом роботи з потоками є клас Thread, який містить усі механізми керування потоком.

Потік можна створити двома способами: успадкуванням класу Thread; реалізацією інтерфейсу Runnable.

Створення потоку через Thread:

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Потік виконується");
    } }
```

Запуск потоку:

```
MyThread t = new MyThread();
t.start();
```

Метод start() запускає новий потік, а не просто викликає run().

Для того щоб абстрагуватись від особливостей конкретних пристроїв введення-виведення, в Java використовується поняття *потоку* (stream). Вважається, що в програму іде *вхідний потік* символів Unicode або просто байтів, що сприймається в програмі методом *read()*. Із програми методами *write()* або *print (), println()* виводиться *вихідний потік* символів або байтів. При цьому не має значення, куди направлений потік: на консоль, на принтер, у файл або в мережу, методи *write()* і *print()* нічого про це не знають. Звичайно, повне ігнорування особливостей пристроїв введення-виведення сильно сповільнює передачу інформації. Тому в Java також виділяється файлове введення-виведення, виведення на друк, мережевий потік тощо.

Три потоки визначені в класі *System* статичними полями *in, out і err*. Їх можна використовувати без додаткових визначень. Вони називаються відповідно *стандартним введенням* (stdin), *стандартним виведенням* (stdout) і *стандартним виведенням повідомлень* (stderr). Ці стандартні потоки можуть бути з'єднані з різними конкретними пристроями введення-виведення.

Потоки *out* і *err* – це екземпляри класу *Printstream*, організовуючого вихідний потік байтів. Ці екземпляри виводять інформацію на консоль методами *print(), println()* і *write()*, яких в класі *Printstream* є близько двадцяти для різних типів аргументів.

Потік *err* призначений для виведення системних повідомлень програми: трасування, повідомлень про помилки або просто про виконання певних етапів програми. Такі дані звичайно заносяться в спеціальні журнали, log-файли, а не виводяться на консоль. В Java є засоби перепризначення потоку, наприклад, з консолі у файл.

Потік *in* – це екземпляр класу *inputstream*. Він призначений на клавіатурне виведення з консолі методами *read()*. Клас *inputstream* абстрактний, тому реально використовується якийсь із його підкласів.

Методи організації потоків зібрані в класи пакета *java.io.* Крім класів, організуючих потік, в пакет java.io входять класи з методами перетворення потоку, наприклад, можна перетворити потік байтів, утворюючих цілі числа, в потік цих чисел. Ще одна можливість, представлена класами пакету *java.io,* – злиття декількох потоків в один потік.

Створення потоку через Runnable:

```
class MyRunnable implements Runnable {
```

```
    public void run() {
        System.out.println("Новий потік");
    }
}
```

Запуск потоку:

```
Thread t = new Thread(new MyRunnable());
t.start();
```

Потік у Java проходить кілька станів:

- New – створений, але не запущений;
- Runnable – готовий до виконання;
- Running – виконується;
- Blocked / Waiting – очікує;
- Terminated – завершений.

В Java є цілих чотири ієрархії класів для створення, перетворення і злиття потоків. На чолі ієрархії чотири класи, безпосередньо розширюючих клас *Object*:

- ***Reader*** – абстрактний клас, в якому зібрані самі загальні методи символьного введення;
- ***Writer*** – абстрактний клас, в якому зібрані загальні методи символьного виведення;
- ***Inputstream*** – абстрактний клас з загальними методами байтового введення;
- ***Outputstream*** – абстрактний клас з загальними методами байтового виведення.

Класи вхідних потоків *Reader* і *Inputstream* визначають по три методи введення:

- ***read ()*** – повертає один символ або байт, взятий із вхідного потоку, в вигляді цілого значення типу *int*; якщо потік уже закінчився – повертає -1;
- ***read (char[] buf)*** – заповняє заздалегідь визначений масив *buf* символами із вхідного потоку; в класі *Inputstream* масив типу *byte[]* і заповняється він байтами; метод повертає фактичне число взятих із потоку елементів або -1, якщо потік уже закінчився;
- ***read (char[] buf, int offset, int len)*** – заповнює частину символьного або байтового масиву *buf*, починаючи з індексу *offset*, число взятих із потоку елементів рівне *len*; метод повертає фактичне число взятих із потоку елементів або -1.

Класи вихідних потоків *Writer* і *Outputstream* визначають по три методи введення:

- ***write (char[] buf)*** – виводить масив у вихідний потік; в класі *Outputstream* масив має тип *byte[]*;
- ***write (char[] buf, int offset, int len)*** – виводить *len* елементів масиву *buf*, починаючи з елемента із індексом *offset*;
- ***write (int elem)*** – в класі *Writer* виводить 16, а в класі *Outputstream* 8 молодших біт аргументу *elem* у вихідний потік.

В класі Writer є ще два методи:

- ***write (string s)*** – виводить рядок *s* у вихідний потік;
- ***write (String s, int offset, int len)*** – виводить *len* символів рядка *s*, починаючи із символу з номером *offset*.

Всі класи пакету *java.io* можна розділити на дві групи: класи, що створюють потік (data sink), і класи, що керують потоком (data processing).

II. Консольне введення-виведення.

Для виведення на консоль ми використовували метод *println()* класу *Printstream*, ніколи не визначаючи екземпляри цього класу. Ми просто використовували статичне поле *out* класу *System*, яке являється об'єктом класу *PrintStream*. Виконуюча система Java зв'язує це поле з консоллю. Якщо вам набридло писати *system.out.println()*, то можна визначити нове посилання на *System.out*, наприклад:

```
PrintStream pr = System.out;
```

і писати просто *pr.println()*.

Консоль являється байтовим пристроєм, і символи *Unicode* перед виведенням на консоль повинні бути перетворені в байти. Для символів *Latin 1* з кодами '\u0000' – '\u00FF' при цьому просто відкидається нульовий старший байт і виводяться байти '0x00' – '0xFF'. Для кодів кирилиці, які лежать в діапазоні '\u0400' – '\u04FF' кодування *Unicode*, і інших національних алфавітів відбувається перетворення по кодовій таблиці, відповідній установленій на комп'ютері локалі.

Труднощі з відображенням кирилиці виникають, якщо виведення на консоль відбувається в кодуванні, відмінному від локалі. Саме так відбувається в русифікованих версіях MS Windows NT/2000. Звичайно в них встановлюється локаль з кодовою сторінкою CP1251, а виведення на консоль відбувається в кодуванні CP866. В цьому випадку потрібно замінити *Printstream*, який не може працювати з символьним потоком, на *Printwriter* і «вставити перехідне кільце» між потоком символів *Unicode* і потоком байт *System.out*, що виводяться на консоль, у вигляді об'єкта класу *OutputstreamWriter*. В конструкторі цього

об’єкта належить вказати потрібне кодування, в даному випадку CP866. Все це можна зробити одним оператором:

```
PrintWriter pw = new PrintWriter(new
OutputstreamWriter(System.out, "Cp866",true);
```

Клас *Printstream* буферизує вихідний потік. Другий аргумент *true* його конструктора викликає примусове скидання вмісту буфера у вихідний потік після кожного виконання методу *println()*. Для скидання буферу після кожного *print()* треба писати *flush()*.

Введення з консолі відбувається методами *read()* класу *Inputstream* за допомогою статичного поля *in* класу *System*. З консолі йде потік байт, отриманих із scan-кодів клавіатури. Ці байти повинні бути перетворені в символи *Unicode* такими ж кодовими таблицями, як і при виведенні на консоль. Перетворення йде по тій же схемі – для правильного введення кирилиці зручніше всього визначити екземпляр класу *BufferedReader*, використовуючи в якості «перехідного кільця» об’єкт класу *InputstreamReader*:

```
BufferedReader br = new BufferedReader( new
InputstreamReader(System.an,"Cp866"));
```

Клас *BufferedReader* перевизначає три методи *read()* свого суперкласу *Reader*. Крім того, він містить метод *readLine()*, який повертає рядок типу *String*, що містить символи вхідного потоку, починаючи з поточного, і закінчуючи символом *'\n'* або *'\r'*. Ці символи-розділювачі не входять в повернений рядок. Якщо у вхідному потоці немає символів, то повертається null.

III. Файлове введення-виведення.

Файлове введення-виведення в Java призначене для збереження, читання та обробки даних, що розміщуються у зовнішніх файлах. Воно є важливою складовою програмування, оскільки дає змогу працювати з текстовими документами, конфігураційними файлами, журналами виконання програм і різними видами даних, які необхідно зберігати поза межами оперативної пам’яті.

У Java механізми роботи з файлами реалізовані переважно через пакет java.io, який містить класи для:

- створення файлів;
- читання інформації;
- запису даних;
- буферизованої обробки;

– роботи з потоками байтів і символів.

Оскільки файли в більшості сучасних операційних систем розуміються як послідовність байт, то для файлового введення-виведення створюються байтові потоки за допомогою класів *FiIeInputstream* і *FileOutputstream*. Це особливо зручно для бінарних файлів, що зберігають байт-код, архіви, зображення, звук. Але дуже багато файлів містять тексти, складені із символів. Незважаючи на те, що символи можуть зберігатися в кодуванні *Unicode*, ці тексти частіше всього записані в байтових кодуваннях. Тому і для текстових файлів можна використовувати байтові потоки. В такому випадку з боку програми необхідно організовувати перетворення байтів у символи і навпаки.

Щоб полегшити це перетворення, в пакет *java.io* введені класи *FiIeReader* в *FileWriter*. Вони організовують перетворення потоку: із сторони програми потоки символьні, із сторони файлу – байтові. Це відбувається тому, що дані класи розширюють класи *InputStreamReader* і *OutputstreamWriter*, відповідно, містять «перехідне кільце» всередині себе. Незважаючи на відмінність потоків, використання класів файлового введення-виведення дуже схоже. В конструкторах всіх чотирьох файлових потоків задається ім’я файлу у вигляді рядка типу *string* або посилання на об’єкт класу *File*. Конструктори не тільки створюють об’єкт, але і відшуковують файл і відкривають його. Наприклад:

```
Fileinputstream fis = new FilelnputStreamC'PrWr.Java");
FileReader fr = new
FileReader("D:\\jdkl.3\\src\\PrWr.Java");
```

При невдачі відбувається виключення класу *FileNotFoundException*, але конструктор класу *FileWriter* відмічає більш загальне виключення *IOException*. Після відкриття вихідного потоку типу *FileWriter* або *FileQutputStream* вміст файлу, якщо він не був порожнім, стирається. Для того щоб можна було робити запис в кінець файлу, і в тому і в іншому класі передбачений конструктор з двома аргументами. Якщо другий аргумент рівний *true*, то відбувається дозапис в кінець файлу, якщо *false*, то файл заповнюється новою інформацією.

По закінченню роботи з файлом потік належить закрити методом *close()*. Перетворення потоків у класах *FileReader* і *FileWriter* виконується по кодових таблицях, установлених на комп’ютері локалі. Для правильного введення кирилиці треба застосувати *FileReader*, а не *FileInputStream*. Якщо файл містить текст в кодуванні, відмінний від локального кодування, то необхідно вставляти «перехідне кільце» вручну, як це робилось для консолі, наприклад:

```
InputStreamReader isr = new InputStreamReader(fis, "KOI8_R"));
```

Клас *File* містить близько сорока методів, що дозволяють взнати різні властивості файлу або каталогу. Перш за все, логічними методами *isFile(), isDirectory()* можна вияснити, чи являється шлях, вказаний в конструкторі, шляхом до файлу або каталогу. Для каталогу можна отримати його зміст – список імен файлів і підкаталогів – методом *list()*, повертаючим масив рядків *string[]*. Можна отримати такий же список у вигляді масиву об'єктів класу *File[]* методом *listFiles()*. Можна вибрати із списку тільки деякі файли, реалізувавши інтерфейс *FileNameFiiter* і звернувшись до методу *list(FileNameFilter filter)*.

Якщо каталог з вказаним в конструкторі шляхом не існує, його можна створити логічним методом *mkdir()*. Цей метод повертає *true*, якщо каталог вдалось створити. Логічний метод *mkdir()* створює ще і всі неіснуючі каталоги, вказані в шляху. Порожній каталог видаляється методом *delete()*. Для файлу можна отримати його довжину в байтах методом *length()*, час останньої модифікації в секундах методом *lastModified()*. Якщо файл не існує, ці методи повертають нуль. Логічні методи *canRead(), canWrite()* показують права доступу до файлу. Файл можна перейменувати логічним методом *renameTo(File newMame)* або видалити логічним методом *delete()*. Ці методи повертають *true*, якщо операція пройшла успішно. Якщо файл з указаним в конструкторі шляхом не існує, його можна створити логічним методом *createNewFile()*, що повертає *true*, якщо файл не існував, і його вдалось створити, і *false*, якщо файл існував.

Статичними методами:

```
createTempFile(String prefix, String suffix, File
tmpDir)
createTempFile(String prefix, String suffix)
```

можна створити тимчасовий файл з іменем *prefix* і розширенням *suffix* в каталозі *tmpDir* або каталозі, вказаному в системній властивості *java.io.tmpdir*.

Ім'я *prefix* повинно містити не менше трьох символів. Якщо *suffix = null*, то файл одержить суфікс *.tmp*. Перечислені методи повертають посилання типу *File* на створений файл. Якщо звернутися до методу *deleteOnExit()*, то по завершенні роботи JVM тимчасовий файл буде знищений.

IV. Буферизоване введення-виведення.

Буферизоване введення-виведення в Java використовується для підвищення ефективності роботи з файлами та потоками даних. Його сутність полягає в тому, що інформація передається не по одному символу чи байту, а накопичується в спеціальній проміжній області пам'яті – буфері, після чого

обробляється блоками. Такий підхід суттєво зменшує кількість звернень до зовнішніх пристроїв і прискорює виконання програм.

Під час звичайного введення-виведення кожна операція читання або запису супроводжується безпосереднім зверненням до файла чи пристрою. Буферизація дозволяє:

- зменшити кількість фізичних операцій введення-виведення;
- прискорити обробку великих обсягів даних;
- підвищити продуктивність програми.

Операції введення-виведення в порівнянні з операціями в оперативній пам'яті виконуються дуже повільно. Для компенсації в оперативній пам'яті виділяється область буфера обміну даних, в якому поступово накопичується інформація. Коли буфер заповнений, його вміст швидко переноситься процесором, буфер очищується і знову заповнюється інформацією. Життєвий приклад буфера – поштова скринька, в якій накопичуються листи. Ми кидаємо в нього листа і йдемо в своїх справах, не очікуючи приїзду поштової машини. Поштова машина періодично очищує поштову скриньку, переносячи відразу велику кількість листів.

Класи файлового введення-виведення не займаються буферизацією. Для цієї мети є чотири спеціальні класи *BufferedXxx*, перечислені вище. Вони приєднуються до потоків введення-виведення як «перехідне кільце», наприклад:

```
BufferedReader br = new BufferedReader(isr);
BufferedWriter bw = new BufferedWriter(fw);
```

Програма лістингу, поданого нижче, читає текстовий файл, написаний в кодуванні CP866, і записує його вміст у файл в кодуванні KOI8_R.

```
import java.io.*;
class DOStoUNIX{
public static void main(String[] args) throws
IOException{
if (args.length != 2){
System.err.println("Usage: DOStoUNIX Cp866file
KOI8_Rfile");
System.exit(0);
}
BufferedReader br = new BufferedReader(
new InputStreamReader(
new FileInputStream(args[0]), "Cp866"));
BufferedWriter bw = new BufferedWriter(
new OutputStreamWriter(
new FileOutputStreamtargs[1]), "KOI8_R"));
```

```
int c = 0;
while ((c = br.readO) != -1)
bw.write((char)c);
br.closeO; bw.close();
System.out.println("The job's finished.");
} }
```

Під час зчитування і запису застосовується буферизація.

Буферизоване введення-виведення використовується для обробки великих текстових файлів, журналів систем, конфігураційних даних, мережевих потоків.

V. Прямий доступ до файлу та канали обміну інформацією.

Якщо необхідно інтенсивно працювати з файлом, записуючи в нього дані різних типів Java, змінюючи їх, відшукуючи і читаючи потрібну інформацію, то краще всього скористатися методами класу *RandomAccessFile*. В конструкторах цього класу:

```
RandomAccessFile(File file, String mode)
RandomAccessFile(String fileName, String mode)
```

другим аргументом *mode* задається режим відкриття файлу. Це може бути рядок "r" – відкриття файлу тільки для читання, або "rw" – відкриття файлу для читання і запису. Цей клас зібрав всі корисні методи роботи з файлом. Він містить всі методи класів *DataInputstream* і *DataOutputstream*, крім того, дозволяє прочитати відразу цілий рядок методом *readln()* і відшукати потрібні дані у файлі. Байти файлу нумеруються, починаючи з 0, подібно елементам масиву. Файл має неявний покажчик поточної позиції. Читання і запис відбувається, починаючи з поточної позиції файла. При відкритті файлу конструктором, покажчик стоїть на початку файлу, в позиції 0. Поточну позицію можна взнати методом *getFiiePointer()*. Кожне читання або запис переміщує покажчик на довжину прочитаного або записаного даного. Завжди можна перемістити покажчик в нову позицію *pos* методом *seek (long pos)*. Метод *seek(0)* переміщує покажчик на початок файлу. В класі немає методів перетворення символів в байти і назад по кодовим таблицях, тому він не пристосований для роботи з кирилицею.

Значних зусиль вартує організовувати правильний обмін інформацією між підпроцесами. В пакеті *java.io* є чотири класи *pipedxxx*, полегшуючі це завдання. В одному підпроцесі – джерелі інформації – створюється об’єкт класу *PipedWriter* або *PipedOutputstream*, в який записується інформація методами *write()* цих класів. В другому підпроцесі – приймачу інформації – формується об’єкт класу *PipedReader* або *PipedInputstream*. Він зв’язується з об’єктом-

джерелом за допомогою конструктора або спеціальним методом *connect()*, і читає інформацію методами *read()*. Джерело і приймач можна створити і зв'язати в зворотному порядку.

Так створюється *однонаправлений канал* інформації. Це деяка область оперативної пам'яті, до якої організований сумісний доступ двох або більше підпроцесів. Доступ синхронізується, записуючі процеси не можуть завадити читанню. Якщо треба організувати двосторонній обмін інформацією, то створюються два канали.

VI. Серіалізація об'єктів.

Методи класів *ObjectlnputStream* і *ObjectOutputStream* дозволяють прочитати із вхідного байтового потоку або записати у вихідний байтовий потік дані складних типів – об'єкти, масиви, рядки – подібно тому, як методи класів *Datainputstream* і *DataOutputstream*, що читають і записують дані простих типів. Схожість підсилюється тим, що клас *Objectxxx* містить методи як для читання, так і запису простих типів. Між іншим, ці методи призначені не для використання в програмах, а для запису-читання полів об'єктів і елементів масивів.

Процес запису об'єкта у вихідний потік отримав назву ***серіалізації*** (serialization), а читання об'єкта із вхідного потоку і відновлення його в оперативній пам'яті – ***десеріалізації*** (deserialization).

Серіалізація об'єкта порушує його безпеку, оскільки будь-який процес може серіалізувати об'єкт в масив, переписати деякі елементи масиву, представляючі private-поля об'єкта, забезпечивши собі, наприклад, доступ до секретного файлу, а потім десеріалізувати об'єкт із зміненими полями і здійснювати з ним недопустимі дії.

Тому серіалізувати можна не кожний об'єкт, а тільки той, котрий реалізує інтерфейс serializable. Цей інтерфейс не містить ні полів, ні методів. Реалізувати в ньому нема чого. По суті запис:

```
class A implements SeriaiizabIe{...}
```

це тільки мітка, дозволяюча серіалізацію класу A. Як завжди в Java, процес серіалізації максимально автоматизований. Досить створити об'єкт класу *ObjectOutputStream*, зв'язавши його з вихідним потоком, і вивести в цей потік об'єкти методом *writeObject()*:

```
MyClass me = new MyClass("abc", -12, 5.67e-5);
int[] arr = {10, 20, 30};
ObjectOutputStream oos = new ObjectOutputStream(
new FileOutputStream("myobjects.ser")) ;
```

```
oos.writeObject(me);
oos.writeObject(arr);
oos.writeObject("Some string");
oos.writeObject (new Date());
oos.flush();
```

У вихідний потік виводяться всі нестатичні поля об’єкта, незалежно від прав доступу до них, а також дані про клас цього об’єкта, необхідні для його правильного відновлення при десеріалізації. Байт-коди методів класу не серіалізуються. Якщо в об’єкті присутні посилки на інші об’єкти, то вони теж серіалізуються, а в них можуть бути посилання на інші об’єкти, котрі знову-таки серіалізуються, і отримується ціла множина зв’язаних між собою серіалізованих об’єктів. Метод *writeObject()* розпізнає два посилання на один об’єкт і виводить його у вихідний потік тільки один раз. До того ж, він розпізнає посилання, замкнуті в кільце, і уникає зациклювання.

Клас *java.awt.Component* реалізує інтерфейс *Serializable*, а це означає, що всі графічні компоненти можна серіалізувати. Не реалізують цей інтерфейс звичайно класи, тісно пов’язані з виконанням програм, наприклад, *java.awt.Toolkit*. Стан екземплярів таких класів немає рації зберігати або передавати по мережі. Не реалізують інтерфейс *Serializable* і класи, що містять внутрішні дані Java «для службового користування». Десеріалізація відбувається так же просто, як і серіалізація:

```
ObjectlnputStream ois = new ObjectInputStream
(new FilelnputStream("myobjects.ser"));
MyClass mcl = (MyClass)ois.readObject();
int[] a = (int[])ois.readObject();
String s = (String)ois.readObject();
Date d = (Date)ois.readObject();
```

Потрібно тільки додержуватися порядку читання елементів потоку. Процес серіалізації навіть можна повністю налагодити під свої потреби, перевизначивши методи введення-виведення і скористувавшись допоміжними класами.

VII. Друк в Java 2.

Оскільки принтер – пристрій графічний, то виведення на друк дуже схоже на виведення графічних об’єктів на екран. Тому в Java засоби друку входять в графічну бібліотеку AWT і в систему Java 2D. В графічному компоненті крім графічного контексту – об’єкта класу *Graphics*, створюється ще «друкарський контекст». Це теж об’єкт класу *Graphics*, реалізуючий інтерфейс *printGraphics* і отриманий із іншого джерела – об’єкта класу *printjob*, що входить в пакет

java.awt. Сам же цей об'єкт створюється за допомогою класу *Toolkit* пакету *java.awt*. На практиці це виглядає так:

```
PrintJob pj = getToolkitO.get,Print Job (this, "Job Title", null);
                Graphics pg = pj.getGraphics();
```

Метод *getPrintJob()* спочатку виводить на екран стандартне вікно Print операційної системи. Коли користувач вибере в цьому вікні параметри друку і почне друк кнопкою ОК, створюється об'єкт *pj*. Якщо користувач відмовляється від друку кнопкою *Cancel*, то метод повертає *null*. В класі *Toolkit* два методи *getPrint Job ()*:

```
getPrintJob(Frame frame,String jobTitle,JobAttributes jobAttr,
                    PageAttributes pageAttr)
  getPrintJob(Frame frame, String jobTitle, Properties prop)
```

Коротко охарактеризуємо аргументи методів:

- ***аргумент frame*** вказує на вікно верхнього рівня, керуюче друком. Цей аргумент не може бути *null*. Рядок *jobTitle* задає заголовок завдання, який не друкується, і може бути рівним *null*;
- ***аргумент prop*** залежить від реалізації системи друку (часто це просто *null*). В даному випадку задаються стандартні параметри друку;
- ***аргумент jobAttr*** задає параметри друку. Клас *JobAttributes*, екземпляром якого являється цей аргумент, має складну будову. В ньому п'ять підкласів, які містять статичні константи — параметри друку, котрі використовуються в конструкторі класу. Є також конструктор по замовчуванню, задаючий стандартні параметри друку;
- ***аргумент pageAttr*** задає параметри сторінки. Клас *pageProperties* теж містить п'ять підкласів із статичними константами, котрі і задають параметри сторінки і використовуються в конструкторі класу. Якщо для друку досить стандартних параметрів, то можна скористатися конструктором по замовчуванню.

Після того як «друкарський контекст» – об'єкт *pg* класу *Graphics* – визначений, можна викликати метод *print(pg)* або *printAll(pg)* класу *Component*. Цей метод встановлює зв'язок з принтером по замовчуванню і викликає метод *paint(pg)*. На друк виводиться все те, що задано цим методом. Наприклад, щоб роздрукувати текстовий файл, потрібно в процесі введення розбити його текст на рядки і в методі *paint(pg)* вивести рядки методом *pg.drawstring()*. При цьому слід врахувати, що в «друкарському контексті» немає шрифту по замовчуванню, завжди потрібно встановлювати шрифт методом *pg.setFont()*.

Після виконання всіх методів *print()* застосовується метод *pg.dispose()*, викликаючиий перебіг сторінки, і метод *pj.end()*, закінчуючий друк.

Будь-який клас Java 2D, що збирається друкувати хоча б одну сторінку тексту, графіки або зображення називається класом, *малюючим сторінки*. Такий клас повинен реалізувати інтерфейс *Printable*. В цьому інтерфейсі описані дві константи і тільки один метод *print()*. Клас, малюючий сторінки, повинен реалізовувати цей метод. Метод *print()* повертає ціле типу *int* і має три аргументи:

```
print(Graphics g, PageFormat pf, int ind);
```

Перший аргумент *g* – це графічний контекст, що виводиться на лист паперу. Другий аргумент *pf* – екземпляр класу *PageFormat*, визначаючий розмір і орієнтацію сторінки. Третій аргумент *ind* – порядковий номер сторінки, що відраховується від нуля. Метод *print()* класу, малюючого сторінки, заміняє собою метод *paint()*, що використовується стандартними засобами друку AWT. Клас, малюючий сторінки, не зобов'язаний розширювати клас *Frame* і перевизначати метод *paint()*. Все заповнення графічного контексту методами класу *Graphics* або *Graphics2D* тепер виконується в методі *print()*. Коли друк сторінки буде закінчено, метод *print()* повинен повернути ціле значення, задане константою PAGE_EXISTS. Буде зроблено повторне звернення до методу *print()* для друку наступної сторінки. Аргумент *ind* при цьому зростає на 1. Коли *ind* перевищить кількість сторінок, метод *print()* повинен повернути значення NO_SUCH_PAGE, що служить сигналом закінчення друку.

Слід пам'ятати, що система друку може декілька раз звернутися до методу *paint()* для друку однієї і тієї ж сторінки. При цьому аргумент *ind* не змінюється, а метод *print()* повинен створити той же графічний контекст.

Клас *PageFormat* визначає параметри сторінки. На сторінці вводиться система координат з одиницею довжини 1/72 дюйма, початок якої і напрям осей визначається однією із трьох констант:

- ***PORTRAIT*** – початок координат розташовано в лівому верхньому куті сторінки, вісь Ох направлена вправо, вісь Оу – вниз;
- ***LANDSCAPE*** – початок координат в лівому нижньому куті, вісь Ох йде вверх, вісь Оу – вправо;
- ***REVERSE_LANDSCAPE*** – початок координат в правому верхньому куті, вісь Ох йде вниз, вісь Оу – вліво.

Більшість принтерів не може друкувати без полів, на всій сторінці, а здійснює виведення тільки в деяку *область друку*, координати лівого верхнього кута якої повертаються методами *getImageabIeX()* і *getlmageableY()*, а ширина і висота – методами *getlmageableWidth()* і *getlmageableHeight()*. Ці значення треба враховувати при розташуванні елементів в графічному контексті, наприклад, при розміщенні рядків тексту методом *drawstring()*.

Метод *pageDiaIog(PageDiaiog pd)* відкриває на екрані стандартне вікно *Параметри сторінки* операційної системи, в якому уже задані параметри, визначені в об'єкті *pd*. Якщо користувач вибрав у цьому вікні кнопку *Відміна*, то повертається посилання на об'єкт *pd*, якщо кнопку *ОК*, то створюється і повертається посилання на новий об'єкт. Об'єкт *pd* в будь-якому випадку не змінюється. Він звичайно створюється конструктором. Можна задати параметри сторінки і із програми, але тоді слід спочатку визначити об'єкт класу *Paper* конструктором по замовчуванню:

```
Paper p = new Paper()
```

Потім наступними методами задається розмір сторінки і області друку:

```
p.setSize(double width, double height)
p.setlmageableArea(double x, double y, double width, double
height)
```

Далі визначається об'єкт класу *pageFormat* з параметрами по замовчуванню:

```
PageFormat pf = new PageFormat()
```

і задаються нові параметри методом: *pf.setPaper(p).*

Після цього викликати на екран вікно *Параметри сторінки* методом *pageDiaIog()* не обов'язково, оскільки ми отримаємо *мовчазний* процес друку. Так робиться в тих випадках, коли друк виконується в фоновому режимі окремим підпроцесом. Далі треба дати *завдання на друк* (print job) — вказати кількість сторінок, їх номери, порядок друку сторінок, кількість копій. Всі ці дані збираються в класі *PrinterJob*.

Система друку Java 2D розрізняє два види завдань. В більшості простих завдань – *Printable Job* – є тільки один клас, малюючий сторінки, тому у всіх сторінок одні й ті ж параметри, сторінки друкуються послідовно з першої по останню або з останньої сторінки по першу, це залежить від системи друку. Другий, більш складний вид завдань – *Pageable Job* – визначає для друку кожної сторінки свій клас, малюючий сторінки. Тому у кожної сторінки можуть бути власні параметри. Крім того, можна друкувати не все, а тільки вибрані сторінки, виводити їх в зворотному порядку, друкувати на обох сторонах листа. Для здійснення цих можливостей визначається екземпляр класу *Book* або створюється клас, реалізуючий інтерфейс *Pageable*. Залишається задати число копій (якщо воно більше 1), методом *setCopies(int n)* і завдання сформовано.

ЛЕКЦІЯ 11

Тема: *Основні терміни та поняття web-програмування.*

План:

1. Базові відомості про Інтернет та WEB.
2. Основні принципи роботи в Web.
3. Історія PHP.
4. Можливості PHP.

І. Базові відомості про Інтернет та WEB.

Поява комп'ютерів не могла не викликати появи засобів зв'язку їх між собою, оскільки комп'ютери були створені для обробки інформації, а інформацію як відомо потрібно якимось чином одержувати і після обробки передавати. Таким чином комп'ютери спочатку об'єднували в локальні мережі, потім в глобальні мережі, а потім глобальні мережі поєднались між собою. Поєднання глобальних комп'ютерних мереж, що поєднують комп'ютери в усьому світі в єдиному інформаційному просторі, носить назву – Інтернет. Інтернет багатогранний і не можна чітко визначити, що це таке. Він не вирішив проблему збереження й упорядкування інформації, але вирішив проблему її передачі, він дав можливість одержати будь-яку інформацію де завгодно, коли завгодно.

Інтернет можна розглядати в широкому та вузькому розумінні. Найбільш вузький погляд – Інтернет, це мережа мереж, всесвітня комп'ютерна мережа в вузькому розумінні, але більш ширший погляд, Інтернет – це кіберпростір, що народжує кіберкультуру зі своїми думками, своєю мовою, своєю етикою.

Отже, Інтернет – це мережа мереж, яка містить велике число серверів, таких як WWW, електронна пошта, мережні новини USENET, пошукова система WAIS, сервіси: Gopher, FTP, IRC, MUD, MOO, ICQ та інші. Найбільший з цих серверів це WWW – World Wide Web (Всесвітня павутина), який коротко ще називають Web. Web настільки популярний серед людей, що деякі люди, думають, що Web – це і є Internet. Але це невірно, тому що Web – тільки одна з багатьох служб, що використовуються в Internet.

Web була винайдена в 1980 році в CERN (європейської лабораторії фізики елементарних часток). Співробітник цієї лабораторії Тім Бернерс Лі створив програму за назвою Enquire Within Upon Everything (Enquire), щоб відстежити зв'язки між документами, включивши в них посилання один на одного. І, як сказав Тім у 1994 році, даючи інтерв'ю кореспондентам видання Internet World, він хотів знайти більш зручний і логічний спосіб представлення інформації. У 1989 році Бернерс Лі запропонував глобальне узагальнення цієї

ідеї – зв’язати гіпертекстові документи в усьому світі. Інформація повинна міститися на серверах, а для її перегляду потрібно використовувати особливі програми-браузери. З того часу почався бурхливий розвиток Web.

Для роботи в Web необхідна спеціальна програма-клієнт, що називається Web-браузером. Одним з перших браузерів була програма Mosaic, розроблена засновником фірми Netscape Марком Ендрісеном (Marc Andreesen). Перша версія Mosaic була випущена центром NCSA (National Center for Supercomputing Applications – Національний центр суперкомп’ютерних додатків) у 1993 році. У 1994 році Ендрісен залишив NCSA, щоб заснувати фірму Mosaic Communications, на якій почали розробляти браузер Netscape Navigator, який в даний час є найбільшим конкурентом браузера Internet Explorer фірми Microsoft.

II. Основні принципи роботи в Web.

Інтернет має кілька служб, зокрема, електрону пошту, телеконференції, передавання файлів тощо. Більшість документів із різноманітною інформацією з різних галузей знань мають гіпертекстовий формат. Службу Інтернет яка управляє передаванням таких документів, називають WorldWideWeb. Цим терміном або простором Web називають обширну сукупність Web-документів, між якими існують гіпертекстові зв’язки.

Окремі документи, які складають простір Web, називають Web-сторінками. Вони зберігаються на жорстких дисках Web-серверів. Web-сервери – це спеціалізовані комп’ютери з відповідним програмним забезпеченням, яке дає можливість доступу користувачів до їх даних.

Групу сторінок, присвячену певній темі та розміщену в певному каталозі Web-сервера, називають Web-вузлом або Web-сайтом. Один фізичний Web-сервер може містити кілька Web-сайтів. Web-сторінки мають вигляд звичайних текстових документів, в які введено вказівки форматування або елементи так званого гіпертексту.

Теоретично гіпертекст – це всього лише зручний спосіб представлення інформації. Але на практиці гіпертекст – це можливість зробити посилання на інші документи за допомогою слів, фраз, малюнків. Ім’я кожного з цих місць можна зв’язати з іншим документом, у якому міститься більш докладна інформація. Коли користувач вибирає посилання в першому документі, браузер відкриває другий документ із більш докладними даними.

У гіпертексту є дві важливі особливості.

1. Інформація ніяк не впорядковується – документи просто зв’язуються один з одним за допомогою посилань. Хоча головною метою багатьох методів є

саме впорядкування інформації тим або іншим способом (наприклад, у виді ієрархії), у гіпертексті основна увага приділяється створенню інформаційних зв'язків. Таким чином, гіпертекст – це спроба створення моделі, що описує спосіб представлення інформації в мозку людини.

2. Інформаційні зв'язки можуть існувати між різними документами. Створюючи впорядкований список або схему, ви розміщуєте на кожному місці в списку або ієрархії (тобто в структурі) тільки один елемент. А в гіпертексті кожен інформаційний фрагмент (або елемент) може знаходитися в багатьох, причому зовсім різних, місцях структури.

Термін гіпермедіа (hypermedia) використовується для опису того, що ви знаходите в Web. Гіпермедіа – це природне узагальнення поняття гіпертексту, що відноситься до документів, у яких розміщується не тільки текст, але і мультимедіа, тобто зображення, відеозаписи і звук. Ці елементи також можна зв'язувати з іншими документами гіпермедіа. Наприклад, на Web-сторінці можна зв'язати зображення з документом таким чином, що якщо користувач натисне на зображенні, браузер відкриє відповідний документ.

HTML (Hypertext Markup Language – мова гіпертекстової розмітки) служить для опису Web-сторінки, що зберігається у вигляді звичайного текстового файлу з розширенням *.htm або *.html. Головна мета HTML — описати формат вмісту Web-сторінки, він описується з допомогою дескрипторів HTML. Дескриптори визначають способи форматування тексту, служать розпізнавальними знаками зображень або таблиць, дозволяють зв'язувати слова або фрази з іншими документами в Internet.

Під ***гіпертекстовим документом*** розуміють документ, що містить так звані посилання на інший документ. Реалізовано все це через протокол передачі гіпертексту ***HTTP*** (HyperText Transfer Protocol).

Інформація в документах Web може бути знайдена за ключовими словами. Це означає, що кожен оглядач Web містить певні гіпертекстові посилання, через які утворюються так звані гіперзв'язки, що дозволяють мільйонам користувачів Internet вести пошук інформації по всьому світу.

Мова HTML є досить простою. ЇЇ управляючі коди власне і компілюються браузером для відображення на екрані та витікають з ASCII. Існує два типи редакторів: WYSIWYG (What-You-See-Is What-You-Get Що-ти-бачиш-те-і-отримуєш) і редактори, що працюють безпосередньо з HTML-кодом.

HTML оперує всього двома поняттями:

– *тег* – оформлена одиниця HTML-коду. Наприклад, <HEAD>, <BODY>, <HTML> і так далі. Теги бувають початковими (що відкривають) і кінцевими (що закривають, починаються знаком "/"). Наприклад, згаданим вище тегам відповідають закриваючі теги </HEAD>, </BODY>,

</HTML>;

– *елемент* – поняття, введене для зручності. Наприклад, елемент HEAD складається з двох тегів – що відкриває <HEAD> і що закриває </HEAD>. Отже, елемент – більш ширше поняття, що позначає пару тегів і ділянку документа між тегами, на який розповсюджується їх вплив.

Web-сторінки зберігаються у файлах з розширенням *.htm або *.html. Але якщо на Web-сервері використовується технологія ASP (Active Server Page – активні сторінки сервера) чи технологія PHP (Personal Home Page), то замість HTML-сторінок на сервері зберігаються ASP-файли чи PHP файли. ASP-файл чи PHP-файл, містить сценарій, по якому Web-сервер динамічно створює HTML-сторінку в момент звертання до нього браузера. Технології ASP і PHP дозволяють представляти інформацію в більш динамічному вигляді, даючи можливість звертатися до баз даних або до іншої інформації, що неможливо представити у виді звичайних HTML-документів.

Технологія ASP створена фірмою Microsoft та вбудована в офісні додатки Microsoft Office. Тому для користувачів, які знайомі з принципами роботи додатків Microsoft Office кращим рішенням при створенні web-сторінок є використання HTML в поєднанні з технологією ASP.

III. Історія PHP.

PHP (Personal Home Page) – це серверна мова створення сценаріїв.

Мова PHP була розроблена як інструмент для вирішення чисто практичних завдань. Її творець, Расмус Лердорф, хотів взнати, скільки людей читають його online-резюме, і написав для цього простеньку CGI-оболонку на мові Perl, призначену виключно для певної мети – збору статистики відвідування.

CGI (Common Gateway Interface – загальний інтерфейс шлюзів) – технологія, що дозволяє запускати на web-сервері програми, що мають можливість отримувати дані від відвідувачів сайтів, підтримуваних цим web-сервером, і, у свою чергу, видавати їм оброблені дані у вигляді web-сторінок або інших файлів.

Такі застосування (їх називають шлюзами або CGI-програмами) запускаються в режимі реального часу. Сервер передає запити користувача CGI-програмі, яка їх обробляє і повертає результат своєї роботи на екран користувача. Таким чином, відвідувач отримує динамічну інформацію, яка може змінюватися в результаті впливу різних чинників. Сам шлюз (скрипт CGI) може бути написаний на різних мовах програмування – C/C++, Fortran, Perl, TCL, UNIX Schell, Visual Basic, Python і ін.

Незабаром з’ясувалося, що оболонка володіє невеликою продуктивністю, і довелося переписати її, але вже на мові С. Після цього «исходники» були викладені на загальний огляд для виправлення помилок і доповнення. Користувачі сервера, де розташовувався сайт з першою версією PHP, зацікавилися інструментом, з'явилися охочі його використовувати. Отже, швидко PHP перетворився на самостійний проект, і на початку 1995 року вийшла перша відома версія продукту, що називалася Personal Home Page Tools (засоби для персональної домашньої сторінки). Засоби ці були більш ніж скромними: аналізатор коду, що розуміє всього лише декілька спеціальних команд, і набір утиліт, корисних для створення гостьової книги, лічильника відвідувань, чату тощо.

До середини 1995 року після грунтовної переробки з’явилася друга версія продукту, названа PHP/FI (Personal Home Page / Forms Interpreter – персональна домашня страница / інтерпретатор форм). Вона включала набір базових можливостей сьогоднішнього PHP, а також можливість автоматично обробляти html-форми і вбудовуватися в html-коди. Синтаксис PHP/FI сильно нагадував синтаксис Perl, але був простішим.

У 1997 вийшла друга версія С-реалізації PHP – PHP/FI 2.0. До того моменту PHP використовували вже декілька тисяч людей в усьому світі, приблизно з 50 тис. доменів, що складало близько 1% всього числа доменів Internet. Число розробників PHP збільшилося до декількох чоловік, але, не дивлячись на це, PHP/FI 2.0 все ще залишався великим проектом однієї людини. Офіційно PHP/FI 2.0 вийшов тільки в листопаді 1997 року, проіснувавши до цього в основному в бета-версіях. Незабаром після виходу його замінили альфа-версії PHP 3.0.

PHP 3.0 була першою версією PHP, що нагадувала мову прогамування, яка існує сьогодні. Він досить сильно відрізнявся від PHP/FI 2.0 і з’явився знову ж таки як інструмент для вирішення конкретного прикладного завдання. Його творці, Енді Гутманс (Andi Gutmans) і Зів Сураськи (Zeev Suraski), в 1997 році переписали наново код PHP/FI, оскільки він здався їм непридатним для розробки додатку електронної комерції, над яким вони працювали. Для того, щоб отримати допомогу в реалізації проекту від розробників PHP/FI, Гутманс і Сураськи вирішили об’єднатися з ними і оголосити PHP3 офіційним наступником PHP/FI. Після об’єднання розробка PHP/FI була повністю припинена.

Однією з сильних сторін PHP 3.0 була можливість розширення ядра. Саме властивість розширюваності PHP 3.0 привернула увага безлічі розробників, охочих додати свій модуль розширення. Крім того, PHP 3.0 надавала широкі можливості для взаємодії з базами даних, різними протоколами і API.

Важливим кроком до успіху виявилася розробка нового, набагато могутнішого і повнішого синтаксису з підтримкою ООП. З моменту появи PHP 3.0 змінилася не тільки функціональність і внутрішня побудова мови, але і її назва. У абревіатурі PHP більше не було згадки про персональне використання, PHP стало скороченням від PHP: Hypertext Preprocessor, що означає «PHP: препроцесор гіпертексту».

Офіційно PHP 3.0 вийшов в червні 1998 року, після 9 місяців публічного тестування. А вже до зими Енді Гутманс і Зів Сураськи почали переробку ядра PHP. У їх завдання входило збільшення продуктивності роботи складних застосувань і поліпшення модульності коду, що лежить в основі PHP.

Нове ядро було назване «Zend Engine» (від імен творців: Zeev і Andi) і вперше представлено в середині 1999 року. А вже в травні 2000 року вийшов PHP 4.0, заснований на цьому ядрі. Крім поліпшення продуктивності, PHP 4.0 мав ще декілька ключових нововведень, таких як підтримка сесій, буферизація виводу, безпечніші способи обробки інформації, що вводиться користувачем, і декілька нових мовних конструкцій.

П’ята версія PHP була випущена розробниками 13 липня 2004 року. В першу чергу було вирішено підсилити об’єктні можливості мови, що дозволяло використовувати його для реалізації масштабних проектів. PHP 5 володіє прекрасним потенціалом реалізації об'єктного програмування. Окрім цього, PHP збагатився рядом цінних розширень для роботи з XML, різними джерелами даних, генерації графіки і інше. Серед інших украй корисних доповнень в PHP 5 слід зазначити нову схему обробки виключень. Конструкція try/catch/throw дозволяє весь код обробки помилок локалізувати в одному місці сценарію.

Шоста версія PHP знаходилась у стадії розробки з жовтня 2006 року. У ній вже було зроблено безліч нововведень, як, наприклад, виключення з ядра регулярних виразів POSIX і «довгих» суперглобальних масивів, видалення директив safe_mode, magic_quotes_gpc і register_globals з конфігураційного файлу php.ini. Однак її розробка була припинена в 2010 через складнощі з реалізацією підтримки Юнікоду. Тому всі нововведення PHP 6, крім тієї самої підтримки, були реалізовані в PHP 5.3 та PHP 5.4.

У 2014 році було проведено голосування, за результатами якого наступна версія отримала назву PHP 7. Вихід нової версії планувався в середині жовтня 2015. У березні 2015 року Zend представили інфографіку у якій описані основні нововведення PHP 7. Однак реліз нової версії PHP 7 з’явився лише в грудні 2015 року. Ґрунтується вона на експериментальній гілці PHP, яка спочатку називалася phpng (наступне покоління PHP), і розроблялася з вказівкою на збільшення продуктивності та зменшення споживання пам’яті. У новій версії

додана можливість вказувати тип повертаючих з функції даних, а також доданий контроль переданих типів для скалярних даних.

PHP версії 8.0 була випущена 26 листопада 2020-го. Головними нововведеннями стали:

1. Іменовані аргументи:
```
<?php
htmlspecialchars($string, double_encode: false);
```
2. Замість анотацій PHP Doc тепер можна використовувати структуровані метадані з власним синтаксисом PHP
3. Опис властивостей на рівні конструктора:
```
<?php
class Point {
  public function __construct(
    public float $x = 0.0,
    public float $y = 0.0,
    public float $z = 0.0,
  ) {}  }
```
4. Об'єднання типів:
```
<?php
function foo(int|float $a) {}
```
5. Оператор nullsafe. Якщо якийсь об'єкт / метод / властивість не визначена тоді результат буде null:
```
<?php
$country = $session?->user?->getAddress()?->country;
```
6. JIT-компіляція.

IV. Можливості PHP.

Основною особливістю мови PHP є тісна інтеграція з HTML, що дає змогу формувати вебсторінки безпосередньо під час обробки запиту користувача. Завдяки простоті синтаксису, широкій бібліотеці вбудованих функцій і підтримці більшості вебсерверів PHP залишається однією з найпоширеніших технологій серверного програмування. Саме тому в першу чергу PHP використовується для створення скриптів, що працюють на стороні сервера. PHP здатний вирішувати ті ж завдання, що і будь-які інші CGI-скрипти, зокрема обробляти дані html-форм, динамічно генерувати html-сторінки тощо. Але є і інші області, де може використовува-тися PHP. Всього виділяють три основні області застосування PHP:

Перша область – це створення додатків (скриптів), які виконуються на стороні сервера. PHP найширше використовується саме для створення такого роду скриптів. Для того, щоб працювати таким чином, знадобиться PHP-парсер (тобто обробник php-скриптів), web-сервер для обробки скрипта, браузер для проглядання результатів роботи скрипта, і, звичайно, певний текстовий редактор для написання самого php-коду. Парсер PHP розповсюджується у вигляді CGI-програми або серверного модуля.

Друга область – це створення скриптів, що виконуються в командному рядку. Тобто з допомогою PHP можна створювати такі скрипти, які виконуватимуться, незалежно від web-сервера та браузера, на конкретній машині.

Третя область – це створення GUI-прикладних програм (графічних інтерфейсів), що виконуються на стороні клієнта. PHP дуже простий у вивченні. Досить ознайомитися лише з основними правилами синтаксису і принципами його роботи, і можна починати писати власні програми, причому братися за такі завдання, вирішення яких на іншій мові вимагало б серйозної підготовки.

PHP підтримується майже на всіх відомих платформах, майже у всіх операційних системах і на самих різних серверах. Це теж дуже важливо. Також у PHP поєднуються дві найпопулярніші парадигми програмування – об’єктна і процедурна.

Якщо говорити про можливості сьогоднішнього PHP**,** то вони виходять далеко за рамки тих, що були реалізовані в його перших версіях. З допомогою PHP можна створювати зображення, PDF-файли, флеш-ролики, в нього включена підтримка великого числа сучасних баз даних, вбудовані функції для роботи з текстовими даними будь-яких форматів, включаючи XML, і функції для роботи з файловою системою. PHP підтримує взаємодію з різними сервісами за допомогою відповідних протоколів, таких як протокол управління доступом до директорій LDAP, протокол роботи з мережевим устаткуванням SNMP, протоколи передачі повідомлень IMAP, NNTP і POP3, протокол передачі гіпертексту HTTP і т.д. Звертаючи увагу на взаємодію між різними мовами, слід згадати про підтримку об’єктів Java і можливості їх використання як об’єкти PHP. Для доступу до віддалених об’єктів можна використовувати розширення CORBA.

Для створення додатків електронної комерції існує ряд корисних функцій, таких як функції здійснення платежів Cybercash, CYBERMUT, VeriSign Payflow Pro і CCVS.

ЛЕКЦІЯ 12

Тема: *Синтаксис та конструкції управління мови PHP.*

План:

1. Загальні правила побудови PHP-програм.
2. Змінні в PHP.
3. Константи.
4. Типи даних мови програмування PHP.
5. Умовні оператори. Цикли.
6. Оператори передачі управління та включення.

I. Загальні правила побудови PHP-програм.

PHP-програма являє собою сукупність інструкцій, які виконуються на сервері під час обробки запиту користувача. Особливістю PHP є те, що програмний код може безпосередньо вбудовуватися в HTML-документ, що робить цю мову зручною для створення динамічних вебсторінок. Для правильного написання програм необхідно дотримуватися певних синтаксичних правил і принципів організації коду

PHP-програми складаються з простого тексту, тому набирати їх можна в будь-якому текстовому редакторові. Сучасні популярні HTML-редактори мають вбудовану підтримку для редагування PHP-програм.

Розширення файлів PHP-програм за замовчуванням *.php*. На підставі цього розширення сервер розпізнає файл як PHP-програму і запускає інтерпретатор.

PHP-програма повинна бути відокремлена від звичайного HTML-тексту. Існує чотири стилі обрамлення PHP-коду:

Таблиця 12.1 – Стилі обрамлення PHP-коду

Стиль	**Відкриваючий тег**	**Закриваючий тег**
Скорочений	<?	?>
XML (стандартний)	<?php	?>
ASP	<%	%>
SCRIPT (програмний)	<SCRIPT LANGUAGE="php">	</SCRIPT>

З перерахованих тегів тільки стандартні і програмні працюють в будь-якій конфігурації PHP.

Так виглядає простий змішаний документ, що складається з HTML-тексту і PHP-коду:

```
<html>
<head>
 <title>Документ, що складається з HTML-текста і PHP-
коду</title>
</head>
<body>
<?php
echo "Перше знайомство з PHP!<br>";
?>
</body>
</html>
```

В описаному коді програми міститься один PHP-оператор echo. Цей оператор передає рядок-аргумент "Перше знайомство з PHP!
" у HTML-сторінку, яку генерує сервер. При цьому тег
 здійснить перехід на новий рядок. Для розділення операторів (по аналогії з Сі) використовується крапка з комою.

Коментарі в PHP-програмі можуть бути трьох стилів:

- /* Багаторядковий коментар у стилі класичного Сі */;
- // Однорядковий коментар в стилі C++;
- # Однорядковий коментар в стилі Perl.

У одному документі можна чергувати HTML-текст і блоки PHP-операторів довільну кількість разів. При цьому всі змінні, функції і класи, визначені в першому блоці, будуть доступні і в подальших блоках.

Якщо у попередньому прикладі замінити виклик функції `echo` на виклик вбудованої функції phpinfo( ), то відбудеться вивід на екран браузера списку параметрів PHP-середовища.

У великих програмах, написаних із використаням мови програмування PHP рекомендується:

- окремо зберігати логіку;
- окремо шаблони;
- окремо конфігурацію.

Рекомендується також робити відступи, використовувати змістовні назви змінних та коментувати складні фрагменти.

Правильна побудова PHP-програм ґрунтується на дотриманні чітких синтаксичних правил, логічній структурі програмного коду та використанні вбудованих засобів мови. Це створює основу для розроблення надійних вебзастосунків.

ІІ. Змінні в PHP.

У PHP змінна – це іменована область пам'яті, призначена для збереження даних, які можуть змінюватися під час виконання програми. Змінні є основним засобом опрацювання інформації, оскільки через них виконуються обчислення, зберігаються результати, передаються значення між частинами програми та організовується взаємодія з користувачем

Змінні в програмах на PHP, відділяються символами $, наприклад:

```
$country = "Ukraine";
```

де *country* – змінна; *Ukraine* – значення.

Існують деякі правила щодо задання імені змінній. Зокрема, ім'я змінної в PHP:

- починається з літери або символу _;
- може містити літери, цифри, _;
- не може починатися з цифри;
- чутливе до регістру.

В PHP існують наступні операції над змінними:

- арифметичні:

$a + $b – складання «Сума $a і $b»;

$a - $b – віднімання «Різниця $a і $b»;

$a * $b – множення змінних $a і $b;

$a / $b – ділення змінної $a на $b;

$a % $b – Modulus, цілочисельний залишок від ділення $a на $b.

- стрічкові:

є дві стрічкові операції. Перша – операція ('.') – повертає об'єднання з правого і лівого аргументів. Друга − операція привласнення ('.='), яка приєднує правий аргумент влівому аргументу.

```
$a = "Привіт "; $b = $a."ВСІМ!"; //тепер $b містить "Привіт ВСІМ!"
$a = "Привіт "; $a .= "ВСІМ!"; //тепер $a містить "Привіт ВСІМ!"
```

- порівняння змінних: вирази порівняння отримують значення 0 або 1, означаючи FALSE або TRUE (відповідно).

PHP підтримує такі вирази: > (більше), >= (більше або рівно), == (рівно), != (не рівно), < (менше) і <= (менше або рівно). Ці вирази найчастіше використовуються усередині умовних операторів, таких як if.

Змінні в PHP є базовим механізмом збереження й опрацювання даних, а завдяки динамічній типізації забезпечують гнучкість програмування. Зокрема, PHP має спеціальні вбудовані змінні: $_GET; $_POST; $_SERVER; $_SESSION; $_COOKIE.

III. Константи.

У PHP константа – це іменоване значення, яке після оголошення не може бути змінене протягом виконання програми. На відміну від змінних, константи не використовують символ $, а їх значення залишається фіксованим упродовж усього часу роботи сценарію. Використання констант є доцільним у випадках, коли необхідно зберігати сталі параметри програми: математичні значення, налаштування підключення, службові ідентифікатори або конфігураційні параметри.

У PHP константи можна створювати двома основними способами:

- за допомогою функції *define()*;
- за допомогою ключового слова *const.*

Для визначення констант за допомогою функції *define()* синтаксис наступний:

```
define("Імя_константи","Значеннння_константи",[Незалежність від
                                регістру])
```

За замовчуванням імена констант чутливі до регістру. Існує угода, по якій імена констант завжди пишуться у верхньому регістрі. Для кожної константи це можна змінити, вказавши як значення аргументу *Незалежність від регістру* значення *True*.

Надати значення константі можна, вказавши її ім’я. На відміну від змінних, не потрібно ставити перед даним типом символ $. Крім того, для набуття значення константи можна використовувати функцію const () з ім’ям константи як параметр. Наприклад:

```
<?php
define("PASSWORD","qwerty"); // визначаємо константу PASSWORD
define("PI","3.14", True); /* визначаємо регістронезалежну
константу PI із значенням 3.14 */
echo (PASSWORD); // виведе значення константи PASSWORD
echo const("PASSWORD"); // теж виведе значення константи
PASSWORD
echo (password); /* виведе password і попередження, оскільки ми
ввели регістронезалежну константу */
echo pi;// виведе 3.14, оскільки константа PI регістронезалежна
?>
```

Окрім змінних, що оголошуються користувачем, в PHP існує ряд констант, визначених самим інтерпретатором. Наприклад:

- **_FILE_** – зберігає ім’я файла програми (і шлях до нього), яка виконується в даний момент;
- **_FUNCTION_** – містить ім’я функції;

– **_CLASS_** – задає ім’я класу;
– **_PHP_VERSION_** – визначення версії інтерпретатора PHP.

Константи використовують для параметрів підключення до бази даних, службових значень, налаштувань системи, фіксованих математичних величин.

IV. Типи даних мови програмування PHP.

PHP підтримує вісім простих типів даних.

Чотири скалярні типи:

– boolean (логічний);
– integer (цілий);
– float (з плаваючою крапкою);
– string (стрічковий).

Два змішані типи:

– array (масив);
– object (об’єкт).

Два спеціальні типи:

– resource (ресурс);
– NULL.

У PHP не прийняте явне оголошення типів змінних. Розглянемо по порядку всі перераховані типи даних.

Тип **boolean** (булевий або логічний тип). Цей простий тип виражає істинність значення, тобто змінна цього типу може мати лише два значення – істину TRUE або хибність FALSE, наприклад:

```
<?php
  $test = True;
?>
```

Логічні змінні використовуються в різних конструкціях управління (циклах, умовах тощо). Мати логічний тип можуть також і деякі оператори (наприклад, оператор рівності). Вони також використовуються в конструкціях управління для перевірки певних умов.

Тип **integer** (цілі). Цей тип задає число з множини цілих чисел Z = {..., -2, -1, 0, 1, 2, ...}. Цілі можуть бути вказані в десятковій, шістнадцятковій або вісімковій системі числення, за бажанням з попереднім знаком «-» або «+».

Якщо використовується вісімкова система числення, потрібно поставити перед числом 0 (нуль), для використання шістнадцяткової системи потрібно поставити перед числом 0x.

```
<?php
$a = 1234; # десяткове число
```

```
$a = -123; # негативне число
$a = 0123; # вісімкове число (еквівалентно числу 83)
$a = 0x1A;  # шістнадцятиричне число (еквівалентно числу 26)
?>
```

Розмір цілого залежить від платформи, хоча, як правило, максимальне значення близько двох мільярдів (це 32-бітове знакове число). Беззнакові цілі PHP не підтримує.

Якщо число перевищує межі цілого типу, воно буде інтерпретовано як число з плаваючою крапкою. У випадку, коли використовується оператор, результатом роботи якого буде число, що перевищує межі цілого, замість нього буде повернене число з плаваючою крапкою.

У PHP не існує оператора ділення цілих. Результатом ділення 1 на 2 буде число з плаваючою крапкою 0.5. Ви можете привести значення до цілого, що завжди округлює його в меншу сторону, або використовувати функцію round(), що округлює значення по стандартних правилах. Для перетворення змінної до конкретного типу потрібно перед змінною вказати в дужках потрібний тип. Наприклад, для перетворення змінної $a=0.5 до цілого типу необхідно написати (integer)(0.5) або (integer) $a чи використовувати скорочений запис (int)(0.5).

Тип **float**. Числа з плаваючою крапкою (вони ж числа подвійної точності або дійсні числа) можуть бути визначені за допомогою будь-якого з наступних синтаксисів:

```
<?php
  $a = 1.234;
  $b = 1.2e3;
  $c = 7E-10;
?>
```

Розмір числа з плаваючою крапкою залежить від платформи, хоча максимум, як правило, ~1.8e308 з точністю близько 14 десяткових цифр.

Тип **string**. Рядок – це набір символів. У PHP символ – це те ж саме, що байт, а це означає, що існує рівно 256 різних символів.

У PHP практично не існує обмежень на розмір рядків, тому немає абсолютно жодних причин турбуватися про їх довжину.

Рядок в PHP може бути визначений трьома різними способами:

– за допомогою одинарних лапок;

– за допомогою подвійних лапок.

Одинарні лапки. Простий спосіб визначити рядок – це вставити її в одинарні лапки «'». Для того, щоб використовувати одинарну лапку усередині рядка, як і в багатьох інших мовах, перед нею необхідно поставити символ оберненої косої риски «\», тобто екранувати її. Якщо обернена коса риска

повинна йти перед одинарною лапкою або бути в кінці рядка, необхідно продублювати її «\\'».

Якщо усередині рядка, поміщеного в одинарні лапки, є знак зворотнього слеша «\», то він розглядається як звичайний символ і виводиться, як і всі інші. Тому обернену косу риску необхідно екранувати, лише якщо вона знаходиться в кінці рядка, перед закриваючою лапкою.

Подвійні лапки. Якщо рядок поміщений в подвійні лапки «"», то PHP розпізнає більшу кількість керуючих послідовностей для спеціальних символів. Деякі з них наведені в таблиці 12.2.

Таблиця 12.2 – Керуючі послідовності в PHP

Послідовність	**Значення**
\n	Новий рядок (LF або 0x0A (10) в ASCII)
\r	Повернення каретки (CR або 0x0D (13) в ASCII)
\t	Горизонтальна табуляція (HT або 0x09 (9) в ASCII)
\\	Обернена коса риска
\$	Знак долара
\"	Подвійна лапка

Найважливішою властивістю рядків в подвійних лапках є обробка змінних.

Тип **array** (масив). Масив в PHP є впорядкованою картою – типом, який перетворює значення в ключі. Цей тип оптимізований в декількох напрямах, тому можна використовувати його як власне масив, список (вектор), хеш-кодування-таблицю (що є реалізацією карти), стек, черга тощо.

Визначити масив можна за допомогою конструкції *array ()* або безпосередньо задаючи значення його елементам.

Мовна конструкція *array ()* приймає параметри пари *ключ => значення*, розділені комами. Символ => встановлює відповідність між значенням і його ключем. Ключ може бути як цілим числом, так і рядком, а значення може бути будь-якого наявного в PHP типу. Числовий ключ масиву часто називають індексом. Індексування масиву в PHP починається з нуля. Приклад визначення масиву за допомогою array():

```
array ([key] => value [key1] => value1 ...)
```

Значення елемента масиву можна надати, вказавши після імені масиву в квадратних дужках ключ шуканого елемента. Якщо ключ масиву є стандартним записом цілого числа, то він розглядається як число, в інших випадках – як

рядок. Тому запис $a["1"] рівносильний запису $a[1], так само як і $a["-1"] рівносильне $a[-1]:

```
<?php
  $books = array ("php" =>"PHP для початківців", 12 => true);
  echo $books["php"]; // виведе "PHP для початківців"
  echo $books[12];    // виведе 1
?>
```

Якщо для елементу ключ не заданий, то як ключ береться максимальний числовий ключ, збільшений на одиницю. Якщо вказати ключ, якому вже було привласнене якесь значення, то воно буде перезаписане.

Якщо використовувати як ключ TRUE або FALSE, то його значення переводиться відповідно в одиницю та нуль типу integer. Якщо використовувати NULL, то замість ключа отримаємо порожній рядок. Можна використовувати і сам порожній рядок як ключ, при цьому його вотрібно помістити в лапки. Як ключ не можна використовувати масиви і об'єкти.

Створити масив можна просто записуючи в нього значення. Значення елемент масиву може набути за допомогою квадратних дужок, усередині яких потрібно вказати його ключ наприклад $book["php"]. Якщо вказати новий ключ і нове значення наприклад $book["new_key"]="new_value", то в масив буде доданий новий елемент. Якщо ключ не буде вказано, а лише здійчнено привласнення значення $book[]="new_value", то новий елемент масиву матиме числовий ключ, на одиницю більший максимального що вже існує. Якщо масив, в який додаємо значення, ще не існує, то він буде створений.

```
<?php
  // додали в масив $books значення  value1 з ключем key
  $books["key"]= value1;
  // додали в масив значення value2 з ключем 13,
  // оскільки максимальний ключ у нас був 12
  $books[] = value2; ?>
```

Для того, щоб змінити конкретний елемент масиву, потрібно просто привласнити йому з його ключем нове значення. Змінити ключ елементу не можна, можна лише видалити елемент (пару ключ/значение) і додати нову. Видалити елемент масиву здійснюється способом використання функції unset().

```
<?php
  $books = array ("php" => "PHP users guide", 12 => true);
  // додали елемент з ключем (індексом)  13 це еквівалентно
  // $books[13]= "Book about Perl";
  $books[] = "Book about Perl";
  // Це додає до масиву новий елемент з ключем "lisp"
```

```
  // і значенням 123456
  $books["lisp"]= 123456;
  // Це видаляє елемент з ключем 12 з масиву
  unset($books[12]);
  // Видаляє масив повністю
  unset ($books);
?>
```

Коли використовуються порожні квадратні дужки, максимальний числовий ключ шукається серед ключів, що існують в масиві з моменту останнього переіндексування. Переіндексувати масив можна за допомогою функції *array_values()*.

Тип **object**. *Об'єкт* – тип даних, що прийшов з об'єктно-орієнтованого програмування (ООП). Згідно принципам ООП, клас – це набір об'єктів, що володіють певними властивостями і методами роботи з ним, а об'єкт відповідно – екземпляр класу.

У PHP для доступу до методів об'єкта використовується оператор ->. Для ініціалізації об'єкта використовується вираз new, що створює в змінній екземпляр об'єкту.

Тип **resource** (ресурси). Ресурс – це спеціальна змінна, що містить посилання на зовнішній ресурс (наприклад, з'єднання з базою даних). Ресурси створюються і використовуються спеціальними функціями (наприклад, mysql_connect(), pdf_new() тощо).

Тип **Null**. Спеціальне значення NULL говорить про те, що змінна не має значення. Змінна вважається NULL, якщо:

- їй була привласнена константа NULL ($var = NULL);
- їй ще не було привласнено яке-небудь значення;
- вона була видалена за допомогою unset().

V. Умовні оператори. Цикли.

Оператор if – це один з найважливіших операторів багатьох мов, включаючи PHP. Він дозволяє виконувати фрагменти коду залежно від умови. Структуру оператора if можна представити таким чином:

```
if (вираз) блок_виконання
```

У процесі обробки скрипту з даним оператором вираз перетвориться у логічний тип. Якщо в результаті перетворення значення виразу істинне (True), то виконується *блок_виконання*, інакше *блок_виконання* ігнорується. Якщо *блок_виконання* містить декілька команд, то він повинен бути поміщений у фігурні дужки { }.

Правила перетворення виразу до логічного типу:

1. У FALSE перетворяться наступні значення:

– логічне False;
– цілий нуль (0);
– дійсний нуль (0.0);
– порожній рядок і рядок "0";
– масив без елементів;
– об’єкт без змінних;
– спеціальний тип NULL.

2. Решта всіх значень перетворюється в TRUE.

Приклад використання оператора *if:*

```
<?
$names = array("Іван","Петро","семен");
if ($names[0]=="Іван") {
    echo "Привіт, Іван!";
    $num = 1;
    $account = 2000;
}
if ($num) echo "Іван перший в списку!";
$bax = 30;
if ($account > 100*$bax+3)
    echo "Ця стрічка не з'явиться
    на екрані, оскільки умова не виконана";
?>
```

Оператор else. Існує декілька розширень оператора if. Оператор else розширює if на випадок, якщо вираз, що перевіряється в if є невірним, і дозволяє виконати за таких умов певні дії.

Структуру оператора if, розширеного за допомогою оператора else, можна представити таким чином:

```
if (вираз) блок_виконання
else блок_виконання1
```

Цю конструкцію if...else можна інтерпретувати приблизно так: якщо виконана умова (тобто вираз = true), то виконуємо дії з *блоку_виконання*, в інакшому випадку – дії з *блоку_виконання 1*.

Подивимося, як можна змінити попередній приклад, враховуючи необхідність здійснення дій і у разі невиконання умови:

```
<?
$names = array("Іван","Петро","Семен");
if ($names[0]=="Іван")
{ echo "Привіт, Іван!";
```

```
    $num = 1;
    $account = 2000;}
else {
    echo "Привіт $names[0].
   А ми чекали Івана:(";
}
if ($num) echo "Іван перший в списку!";
else echo "Іван не перший в списку?!";
$bax = 30;
if ($account > 100*$bax+3)
    echo "Цей рядок не з'явиться на екрані
    оскільки умова не виконана";
    else echo "Зате з'явиться цей рядок!";
?>
```

Оператор elseif. Способом розширення умовного оператора if – є також використання оператора elseif. elseif – це комбінація else і if. Як і else, він розширює if для виконання різних дій в тому випадку, якщо умова, що перевіряється в if, невірна. Але на відміну від else, альтернативні дії будуть виконані в тому випадку, якщо elseif-умова є вірною. Структуру оператора if, розширеного за допомогою операторів else і elseif, можна представити таким чином:

```
if (вираз) блок_виконання
elseif(вираз1) блок_виконання1
...
else блок_виконанняN
```

Операторів elseif може бути відразу декілька в одному if-блоці. Elseif-твердження буде виконане тільки якщо попередня if-умова є False, всі попередні elseif-умови є False, а дана elseif-умова – True:

```
<?
$names = array("Іван","Петро","Семен");
if ($names[0]=="Іван") {
 // якщо перше ім'я в масиві Іван
    echo "Привіт, Іване!";
}elseif ($names[0] == "Петро"){
 // якщо перше ім'я не Іван, а Петро
    echo "Привіт, Петре!";
}elseif ($names[0] == "Семен"){
 // якщо перше ім'я не Іван, не Петро, а Семен
    echo "Привіт, Семене!";
}else {
 // якщо перше ім'я не Іван не Петро і не Семен
    echo "Привіт я $names[0]. А ти хто такий?";
} ?>
```

PHP пропонує альтернативний синтаксис для деяких своїх структур управління, а саме для if, while, for, foreach і switch. У кожному випадку відкриваючу дужку потрібно замінити на двокрапку (:), а закриваючу – на endif, endwhile тощо.

Наприклад, синтаксис оператора if можна записати таким чином:

```
if(вираз):блок_виконання endif;
```

Сенс залишається тим же: якщо умова, записана в круглих дужках оператора if, виявилася істиною, виконуватиметься весь код, від двокрапки «:» до команди endif.

Використання такого синтаксису корисне під час вбудовування php в html-код.

```
<?php
  $names = array("Іван","Петро","Семен");
  if ($names[0]=="Іван"): echo 'Привіт, Іван!'; endif;
?>
```

Альтернативний синтаксис також може використовуватися в конструкціях else і elseif.

Оператор switch. Конструкція switch дозволяє перевіряти умову і виконувати залежно від цього різні дії. Назву даного оператора можна перевести як «перемикач».

Структуру switch можна записати таким чином:

```
switch (вираз або змінна){
case значення1:
    блок_дій1
break;
case значення2:
    блок_дій2
break;
...
default:
    блок_дій_за_замовчуванням
}
```

На відміну від if, тут значення виразу не приводиться до логічного типу, а просто порівнюється із значеннями, перерахованими після ключових слів *case* (значення 1, значення 2 і т.д.). Якщо значення виразу співпало з якимсь варіантом, то виконується відповідний *блок_дій* – від двокрапки після значення, що співпало, до кінця *switch* або до першого оператора *break,* якщо такий є. Якщо значення виразу не співпало ні з одним з варіантів, то виконуються дії за замовчуванням (*блок_дій_за_замовчуванням*), що знаходяться після ключового

слова *default*. Вираз в switch обчислюється тільки один раз, а в операторі *elseif* – кожного разу.

Приклад з використанням оператору *switch*:

```
<?
$names = array("Іван","Петро","Семен");
switch ($names[0]){
case "Іван":
    echo "Привіт, Ваня!";
break;
case "Петро":
    echo "Привіт, Петре!";
break;
case "Семен":
    echo "Привіт, Семене!";
break;
default:
    echo "Привіт $names[0].
   А як Вас звуть?";
}
?>
```

Якщо в даному прикладі опустити оператор *break*, наприклад, в *case "Петро"*:, то у випадку коли змінна опиниться рівною рядку "Петро" – після виводу на екран повідомлення "Привіт, Петре!" програма піде далі і виведе також повідомлення "Привіт, Семене!" і тільки потім, зустрівши *break*, продовжить своє виконання за межами *switch*.

Для конструкції *switch*, як і для *if*, можливий альтернативний синтаксис, де відкриваюча *switch* фігурна дужка замінюється двокрапкою, а закриваюча – *endswitch* відповідно.

У PHP існує декілька конструкцій, що дозволяють виконувати дії, що повторюються, залежно від умови. Це цикли *while, do..while, foreach* і *for*. Розглянемо їх детальніше.

Структура ***циклу while***:

```
while (вираз) { блок_виконання }   або   while (вираз):
                  блок_виконання endwhile;
```

while – це простий цикл. Він наказує PHP виконувати команди *блоку_виконання* до тих пір, поки вираз обчислюється як *True* (тут, як і в if, відбувається приведення виразу до логічного типу). Значення виразу перевіряється кожного разу на початку циклу, так що, навіть якщо його значення змінилося в процесі виконання *блоку_виконання*, цикл не буде

зупинений до кінця ітерації (тобто поки всі команди *блоку_виконання* не будуть виконані).

```
<?
//ця програма надрукує всі парні цифри
    $i = 1;
    while ($i < 10) {
        if ($i % 2 == 0) print $i;
    // друкуємо цифру, якщо вона парна
        $i++;
    // збільшуємо $i на одиницю
    }
?>
```

Цикли do..while дуже схожі на цикли *while*, з тією лише різницею, що істинність виразу перевіряється в кінці циклу, а не на початку. Завдяки цьому *блок_виконання* циклу *do...while* гарантовано виконується хоч би один раз.

Структура циклу:

```
do { блок_виконання} while (вираз);
```

Приклад використання оператору do..while:

```
<?
// ця програма надрукує число 12, не дивлячись на те
// що умова циклу не виконана
$i = 12;
    do{
        if ($i % 2 == 0) print $i;
    // якщо число парне, то друкуємо його
        $i++;
    // збільшуємо число на одиницю
    }while ($i<10)
?>
```

For – найскладніші цикли в PHP.

Структура:

```
for (вираз1; вираз2; вираз3) {блок_виконання}
```

або

```
for (вираз1; вираз2; вираз3): блок_виконання endfor;
```

Умова складається відразу з трьох виразів. Перший вираз *вираз1* обчислюється безумовно один раз на початку циклу. На початку кожної ітерації обчислюється *вираз2*. Якщо він є *True*, то цикл продовжується і виконуються всі команди *блоку_виконання*. Якщо *вираз 2* обчислюється як *False*, то виконання циклу зупиняється. В кінці кожної ітерації (тобто після виконання всіх команд блоку_виконання) обчислюється *вираз3*.

Кожен з виразів 1, 2, 3 може бути порожнім. Якщо *вираз 2* є порожнім, то це означає, що цикл повинен виконуватися невизначений час (в цьому випадку PHP вважає цей вираз завжди істинним). Наприклад, всі парні цифри можна вивести з використанням циклу *for* таким чином:

```
<?php
for ($i=0; $i<10; $i++){
    if ($i % 2 == 0) print $i;
    // друкуємо парні числа }
?>
```

Якщо опустити другий вираз (умова $i<10), то це рівнозначно задачі зупинки циклу оператором break.

```
<?php
for ($i=0; ; $i++){
    if ($i>=10) break;
// якщо $i більше або рівне 10, то припиняємо роботу циклу
    if ($i % 2 == 0) print $i;
// якщо число парне, то друкуємо його
}
?>
```

Можна опустити всі три вирази. В цьому випадку просто не буде задане початкове значення лічильника $i і воно не змінюватиметься жодного разу в кінці циклу. Всі ці дії можна записати у вигляді окремих команд або в *блоці_виконання*, або перед циклом:

```
<?php
$i=0; // задаємо початкове значення лічильника
for ( ; ; ){
    if ($i>=10) break;
// якщо $i більше або рівне 10, то припиняємо роботу циклу
    if ($i % 2 == 0) print $i;
// якщо число парне, то друкуємо його
$i++; // збільшуємо лічильник на одиницю  }
?>
```

Конструкція ***foreach*** призначена виключно для роботи з масивами. Синтаксис:

```
foreach ($array as $value) {блок_виконання}
або
foreach ($array as $key => $value)
   { блок_виконання}
```

У першому випадку формується цикл за всіма елементами масиву, заданого змінною $array. На кожному кроці циклу значення поточного

елементу масиву записується в змінну $value, і внутрішній лічильник масиву пересувається на одиницю (отже на наступному кроці буде видний наступний елемент масиву). Усередині *блоку_виконання* значення поточного елементу масиву може бути набуте за допомогою змінної $value. Виконання *блоку_виконання* відбувається стільки раз, скільки елементів в масиві $array.

Друга форма запису до перерахованого вище на кожному кроці циклу записує ключ поточного елемента масиву в змінну $key, яку теж можна використовувати в *блоці_виконання*.

```
<?php
$names = array("Іван","Петро","Семен");
foreach ($names as $val) {
    echo "Привіт $val <br>";
    // виведе всім вітання
}
foreach ($names as $k => $val) {
    // окрім вітання виведуться номери в списку,
тобто ключі
    echo "Привіт $val !
     Ти в списку під номером $k <br>";
}
?>
```

Коли *foreach* починає виконання, внутрішній покажчик масиву автоматично встановлюється на перший елемент.

VI. Оператори передачі управління та включення.

Іноді у разі особливих обставин потрібно негайно завершити роботу циклу і передати управління першій інструкції програми, що розташована за останньою інструкцією циклу. Для цього використовують операторів break і continue.

Оператор break закінчує виконання поточного циклу for, foreach, while, do..while або switch. *Break* може використовуватися з числовим аргументом, який повідомляє роботу скількох структур управління, що містять його, потрібно завершити:

```
<?php
$i=1;
while ($i) {
    $n = rand(1,10);
    // генеруємо довільне число
    // від 1 до 10
    echo "$i:$n ";
```

```
    // виводимо номер ітерації і число, що
згенерувалось
    if ($n==5) break;
/* Якщо згенерувало число 5, то припиняємо роботу
циклу. В цьому випадку все, що знаходиться після цієї
стрічки усередині циклу, не буде виконано */
    echo "Цикл працює <br>";
    $i++;
}
echo "<br>Число ітерацій циклу $i ";
?>
```

Результатом роботи цього скрипта буде:

```
1:7 Цикл працює
2:2 Цикл працює
3:5 Число ітерацій циклу 3
```

Якщо після оператору *break* вказати число, то припинеться саме така кількість циклів, що містять цей оператор.

Оператор continue дозволяє пропустити подальші інструкції з *блоку_виконання* будь-якого циклу і продовжити виконання по новому колу. *Continue* можна використовувати з числовим аргументом, який вказує, скільки управлінських конструкцій, що містять його, повинні завершити роботу.

Приклад з використанням оператора *continue*:

```
<?php
$i=1;
while ($i<4) {
    $n = rand(1,10);
   // генеруємо довільне число від 1 до 10
    echo "$i:$n";
   // виводимо номер ітерації і число, що згенерувалось
    if ($n==5) {
        echo "Нова ітерація";
        continue;
/* Якщо згенерувало число 5 то починаємо нову ітерацію
циклу
$i не збільшується */
    }
    echo "Цикл працює <br>";
    $i++;
}
echo "<br>Число ітерацій циклу $i ";
?>
```

Результатом роботи цього скрипта буде:

```
1:10 Цикл працює
2:5 Нова ітерація 2:1 Цикл працює
3:1 Цикл працює
Число ітерацій циклу 4
```

Відмітимо, що після виконання оператора *continue* робота циклу не закінчується. У прикладі лічильник циклу не закінчується у разі отримання числа 5, оскільки він знаходиться після оператора *continue*. Фактично за допомогою *continue* ми намагаємося уникнути ситуації, коли згенерує число 5. Тому можна було здійснити заміну оператора *continue* на оператор перевірки істинності виразу.

У PHP існує одна особливість використання оператора *continue* – в конструкціях *switch* він працює так само, як і *break*. Якщо *switch* знаходиться усередині циклу і слід почати нову ітерацію циклу, використовуємо *continue 2*.

Оператор include дозволяє включати код, що міститься у вказаному файлі, і виконувати його стільки раз, скільки програма зустрічає цей оператор.

Включення може проводитися будь-яким з перерахованих способів:

```
include 'ім'я_файлу';
include $file_name;
include ("ім'я_файлу");
```

Наведемо приклад застосування оператора *include*. Нехай у файлі *params.inc* у нас зберігається набір певних параметрів і функцій. Кожного разу, коли нам потрібно буде використовувати ці параметри (функції), вставлятимемо в текст основної програми команду *include 'params.inc'*.

```
params.inc
<?php
$user = "Тарас";
$today = date("d.m.y");
/* функція date() повертає дату і час */
?>

include.php
<?php
include ("params.inc");
/* змінні $user і $today задані у файлі
params.inc. Тут ми теж можемо ними користуватися
завдяки команді include("params.inc")*/
echo "Привіт $user!<br>";
    // виведе "Привіт, Тарас!"
echo "Сьогодні $today";
  // виведе, наприклад, "Сьогодні (поточна дата)"
?>
```

Використання оператора *include* еквівалентне простій вставці змістовної частини файлу *params.inc* у код програми *include.php*. У момент вставки файлу відбувається перемикання з режиму обробки PHP в режим HTML. Тому код, що

включається потрібно обробити як PHP-скрипт. Для цього він повинен бути поміщений у відповідні теги.

Пошук файлу для вставки відбувається за наступними правилами:

1. Спочатку ведеться пошук файлу в *include_path* щодо поточної робочої директорії.

2. Якщо файл не знайдений, то пошук проводиться в *include_path* щодо директорії поточного скрипту.

3. Параметр *include_path*, визначуваний у файлі налаштувань PHP, задає імена директорій, в яких потрібно шукати файли, що включаються.

Якщо файл включений за допомогою *include*, то код, що міститься в ньому, успадковує область видимості змінних рядка, де з'явився *include*. Будь-які змінні викликаного файлу будуть доступні в викликаному файлі з цього рядка і далі.

Окрім локальних файлів, за допомогою *include* можна включати і зовнішні файли, вказуючи їх url-адресу. Дана можливість контролюється директивою *url_fopen_wrappers*, яка, як правило, включена у файл налаштувань PHP за замовчуванням.

Під час використання *include* можливі два види помилок – помилка вставки (наприклад, не можна знайти вказаний файл, невірно написана сама команда вставки тощо.) або помилка виконання (якщо помилка міститься у файлі, що вставляється). У будь-якому випадку за помилки в команді *include* виконання скрипту не завершується.

Оператор require у мові програмування PHP використовується для підключення зовнішнього файла до поточного сценарію. Його основне призначення полягає у включенні коду, який повинен бути обов'язково доступним для подальшого виконання програми. Це дозволяє розділяти великий програмний код на окремі логічні модулі: файли конфігурації, бібліотеки функцій, шаблони інтерфейсу або модулі підключення до бази даних.

Оператор require виконує:

- підключення іншого PHP-файла;
- вставлення його вмісту у поточний сценарій;
- подальше виконання коду як єдиного цілого.

Загальний синтаксис: require "ім'я_файла.php";

Нехай існує файл config.php:

```
<?php
define("SITE", "Навчальний сайт");
?>
```

Основна програма:

```
<?php
require "config.php";

echo SITE;
?>
```

Після підключення константа стає доступною у головному файлі.

Якщо файл не знайдено, виконання програми зупиняється з фатальною помилкою:

```
require "unknown.php";
```

У цьому випадку сценарій припиняє роботу.

Оператор *require* застосовується тоді, коли без файла програма не може працювати.

Найчастіше *require* застосовується для підключення: параметрів конфігурації; функціональних бібліотек; файлів підключення до бази даних. Наприклад: *require "db.php";*

Оператор require_once. Для уникнення повторного підключення використовується:

```
require_once "config.php";
```

Перевага: файл підключається лише один раз, коли доцільно використовувати *require_once*.

Особливо важливо використовувати його при програмуванні великих проектів та багаторівневому підключенні файлів, а також роботі з класами.

Приклад модульної структури:

```
require "header.php";
 require "menu.php";
require "footer.php";
```

Це дозволяє будувати структуровані вебзастосунки. Зокрема, оператор require досить часто використовується для здійснення: повторного використання коду; централізованого керування налаштуваннями; побудови модульної архітектури.

Оператори ***require*** та ***require_once*** є важливими засобами організації PHP-програм, оскільки дозволяють розділяти код на окремі модулі, підвищувати його читабельність і забезпечувати надійне підключення необхідних компонентів.

ЛЕКЦІЯ 13

Тема: *Інструменти та технології програмування на стороні клієнта і сервера.*

План:

1. Клієнт-серверна архітектура.
2. Моделі взаємодії клієнт-сервер.
3. Програмування на стороні клієнта.
4. Інструменти і технології web-програмування.

І. Клієнт-серверна архітектура.

Більшості розробників сайтів, веб-сервісів і мобільних додатків рано чи пізно доводиться мати справу з клієнт-серверною архітектурою, а саме: розробляти web API або інтегруватися з ним.

Клієнт-серверна архітектура – обчислювальна чи мережева архітектура, в якій завдання чи мережеве навантаження розподілені між постачальниками послуг (сервісів), котрі називають серверами, і замовниками послуг, котрі називають клієнтами (рис. 13.1).

Сервер – це програма, що надає деякі послуги іншим програмам і обслуговує запити клієнтів на отримання ресурсів певного виду.

Клієнт – це програма, що використовує послугу, надану програмою сервера.

Рисунок 13.1 – Схематичне зображення клієнт-серверної архітектури

Клієнт-серверна архітектура є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

– набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;

– набір клієнтів, які використовують сервіси, що надаються серверами;
– мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив’язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Загальноприйнятим є положення, що клієнти та сервери – це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми – і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов’язків між клієнтом та сервером. Можна відокремити три рівні операцій:

– рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
– прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
– рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура (рис. 13.2) передбачає взаємодію двох програмних модулів – клієнтського та серверного.

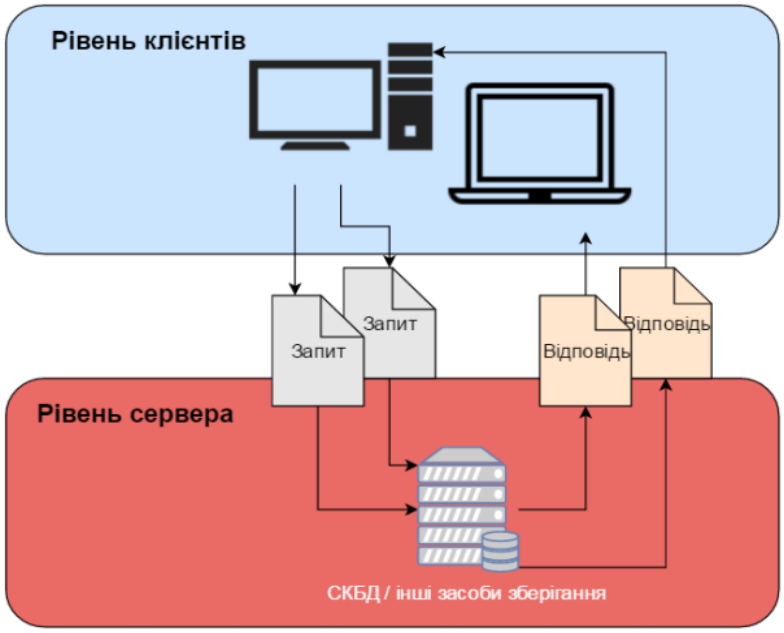

Рисунок 13.2 – Дворівнева клієнт-серверна архітектура

В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

- модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
- модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

У кінці 1990-х років з'являється так звана *багаторівнева клієнт-серверна архітектура*. Логічно модель має таку ж саму структуру (рис. 13.3), але всеохоплююче використання Інтернету внесло свої корективи, ставши важливою частиною багатьох програмних додатків.

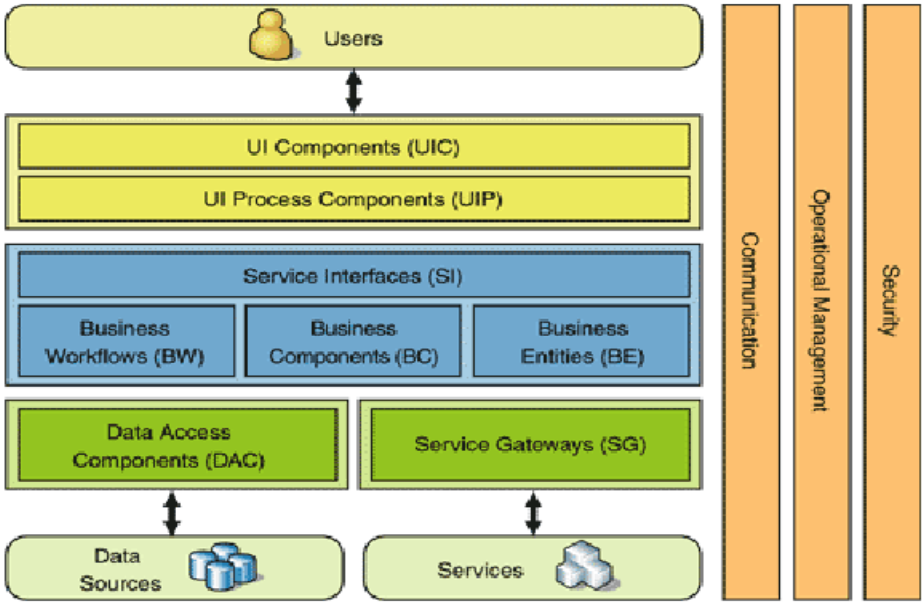

Рисунок 13.3 – Багаторівнева клієнт-серверна архітектура

Великі корпоративні додатки часто структуровані навколо бізнес-процесів та бізнес-компонентів. Ці поняття розглядаються в рамках цілого ряду компонентів, сутностей, агентів та інтерфейсів бізнес-рівня:

- *бізнес-компоненти* – програмні реалізації концепцій чи процесів. Вони складаються з усіх артефактів необхідних для представлення, реалізації, розгортування конкретної концепції як автономного елемента більшої системи, котрий можна використовувати повторно.
- *бізнес-сутності* – це структури, що виступають контейнерами даних. Вони інкапсулюють та приховують деталі специфічного формату представлення даних. Наприклад, бізнес сутність може інкапсулювати набір записів, отриманих з бази даних. Пізніше, ця ж бізнес-сутність може

бути змінена для огортання в XML-документ з мінімальним впливом на інші частини додатку.

– *сервісні інтерфейси* – додаток може надавати частину його функціоналу як сервіс, котрий можуть використовувати інші додатки. В ідеалі він приховує деталі реалізації і надає тільки тонкий шар інтерфейсу.
– *бізнес-процеси* – відображають діяльність бізнесу на високих рівні абстракції системи, зокрема обробка замовлення, підтримка користувача, закупка товару.

II. Моделі взаємодії клієнт-сервер.

У своєму розвитку технології «клієнт-сервер» пройшли кілька етапів, тому є різні моделі технології. Їх реалізація заснована на поділі структури СУБД на три компоненти:

– введення і відображення даних (інтерфейс з користувачем);
– прикладний компонент (запити, події, правила, процедури та функції, які характерні для даної предметної області);
– функції керування ресурсами (файловою системою, базою даних тощо).

Зв'язок між компонентами здійснюється за певними правилами, які називають «протокол взаємодії».

Існують різні класифікації, але однією з найпоширеніших є використання чотирьох моделей технології «клієнт-сервер»:

Модель файлового серверу (File Server – FS) – це природне розширення персональних СУБД для підтримки багатокористувацького режиму і в цьому плані ще довго буде зберігати своє значення.

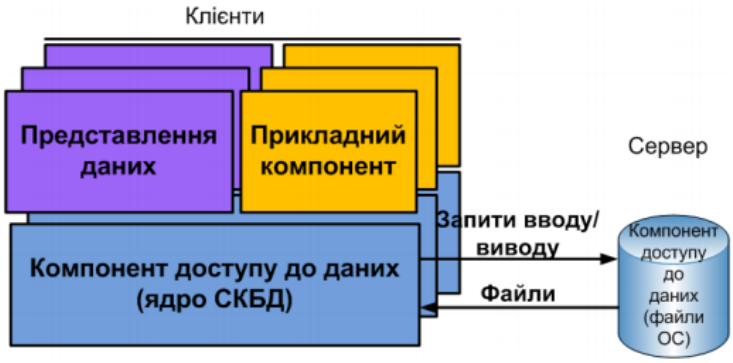

Рисунок 13.4 – Схема взаємодії FS-моделі

Особливості FS-моделі: всі основні компоненти розміщуються на клієнтським комп'ютері; модель характеризує не стільки спосіб створення ІС, скільки загальний спосіб взаємодії комп'ютерів в локальній мережі; один з комп'ютерів виділяється і визначається файловим сервером, тобто загальним

сховищем будь-яких даних; сервер виконує чисто пасивну функцію; дуже проста та зрозуміла модель.

У стандартній файл-серверній архітектурі дані, розташовуючись на файл-сервері, є, по суті, пасивним джерелом. Вся відповідальність за їх отримання, обробку, а також за підтримку цілісності бази даних лежить на додатку, запущеному з робочої станції. При цьому, оскільки обробка даних здійснюється на робочій станції, по мережі переганяється вся необхідна для цієї обробки інформація, хоча обсяг даних, які цікавлять користувача, може бути менше в десятки разів.

На основі моделі файлового сервера функціонують такі популярні СУБД як FoxPro (Microsoft), dBase (Borland), CF-Clipper (Computer Associates International), Paradox (Borland) тощо. СУБД розглянутого класу коштують недорого, прості в установці та освоєнні. Також відсутні високі вимоги до продуктивності сервера та програмні компоненти СУБД не розподілені.

Модель віддаленого доступу до даних (Remote Data Access – RDA) – архітектура, яка заснована на обліку специфіки розміщення і фізичного маніпулювання даними у зовнішній пам'яті для реляційних СУБД (рис. 13.5).

У RDA-моделі для обробки даних виділяється спеціальне ядро – так званий SQL-сервер, який приймає на себе функції обробки запитів користувачів, іменованих тепер клієнтами. Сервер баз даних являє собою програму, яка виконується, як правило, на потужному комп'ютері.

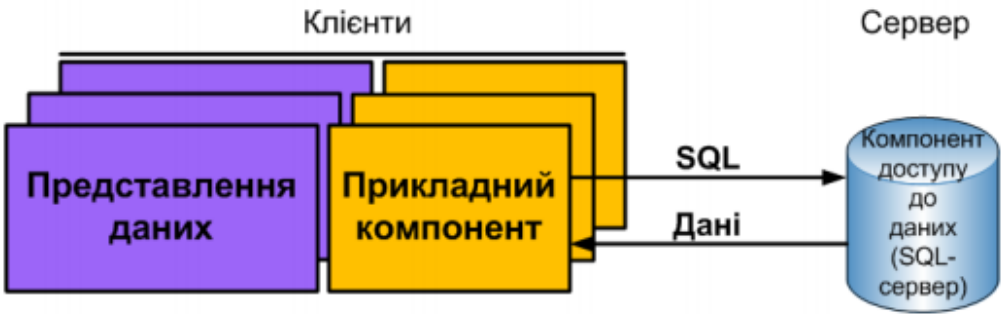

Рисунок 13.5 – Схема взаємодії RDA-моделі

Тут додатки також виконуються, в основному, на робочих станціях. Додаток включає модулі для організації діалогу з користувачем і бізнес-правила (транзакції). Ядро СУБД є загальним для всіх робочих станцій і функціонує на сервері. Оператори звернення до СУБД (SQL-оператори), закодовані в транзакції, не виконуються на робочій станції, а пересилаються для обробки на сервер. Ядро СУБД транслює запит і виконує його, звертаючись для цього до індексів та інших проміжних

даних. Назад на робочу станцію передаються тільки результати обробки оператора.

Модель сервера бази даних. Для поліпшення попередньої RDA-моделі в сучасних СУБД використовується модель сервера баз даних (DataBase Server – DBS), в якій на сервері можуть запускатися так звані збережені процедури і тригери, які разом з ядром СУБД утворюють сервер бази даних (рис. 13.6).

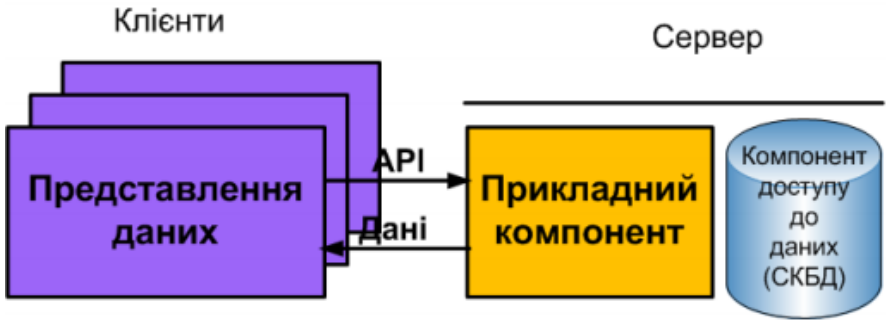

Рисунок 13.6 – Схема взаємодії DBS-моделі

До збережених процедур можна звертатися з додатків на робочих станціях. Це дозволяє скоротити розмір коду прикладної програми і зменшити потік SQL-операторів з робочої станції, так як групу необхідних SQL-запитів можна закодувати в збереженій процедурі.

Тригери – це програми, які виконуються ядром СУБД перед або після оновлення (UPDATE, INSERT, DELETE) таблиці бази даних. Вони дозволяють автоматично підтримувати цілісність бази даних.

Модель сервера бази даних (БД) підтримують наступні СУБД: Oracle, DB2 (IBM), MS SQL Server (Microsoft), MySQL (Oracle), FireBird, PostgreSQL, Sybase (SAP), тощо. Причому на перші чотири СУБД припадають понад 85% ринку. СУБД розглянутого класу мають наступні переваги:

- системи, мають високу продуктивність, тому що запити виконуються на високошвидкісних серверах;
- зниження мережевого трафіку, тому що по шині передаються тільки SQL-запити і результати з виконання;
- СУБД підтримують розподілену обробку;
- більш гнучке «налаштування» на предметну область;
- забезпечення узгодженого стану даних;
- надійність зберігання і обробки даних;
- ефективна координація колективної роботи користувачів із загальними даними;

– в рамках цих СУБД пропонується велика кількість сервісних продуктів, що полегшують розробку додатків і створення розподіленої системи.

Ці СУБД мають і недоліки:

– вони набагато дорожче СУБД попереднього класу, складні в освоєнні;
– для ефективної роботи цих СУБД потрібні високошвидкісні (а тому й дорогі) сервери та мережі.

Модель сервера додатків. Щоб рознести вимоги до обчислювальних ресурсів сервера у відношенні швидкодії і пам'яті за різними машинам, використовується модель сервера додатків (Application Server AS). AS-модель зберігає сильні сторони DBS-моделі.

Особливості AS-моделі (рис. 13.7):

– перенесення прикладного компонента АІС на спеціалізований сервер;
– на клієнтських машинах – тільки інтерфейсна частина системи;
– виклики функцій обробки даних спрямовуються на сервер додатків;
– низькорівневі операції з даними виконує SQL-сервер.

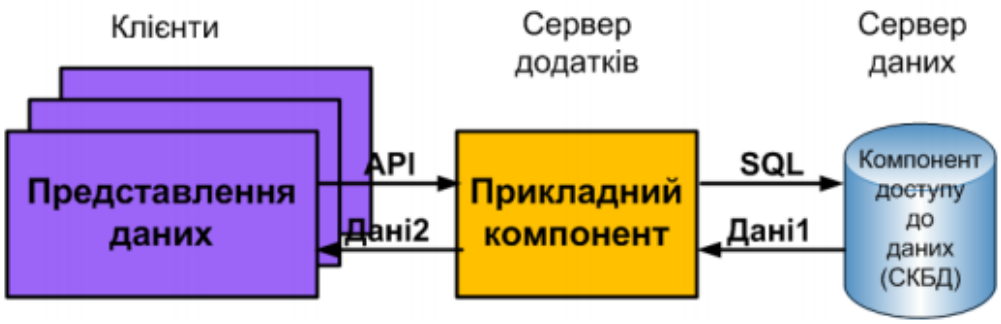

Рисунок 13.7 – Модель сервера додатків

У тому випадку, коли інформаційна система об'єднує досить велику кількість різних інформаційних ресурсів і серверів додатків, постає питання про оптимальне управління всіма її компонентами. Для цього використовують програмні засоби, які часто називають менеджерами транзакцій, моніторами транзакцій (Transaction Processing Monitor - TPM), і які розміщуються на сервері додатків. Також менеджер транзакцій і додатки можуть запускаютися на одному комп'ютері (хоча це не є обов'язковим), щоб зменшити потік SQL-запитів по мережі.

Транзакція – це послідовна сукупність операцій над даними (SQL-інструкцій), що має окреме смислове значення.

Для спілкування прикладної програми з монітором транзакцій використовується спеціалізований API (Application Program Interface).

ІІІ. Програмування на стороні клієнта.

Для того, щоб людина, яка працює в Інтернеті, могла переглянути ту чи іншу сторінку, на її комп'ютері повинно бути встановлено відповідне програмне забезпечення – браузер (веб-оглядач). Найпоширеніші браузери: Google Chrome, Internet Explorer, Firefox, Safari і Opera.

Але, крім браузерів, до серверів можуть звертатися і інші клієнти, а саме – автономні програми. Вони можуть передбачати взаємодію з людиною, а можуть працювати в цілком автоматичному режимі. Типовим класом таких програм є роботи, призначені для автоматичного перегляду веб-ресурсів. Зокрема, роботи є важливим елементом пошукових систем і використовуються ними для перегляду сторінок і збору інформації про них.

На сучасному етапі для програмування модулів проміжного рівня використовується мова серверних сценаріїв PHP, а для управління даними – СУБД MySQL. Таким чином, зв'язку PHP-MySQL слід розглядати як стандартний інструмент для створення порівняно простих інтерактивних веб-сайтів та систем електронної комерції; близько 90% комерційних систем сьогодні створюється саме на цій основі. Водночас як засоби управління даними, так і middleware-засоби можуть бути найрізноманітнішими. Так, для створення серверних застосунків, крім PHP, широко застосовуються Java, Perl, Python. Для зберігання даних та їх передачі часто використовується так звана розширювана мова розмітки.

Для здійснення ***програмування на стороні клієнта*** використовуються наступні інструментальні засоби:

Мова програмування JavaScript – найпопулярніша і підтримувана всіма сучасними Інтернет-браузерами мова програмування. Вона створена для того, щоб зробити веб-сторінки «живими». Скрипти у браузері підключаються безпосередньо до HTML і, як тільки завантажується сторінка — тут же виконуються.

React – це бібліотека JavaScript, яка використовується для створення призначеного для користувача інтерфейсу.

VueJS – це JavaScript бібліотека для створення веб-інтерфейсів з використанням шаблону архітектури MVVM (Model-View-ViewModel).

AngularJS – це JavaScript фреймворк призначений для створення односторінкових веб-додатків.

jQuery – це бібліотека JavaScript, призначена для спрощення скриптинга при роботі з вузлами HTML-елементів в браузері або для роботи в браузері без графічного інтерфейсу.

AJAX – це бібліотека, яка значно спрощує і прискорює написання JavaScript коду, також дозволяє працювати з усіма браузерами.

Таким чином, комбінуючи різні засоби та технології, будь вони клієнтські або серверні, розробник може отримати велику кількість різних інтерактивних можливостей.

IV. Інструменти і технології web-програмування.

Відомо чимало інструментальних засобів та технологій web-програмування. Ми розглянемо найпопулярніші з них, на яких ґрунтується переважна частина сайтів. Але спочатку наведемо основні терміни, що застосовуються в web-програмуванні.

Додаток – програмний продукт, що працює під управлінням операційної системи комп'ютера.

Web-хостинг – послуга, що включає надання дискового простору, підключення до мережі та інших ресурсів для розміщення фізичної інформації на сервері, що постійно перебуває в мережі. Суть його така: програміст створює текстові файли з розширенням *.html, *.php або інші, розмішує їх на комп'ютері в мережі (такий комп'ютер називається сервером) за договором з власником даного сервера (що називається хостером). Хостер присвоює директорії на сервері, надані програмісту, доменне ім'я, за яким кожен користувач Інтернет може переглядати ці файли шляхом набору доменного імені в адресному рядку Web-браузера.

Web-сервер – це додаток, встановлений на комп'ютері в мережі (на сервері), який аналізує код файлу, написаного web-програмістом і розміщеного в одній з папок цього сервера. На підставі аналізу web-сервер формує код, який відправляє по мережі web-браузера. Існує чимало видів Web-серверів, але одним з найпопулярніших з них є Apache.

Мови сценаріїв – це мови програмування, які не потребують компілятора коду, потрібен тільки інтерпретатор, який обробляє код і видає ту ж HTML-сторінку. Існують серверні (інтерпретатор мови встановлений на сервері) і клієнтські мови сценаріїв.

Найбільш потужні, популярні і прості мови сценаріїв – клієнтська JavaScript і серверна мова PHP. JavaScript служить для підвищення динамічності Web-сторінок. PHP забезпечує обмін даними між клієнтським Web-браузером і Web-сервером, обробку запитів, надісланих браузером, на сервері і видачу даних за запитом браузера.

Інтерпретатор сценаріїв – додаток, який зазвичай встановлюють на сервері і який працює спільно з Web-сервером. Служить для обробки коду Web-

сторінки, написаного програмістом. Оброблений код Web-сервер посилає через мережу Web-браузеру.

Технології побудови клієнтського програмного забезпечення ґрунтуються на використанні різних типів протоколів. Одним із них є протокол TCP/IP, який використовує сокети. Сокети, використовувані протоколом TCP/IP, високостандартизировані і широкодоступні.

Іншим типом протоколу є протокол віддаленого виклику процедур RPC (Remote Procedure Call). Але протокол віддаленого виклику процедур RPC є доволі складним, і до того ж відомо велику кількість його різновидів. А також є доволі популярними такі протоколи високого рівня, як CORBA (Common Object Request Broker Architecture – архітектура посередника об'єктних запитів), RMI (Remote Method Invocation – технологія віддаленого виклику методів) і розподілена модель компонентних об'єктів DCOM (Distributed Component Object Model). Ці протоколи усе ще складні і для організації їх роботи потрібно наявність спеціального середовища як на стороні сервера, так і на стороні клієнта.

Саме через повсюдне поширення протоколу HTTP, компанії Microsoft і іншим виробникам мережного програмного забезпечення довелося розробити новий протокол, що одержав назву SOAP (Simple Object Access Protocol – простий протоколу доступу до об'єктів). Для кодування запитів методів об'єктів і супутніх даних у протоколі SOAP використовуються тексти мовою XML (Extensible Markup Language). Великою перевагою протоколу SOAP є його простота. Внаслідок своєї простоти цей протокол може бути легко реалізований на багатьох пристроях. Протокол SOAP (Simple Object Access Protocol) може працювати на верхньому рівні будь-якого стандартного протоколу.

Тенденції розвитку клієнт-серверних систем приводять до переносу функціональності клієнта в спеціалізоване програмне забезпечення, яке також перебуває по стороні сервера і завантажується на робочу станцію користувача при його під'єднанні до інформаційної системи. При цьому робоча станція користувача надає лише середовище для виконання клієнтського програмного забезпечення.

У web-програмуванні використовують такі ***технології***: HTML, XML, DHTML, CSS, OLE/COM, ACTIVEX, DCOM, COM +, COM.NET, Corba, WAP, J2EE, SQL, ADO, ADO.Net, ADOCE, ADOX .Net Compact Framework, ***мови програмування***: C, C ++, ASP, ASP.NET, VB.Net, C #, PHP, Perl, ColdFusion, Python, CGI, JavaScript, VBScript, JScript, ActiveScript (Flash), ***бази даних***: MS SQL Server, MS Access, MYSQL, POSTGRESQL, Oracle, Informix, DB2, Sybase,

Paradox, InterBase, LotusNotes, DBase, FoxPro, FoxBase, Clarion і ***Web-сервери***: MS Internet Information Server (MS IIS), Apache, WebSphere, Tomcat Apache.

Необхідно також розуміти різницю між мовою програмування і фреймворком.

Мова програмування – це просто певний базовий синтаксис (можливо, зі стандартними бібліотеками), за допомогою якого можна створювати додатки.

Фреймворк забезпечує програміста різними бібліотеками, що значно спрощують створення програм і сайтів.

Програмний фреймворк (*software framework*) – це готовий до використання комплекс програмних рішень, включаючи дизайн, логіку та базову функціональність системи або підсистеми.

Деякі мови і фреймворки є нерозривним цілим (наприклад, ASP.NET і JSP). Інші мови можуть використовуватися без фреймворку (PHP і Perl).

Незалежно від того, яка мова програмування буде обрана, основу кожного сайту становить мова гіпертекстової розмітки – HTML. ЇЇ повинні знати всі Web-розробники.

Найпоширенішою клієнтською мовою є JavaScript, розробниками якої були компанія Netscape спільно з компанією SunMicrosystems. Інший варіант клієнтської мови – Visual Basic Script (VBS).

Інші популярні клієнтські мови, а точніше, фреймворки – це Adobe Flash (мова ActionScript) і SilverLight. Adobe Flash застосовується Web-майстрами дуже давно. Головне застосування цієї технології – інтерактивні сайти і сервіси, онлайнові ігри, мультимедійний контент і реклама.

SilverLight – це відносно нова технологія, розроблена компанією Microsoft, яка покликана замінити Adobe Flash. Хоча за допомогою Adobe Flash або SilverLight можна побудувати весь сайт цілком, так робити не слід. Справа в тому, що пошукові системи поки не вміють індексувати ні Adobe Flash, ні SilverLight.

До найпоширеніших інструментальних засобів Web-програмування відносяться системи керування вмістом (СКВ або CMS).

Система керування вмістом (англ. Content Management System) – програмне забезпечення для організації веб-сайтів чи інших інформаційних ресурсів в Інтернеті чи окремих комп'ютерних мережах.

Системи управління веб-сайтом часто розраховані на роботу у певному програмному середовищі. Наприклад, система MediaWiki, під управлінням якої працює Вікіпедія, написана мовою програмування PHP і зберігає вміст і налаштування у базі даних типу MySQL або PostgreSQL. Тому для її роботи потрібно, щоб на сервері, де вона розміщена, були встановлені веб-сервер (Apache, IIS чи інший), підтримка PHP та системи керування базами

даних MySQL або PostgreSQL, а також, в разі необхідності, додаткові програми для обробки зображень чи математичних формул. Такі вимоги є досить типовими для відкритих СКВ.

Існують CMS двох видів: комерційні та вільно поширювані. Більшість поширених безкоштовних CMS надають безкоштовну підтримку за допомогою спільноти на власних форумах або ж спеціалізованих email-розсилок (наприклад Joomla, WordPress та Drupal). Крім того, достатньо поширеною є модель, при якій, власне, сама CMS надається безкоштовна, проте користувач може придбати платну техпідтримку. Для поширених безкоштовних систем діють системи сертифікації, які дозволяють отримати підтримку високого рівня від незалежних розробників.

Щоб обрати систему управління, яка найліпшим чином підійде для створення певного ресурсу, необхідно розглянути представлені на сучасному ринку CMS.

WordPress – є наразі найбільш відомим безкоштовним движком. Ця CMS найчастіше використовується для ведення блогів та створення інформаційних ресурсів. Для роботи з нею не потрібні якісь спеціальні навички, адже движок інтуїтивно зрозумілий навіть для новачка. До того ж, функціонал WordPress можна розширювати за допомогою спеціальних плагінів.

Drupal підходить для створення найбільш складних інтернет-сторінок з можливістю редагування як самого сайту, так і його дизайну. Ця CMS написана мовою програмування PHP.

MODx – ще один безкоштовний движок, головною відмінною рисою якого є можливість створення сайту будь-якої складності з будь-яким функціоналом без впливу на HTML-код.

Серед платних СКВ слід виділити **1С-Bitrix**, яка є продуктом компанії 1С. Наразі вона вважається найякіснішою CMS, яка підходить для створення великих порталів, соціальних мереж, інтернет-магазинів.

DLE – найбільш популярна CMS для організації новостійних ресурсів. Вона дозволяє легко публікувати, редагувати та будь-яким чином налаштовувати новини. Крім цього, грамотна організація структури ядра цього движка мінімізує вимоги до серверу, на якому розташований сайт.

NetCat дозволяє створювати великі інтернет-портали, медійні ресурси, бібліотеки даних, файл-архіви. До її сильних сторін можна віднести поділ адміністративної панелі на 2 частини: для користувача та розробника.

Комерційні CMS поділяються на два типи: системи із закритим кодом (вихідний код закодований (кріптованний) і не допускає будь-яких змін); системи з відкритим кодом (для внесення зміни будь-якої з функціональних можливостей вихідний код відкритий).

ЛЕКЦІЯ 14

Тема: *Програмування на стороні сервера: HTTP та CGI, передача параметрів серверу.*

План:

1. Програмування на стороні сервера.
2. Протокол HTTP. CGI.
3. Передача параметрів серверу та запам'ятовування стану.
4. Міри безпеки.

І. Програмування на стороні сервера.

Сучасні технології створення динамічних сайтів передбачають використання програмного коду, що забезпечує інтерактивність Web-сторінок, і називається *сценарієм* (*скриптом*).

Скриптова мова (*scripting language, мова сценаріїв*) – мова програмування, розроблена для запису «сценаріїв», послідовностей операцій, які користувач може виконувати на комп'ютері. Сценарії, зазвичай, інтерпретуються, а не компілюються.

Розрізняють сценарії, що виконуються на стороні клієнта і такі, що виконуються на стороні сервера. Сценарії на стороні клієнта *(клієнтські скрипти)* виконуються під керуванням браузера, на стороні сервера (серверні скрипти) – під керуванням Web-сервера.

Серверні скрипти виконуються на стороні сервера під керуванням Web-сервера. Коли користувач дає запит на яку-небудь сторінку (переходить на неї по посиланню або вводить адресу в адресному рядку свого браузера), то викликана сторінка спочатку обробляється на сервері, тобто виконуються всі скрипти, пов'язані із формуванням даної сторінки, і тільки потім вона повертається до відвідувача у вигляді HTML-документа (відвідувач ніяк не зможе побачити код серверного скрипта).

Серверні скрипти потрібні, наприклад, у випадку, коли web-сервер повинен повертати різну інформацію для різних людей, а також інформацію, яка змінюється з плином часу (найбільш актуальний приклад – форум).

Серверні скрипти, як правило, взаємодіють з БД. У цьому випадку скрипти виконують запити до СУБД, остання повертає результати (запити-дані), а скрипт формує документ, який передається браузеру. Найбільш часто використовуються СУБД Mysql, PostreSQL, MS SQL Server, Oracle.

Робота серверних скриптів залежить від сервера, на якому розташований сайт, і від того, які технології підтримуються сервером.

Залежно від розв'язуваних задач для створення сайту вибирають ту чи іншу мову серверних скриптів. Для створення малих і середніх інтерактивних сайтів доцільно застосовувати мову сценаріїв PHP. Перевагою мови PHP є те, що вона є безкоштовною, має відкриті вихідні коди і працює майже на всіх платформах. Також використовують мови Perl, Python, тощо. Конкурентами PHP є технології ASP.NET, JSP, Cold Fusion, Perl.

Загалом, серед основних засобів програмування на стороні сервера, крім мови PHP, виділяють:

- *Node.js* – це середовище для виконання вашого JavaScript коду, це просто ще один спосіб виконувати код на вашому комп'ютері;
- *ASP.NET.* – модель для розробки веб-додатків із застосуванням мінімуму коду, яка містить служби, необхідні для побудови веб-додатків для підприємств;
- *Python* – це мова програмування загального призначення, націлений в першу чергу на підвищення продуктивності самого програміста;
- *Ruby on Rails* – веб-орієнтоване середовище розробки з відкритим кодом, оптимізована для зручності програмування та стійкої продуктивності.

Для підтримки роботи динамічних сайтів необхідно, щоб сервер мав спеціалізоване програмне забезпечення, що працює з БД і створює сторінки динамічно. Це програмне забезпечення називається веб-застосуваннями.

Зазвичай, веб-застосування створюються як додатки в архітектурі «клієнт-сервер»: клієнтська частина реалізує користувальницький інтерфейс, формує запити до сервера і обробляє відповіді від нього; серверна частина отримує запит від клієнта, виконує обчислення, після цього формує веб-сторінку і відправляє її клієнтові по мережі з використанням протоколу HTTP.

II. Протокол HTTP. CGI.

HTTP (Hyper Text Transfer Protocol) – протокол передачі даних, що використовується в комп'ютерних мережах. HTTP належить до протоколів моделі OSI 7-го прикладного рівня.

Основним призначенням протоколу HTTP є передача веб-сторінок (текстових файлів з розміткою HTML), хоча за допомогою нього успішно передаються і інші файли, які пов'язані з веб-сторінками (зображення і застосунки), так і не пов'язані з ними (у цьому HTTP конкурує з складнішим FTP).

HTTP припускає, що клієнтська програма – веб-браузер – здатна відображати гіпертекстові веб-сторінки та файли інших типів у зручній для користувача формі. Для правильного відображення HTTP дозволяє клієнтові

дізнатися мову та кодування символів веб-сторінки й/або запитати версію сторінки в потрібних мові/кодуванні, використовуючи позначення із стандарту MIME.

CGI – це набір правил, згідно яким програми на сервері можуть через веб-сервер посилати дані клієнтам. Специфікація CGI супроводилася змінами в HTML і HTTP, що вводили нову характеристику, відому як форми.

Якщо CGI дозволяє програмам посилати дані клієнтові, то форми розширюють цю можливість, дозволяючи клієнтові посилати дані для цієї CGI-програми. Поширені застосування CGI включають:

- Динамічний HTML. Цілі сайти можуть генеруватися однією CGI-програмою.
- Пошукові механізми, що знаходять документи із заданими користувачем словами.
- Гостьові книги і дошки оголошень, в які користувачі можуть додавати свої повідомлення.
- Бланки замовлень.
- Анкети.
- Витягання інформації з розміщеної на сервері бази даних.

Всі вони дають можливість з'єднання CGI з базою даних, що нас особливо цікавить.

Специфікація CGI. Офіційну специфікацію CGI разом з масою інших відомостей про CGI можна знайти на сервері NCSA за адресою http://hoohoo.ncsa.uiuc.edu/cgi/. Є чотири способи, якими CGI передає дані між CGI-програмою і веб-сервером-сервером, а отже, і клієнтом Web: змінні оточення; командний рядок; стандартний пристрій введення; стандартний пристрій виведення. За допомогою цих чотирьох методів сервер пересилає всі дані, передані клієнтом, CGI-програмі. Потім CGI-програма робить свою чарівну справу і пересилає вихідні дані назад серверу, який переправляє їх клієнтові.

Коли CGI-програма викликається за допомогою форми – найбільш поширеного інтерфейсу, браузер передає серверу довгий рядок, на початку якого стоїть шлях до CGI-програмі і її ім'я. Потім слідують різні інші дані, які називаються інформацією шляху і передаються CGI-програмі через змінну оточення PATH_INFO. Після інформації шляху слідує символ «?», а за ним – дані форми, які посилаються серверу за допомогою методу HTTP GET. Ці дані стають доступними CGI-програмі через змінну оточення QUERY_STRING. Будь-які дані, які сторінка посилає з використанням методу HTTP POST.

III. Передача параметрів серверу та запам’ятовування стану.

Робота по протоколу HTTP відбувається таким чином: програма-клієнт встановлює TCP-з’єднання з сервером (стандартний номер порту-80) і видає йому HTTP-запит. Сервер обробляє цей запит і видає HTTP-відповідь клієнтові.

HTTP-запит складається із заголовка запиту і тіла запиту, розділених порожнім рядком. Тіло запиту може бути відсутнім. Заголовок запиту складається з головного (першою) рядка запиту і подальших рядків, що уточнюють запит в головному рядку. Подальші рядки також можуть бути відсутніми. Запит в головному рядку складається з трьох частин, розділених пропусками:

1. Метод (інакше кажучи, команда HTTP):

- *GET* – запит документа. Метод, що найбільш часто вживається; у HTTP/0.9 він був єдиним;
- *HEAD* – запит заголовку документа. Відрізняється від GET тим, що видається лише заголовок запиту з інформацією про документ. Сам документ не видається;
- *POST* – цей метод застосовується для передачі даних CGI-скриптам. Самі дані слідують в подальших рядках запиту у вигляді параметрів;
- *PUT* – розмістити документ на сервері. Використовується рідко. Запит з цим методом має тіло, в якому передається сам документ.

2. Ресурс – це шлях до певного файлу на сервері, який клієнт хоче отримати (або розмістити – для методу PUT). Якщо ресурс – просто який-небудь файл для прочитування, сервер повинен по цьому запиту видати його в тілі відповіді. Якщо ж це шлях до якого-небудь CGI-скрипту, то сервер запускає скрипт і повертає результат його виконання. До речі, завдяки такій уніфікації ресурсів для клієнта практично байдуже, що він є на сервері.

3. Версія протоколу – версія протоколу HTTP, з якою працює клієнтська програма.

Таким чином, простий HTTP-запит може виглядати таким чином:

GET / HTTP/1.0 – запрошується кореневий файл з кореневої директорії web-сервера.

Рядки після головного рядка запиту мають наступний формат: *Параметр: значення*.

Таким чином задаються параметри запиту. Рядки після головного рядка запиту можуть бути відсутніми, в цьому випадку сервер набуває їх значення за замовчуванням або за результатами запиту (при роботі в режимі Keep-Alive).

Перерахуємо декілька найбільш вживаних параметрів HTTP-запиту:

Connection – може приймати значення Keep-Alive і close:

Keep-Alive означає, що після видачі даного документа з'єднання з сервером не розривається, і можна видавати ще запити. Більшість браузерів працюють саме в режимі Keep-Alive, оскільки він дозволяє за одне з'єднання з сервером «викачати» HTML-сторінку і малюнки до неї. Будучи одного дня встановленим, режим Keep-Alive зберігається до першої помилки або до явної вказівки в черговому запиті *Connection: close*.

Close – з'єднання закривається після відповіді на даний запит.

– User-Agent – значенням є «кодове позначення» браузеру.

– Accept – список підтримуваних браузером типів вмісту в порядку їх переваги даним браузером, наприклад: *Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint,* */*. Значення цього параметра використовується в основному CGI-скриптами для формування відповіді.

– Referer – URL, з якого перейшли на цей ресурс.

– Host – ім'я хоста, з якого запрошується ресурс. Корисно, якщо на сервері є декілька віртуальних серверів під одною IP-адресою. В цьому випадку ім'я віртуального сервера визначається по цьому полю.

– Accept-Language – підтримувана мова. Має значення для сервера, який може видавати один і той же документ в різних мовних версіях.

Формат відповіді дуже схожий на формат запиту: він також має заголовок і тіло, розділене порожнім рядком. Заголовок також складається з основного рядка і рядків параметрів, але формат основного рядка відрізняється від такого ж в заголовку запиту. Основний рядок запиту складається з 3-х полів, розділених пропусками:

– *Версія протоколу* – аналогічний відповідному параметру запиту.

– *Код помилки* – кодове позначення «успішності» виконання запиту. Код 200 означає «все нормально» (*OK*).

– *Словесний опис помилки* – «розшифровка» попередньої коду. Наприклад, для 200 це *OK*, для 500 – *Internal Server Error*.

Найбільш вживані параметри HTTP-відповіді:

– *Connection* – аналогічний відповідному параметру запиту. Якщо сервер не підтримує Keep-Alive (є і такі), то значення Connection відповідає завжди close.

– *Content-Type* – містить позначення типу вмісту відповіді.

Залежно від значення Content-Type браузер сприймає відповідь як HTML-сторінку, картинку gif або jpeg, як файл, який треба зберегти на диску, або що-небудь ще і робить відповідні дії. Значення Content-Type для браузеру аналогічно значенню розширення файлу для Windows.

IV. Міри безпеки.

Будь-яка використовувана нами Інтернет-технологія потребує застосування відповідних механізмів захисту, що враховують її специфічні вразливості та потенційні ризики. За умови системного підходу до організації безпеки, коли питання захисту враховуються вже на етапі проєктування програмного продукту, вебсистеми набувають вищого рівня надійності й стабільності. Такий підхід дає змогу уникнути необхідності оперативного усунення вразливостей у процесі експлуатації та постійного пошуку додаткових способів захисту програмного рішення.

Проблема безпеки залишається однією з ключових для більшості вебсайтів, а також для вебпредставництв великих компаній, значна частина яких і сьогодні не має достатнього рівня захисту від сучасних кіберзагроз.

Міри безпеки веб-сайту можна розділити на дві категорії:

– *безпека системи* (гарантія того, що інші люди не можуть модифікувати веб-сайт);
– *інформаційна безпека* (забезпечить безпеку інформації, наприклад власних даних покупця в Інтернет-магазині).

Система безпеки має значення для підтвердження того, що комп'ютерна система в безпеці, і для зменшення проникнення зловмисників на веб-сервер з метою зміни сторінок.

Деякі веб-сайти зберігають особливу інформацію, таку як власні дані користувача (наприклад номер кредитної карточки користувача). Тому для початку необхідно проаналізувати всю зберігаючу інформацію і визначити які дані повинні бути особливо захищені.

Ще одним важливим аспектом в безпеці системи являється програмне забезпечення яке представляє систему. Програмне забезпечення може мати помилки, і такі вразливі місця, які забезпечують доступ до системи без пароля. Програмне забезпечення повинно бути оновленим з допомогою патчів і оновлень.

Веб-сервери являють собою набір сучасних програм, які містять загальні помилки і «дірки» в безпеці, які можуть бути виявлені зловмисниками, і надати доступ зловмисникам до вашої системи навіть якщо вони не знають пароля для доступу.

Проблеми безпеки веб-сайтів найчастіше виникають зі сторони веб-серверів, на яких вони працюють. Деколи ці проблеми відносно незначні і дозволяють зловмиснику зробити ваш сервер неактивним до тих пір поки ви не вирішите проблему. Тому очевидно що всі вразливі місця в безпеці повинні бути вирішені, так як вони можуть надати зловмиснику доступ до файлів

вашого сервера. Всі програми, які задіяні в процес організації доступу і управління сервером, можуть впливати в тому чи іншому випадку на безпеку всієї вашої системи.

Програмне забезпечення (особливо сервер) потрібно налаштувати таким чином, щоб всі необов'язкові функції були відключені. Деяке програмне забезпечення при встановленні на персональний комп'ютер чи ноутбук підключає багато непотрібних функцій. Це означає, якщо знайдеться вразливе місце в будь-якій із цих опцій, ваша система може бути вразлива навіть якщо ви ці функції не використовуєте.

Програми Firewall контролюють і фільтрують доступ до вашого сервера. Виключення складають ті пакети які підходять під визначений набір параметрів. Також необхідно врахувати що програми Firewall можна використовувати для уникнення ризиків в цілях власної безпеки, хоча це не є певним основним й загальним вирішенням, так як існує ймовірність атак, які можуть виникнути через ваш сервер або через інші порти, доступ до яких дозволений.

Після того як вирішили, яку інформацію потрібно захистити, необхідно приділити увагу тому, яким чином інформація може бути перехоплена, і прийняти міри щоб знизити ризик.

В якості прикладу можна взяти номери кредитних карточок із Інтернет-магазину. Перший крок повинен бути направлений на збереження системи безпеки як можна на більш високому рівні. Переконайтесь що все програмне забезпечення оновляється і що воно надійне (наприклад якщо ви використовуєте програму on-line для онлайн-покупок з «корзиною покупця», переконайтесь що немає вразливих місць в цій частині вашого віртуального магазину, і що це було перевірено експертами по безпеці). Корзина покупця в більшості випадків являється дуже великою вразливістю в безпеці.

У разі, коли номери кредитних карт пересилаються, це повинно проходити в повній безпеці. Наприклад, веб-сайт повинен реалізовувати захищене з'єднання тоді, коли користувач вводить номер. Деякі веб-сайти відправляють інформацію заказів незашифрованою поштою або простим текстом. Звичайно це збільшує ризик перехоплення.

Потрібно відмітити, що не має ні одного абсолютно безпечного сайту, але ціль повинна бути така щоб привести до мінімуму ймовірність хакерських атак.

ЛЕКЦІЯ 15

Тема: *Доступ до баз даних. СУБД MYSQL.*

План:

1. Доступ до баз даних.
2. СУБД MySQL. Система безпеки.
3. Утиліти.
4. Мова SQL.

І. Доступ до баз даних.

Однією з найважливіших можливостей PHP є організація доступу до баз даних, що дозволяє створювати динамічні вебзастосунки, у яких інформація не лише відображається, а й зберігається, змінюється та аналізується. Саме взаємодія з системами керування базами даних забезпечує функціонування сучасних інформаційних систем: електронних журналів, навчальних платформ, інтернет-магазинів, корпоративних порталів та інших вебресурсів.

База даних – це сукупність зв'язаних даних, організованих по певних правилах, що передбачають загальні принципи опису, зберігання і маніпулювання, незалежна від прикладних програм.

У PHP база даних використовується для:

- збереження інформації про користувачів;
- обробки результатів запитів;
- ведення облікових записів;
- організації динамічного наповнення вебсторінок.

Найчастіше PHP працює із MySQL, хоча підтримуються також PostgreSQL, SQLite, Oracle та інші системи.

В загальному випадку база даних містить схеми, таблиці, подання, збережені процедури та інші об'єкти. Дані у базі організовують відповідно до моделі організації даних. Таким чином, сучасна база даних, крім саме даних, містить їх опис та може містити засоби для їх обробки.

Бази даних класифікують за різними критеріями. За моделлю організації даних розрізняють такі бази даних:

- **Ієрархічна.** Ієрархічна база даних може бути представлена як дерево, що складається з об'єктів різних рівнів. Між об'єктами існують зв'язки типу «предок-нащадок». При цьому можлива ситуація, коли об'єкт не має нащадків або має їх декілька, тоді як у об'єкта-нащадка обов'язково тільки один предок.
- **Мережна.** Така база даних подібна до ієрархічної, за винятком того, що кожен об'єкт може мати більше одного предка.

- **Реляційна.** Реляційна база даних зберігає дані у вигляді таблиць. Найвживаніші СУБД використовують реляційну модель даних.
- **Об'єктно-орієнтована.** У базі даних цього виду дані оформляють у вигляді моделей об'єктів.

Звернення до баз даних здійснюється за допомогою системи управління базами даних (СУБД). СУБД забезпечує підтримку створення баз даних, централізованого управління і організації доступу до них різних користувачів.

Система управління базами даних – це система, заснована на програмних та технічних засобах, яка забезпечує визначення, створення, маніпулювання, контроль, керування та використання баз даних.

Застосунки для роботи з базою даних можуть бути частиною СУБД або автономними. Найпопулярнішими СУБД є MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Sybase, Interbase, Firebird та IBM DB2.

Сучасні СУБД забезпечують функції щодо керування даними, які можна поділити на такі групи:

- *оголошення даних* – створення, зміна та видалення визначень, які описують організацію даних;
- *модифікація даних* – додавання даних, їх редагування та видалення.
- *отримання даних* – надання даних за запитом застосунку у формі, яка дозволяє їх безпосереднє використання. Дані можуть надаватись або у формі, в якій вони зберігаються у базі даних, або в іншій формі (наприклад, через поєднання різних даних).
- *адміністрування даних* – реєстрування та відслідковування дій користувачів, дотримання безпеки роботи з даними, забезпечення надійності та цілісності даних, моніторинг продуктивності, резервне копіювання та відновлення даних тощо.

У процесі роботи з базами даних використовують мови спеціального призначення:

- *мова визначення даних (Data definition language, DDL)* – це мова, яка описує дані та структури даних, а також визначає взаємозв'язки між ними;
- *мова маніпулювання даними (Data manipulation language, DML)* – це мова, яку підтримує СКБД і яка забезпечує виконання операцій отримання, додавання, зміни та видалення даних;
- *мова запитів (Query language)* – це мова для користувачів, яка забезпечує отримання та оброблення даних у базі даних.

ІІ. СУБД MySQL. Система безпеки.

MYSQL – це реляційна система управління базами даних. Тобто дані в її базах зберігаються у вигляді логічно зв'язаних між собою таблиць, доступ до яких здійснюється за допомогою мови запитів SQL. MYSQL – вільно поширювана система, тобто платити за її використання не потрібно. Крім того, це досить швидка, надійна і, головне, проста у використанні СУБД, цілком відповідна для не дуже глобальних проектів.

MYSQL є, можливо, найяскравішим програмним проектом після виходу Linux. Зараз вона серйозний конкурент великим СУБД в області розробки баз даних. Особливими цілями проектування MYSQL були швидкість, надійність і простота використання. Аби досягти такої продуктивності, її розробник – шведська фірма TCX прийняла рішення зробити багатопотоковим внутрішній механізм MYSQL. Багатопотокове застосування передбачає одночасне виконання декількох завдань – подібно одночасному запуску кількох екземплярів застосування.

MySQL виникла як спроба застосувати mSQL до власних розробок компанії: таблиць, для яких використовувалися ISAM – підпрограми низького рівня для індексного доступу до даних. У результаті був вироблений новий SQL-інтерфейс, але API-інтерфейс залишився в спадок від mSQL.

Працювати з MYSQL можна не тільки в текстовому режимі, але і в графічному. Існує дуже популярний візуальний інтерфейс (до речі, написаний на PHP) для роботи з цією СУБД. Називається він PhpMyAdmin.

Одне з основних завдань, що виникають при роботі з базами даних, – це завдання пошуку. При цьому, оскільки інформації в базі даних, як правило, міститься багато, перед програмістами постає завдання не просто пошуку, а ефективного пошуку, тобто пошуку за порівняно невеликий час і з достатньою точністю. Для цього проводять індексування деяких полів таблиці. Використовувати індекси корисно для швидкого пошуку рядків з вказаним значенням одного стовпця.

У MYSQL існує три види індексів: *PRIMARY, UNIQUE* і *INDEX*, а слово ключ (KEY) використовується як синонім слова індекс (INDEX). Всі індекси зберігаються в пам'яті у вигляді B-дерев.

PRIMARY – унікальний індекс (ключ) з обмеженням, що всі індексовані ним поля не можуть мати порожнього значення (тобто вони *NOT NULL*). Таблиця може мати тільки один первинний індекс, але він може складатися з декількох полів.

Зовнішні ключі використовуються для організації зв’язків між таблицями бази даних (батьківськими і дочірними) та для підтримки обмежень посилальної цілісності даних.

UNIQUE – ключ (індекс), задаючий поля, які можуть мати тільки унікальні значення.

INDEX – звичайний індекс.

У MYSQL кожне вхідне з’єднання обробляється окремим потоком, при цьому ще один потік, що завжди виконується, керує з’єднаннями, тому клієнтам не доводиться чекати завершення виконання запитів інших клієнтів. Одночасно може виконуватися будь-яка кількість запитів. Поки який-небудь потік записує дані в таблицю, всі інші запити, що вимагають доступу до цієї таблиці, просто чекають, поки вона звільниться. Клієнт може виконувати всі допустимі операції, не звертає уваги на інші одночасні з’єднання. Керуючий потік запобігає одночасному запису якими-небудь двома потоками в одну і ту ж таблицю. Така архітектура складніша, ніж однопоточна. Проте виграш в швидкості завдяки одночасному виконанню декількох запитів значно перевершує втрати швидкості, викликані збільшенням складності.

У процесі роботи з базою даних потрібно мати не лише надійний доступ до своїх даних, але і бути упевненим, що в інших немає жодного доступу до них. MYSQL використовує власний сервер баз даних для забезпечення безпеки.

Можливості сервера MySQL:

- простота у встановленні та використанні;
- підтримується необмежена кількість користувачів, що одночасно працюють із БД;
- кількість рядків у таблицях може досягати 50 млн;
- висока швидкість виконання команд;
- наявність простої і ефективної системи безпеки.

При первинній установці MYSQL створюється база даних під назвою «mysql». У цій базі є п’ять таблиць: db, host, user, tables_priv, і columns_priv. Новіші версії MYSQL створюють також базу даних з назвою func, але вона не має відношення до безпеки. MYSQL використовує ці таблиці для визначення того, кому що дозволено робити. Таблиця user містить дані по безпеці, що відносяться до сервера в цілому. Таблиця host містить права доступу до сервера для віддалених комп’ютерів. І нарешті, db, tables_priv і columns_priv керують доступом до окремих баз даних, таблиць і колонок.

У MYSQL є спеціальна функція, що дозволяє приховати паролі від цікавих очей. Функція password() зашифровує пароль.

Імена користувачів MYSQL зазвичай не пов’язані з іменами користувачів операційної системи. За замовчуванням клієнтські засоби MYSQL використовують при реєстрації імена користувачів операційної системи, проте, обов’язкової відповідності не вимагається. У більшості клієнтських застосувань MYSQL можна за допомогою параметра -u підключитися до MYSQL, використовуючи будь-яке ім’я.

При підключенні до бази даних проводяться дві перевірки. Спочатку MYSQL перевіряє, чи є в таблиці user запис, відповідний імені користувача і машини, до якої він підключається. Якщо відповідність не знайдена, в доступі відмовляється. У разі, коли відповідний запис знайдений і має непорожнє поле Password, необхідно ввести правильний пароль. Неправильний пароль призводить до відхилення запиту на підключення.

Якщо з’єднання встановлене, MYSQL переходить до етапу верифікації запиту. При цьому зроблені вами запити зіставляються з вашими правами. Ці права MYSQL перевіряє по таблицях user, db, host, tables_pnv і columns_priv.

Якщо таблиця db містить необхідний дозвіл, подальша перевірка припиняється і виконується команда. Якщо його немає, то MYSQL шукає відповідність в таблиці tables_priv. Якщо це команда SELECT, об’єднуюча дві таблиці, то користувач повинен мати дозвіл для обох цих таблиць. Якщо хоч би один із записів відмовляє в доступі або відсутній, MYSQL таким самим способом перевіряє всі колонки в таблиці columns_priv.

MYSQL завантажує таблиці доступу при запуску сервера. Перевагою такого підходу в порівнянні з динамічним зверненням до таблиць є швидкість. Негативна сторона полягає в тому, що зміни, зроблені в таблицях доступу MYSQL, не відразу починають діяти.

III. Утиліти.

MYSQL поширюється з великим набором допоміжних утиліт, серед яких розрізняють:

1. Утиліти командного рядка (Command Line Tools):

Isamchk – проводить перевірку файлів, що містяться в дані бази. Ці файли називаються ISAM-файлами (ISAM – метод індексованого послідовного доступу). Ця утиліта може усунути велику частину пошкоджень ISAM-файлів.

Isamlog – читає створювані журнали MYSQL, що відносяться до ISAM-файлів. Ці журнали можна використовувати для відтворення таблиць або відтворення змін, внесених до таблиць протягом деякого проміжку часу.

mysql – створює пряме підключення до сервера баз даних і дозволяє вводити запити безпосередньо із запрошення MYSQL.

mysqlaccess – модифікує таблиці прав доступу MYSQL і відображує їх в зручному для читання вигляді. Використання цієї утиліти – хороший спосіб вивчення структури таблиць доступу MYSQL.

Mysqladmin – здійснює адміністративні функції. За допомогою цієї утиліти можна додавати і видаляти цілі бази даних, а також завершувати роботу сервера.

Mysqldump – записує весь вміст таблиці, включаючи її структуру, у файл у вигляді SQL-команд, якими можна відтворити таблицю. Вихідні дані цієї утиліти можна використовувати для відтворення таблиці в іншій базі або на іншому сервері. Синтаксис її вживання: *mysqldump -u user -p dbname --tab=path, де path* – шлях для збереження файлів.

Mysqlimport – прочитує дані з файлу і вводить їх в таблицю бази даних. Це має бути файл з роздільниками, де роздільник може бути будь-якого звичайного вигляду, наприклад, кома або лапки.

Mysqlshow – виводить на екран структуру баз даних, що є на сервері, і таблиці, з яких вони складаються.

2. Утиліти сторонніх розробників. Жоден постачальник або розробник не може самостійно надати всі необхідні для програмного продукту засоби підтримки. За найсвіжішим списком потрібно звернутись на домашню сторінку MYSQL: *http://www.mysql.com/Contrib*.

3. Утиліти перетворення баз даних:

access_to_mysql – перетворить бази даних Microsoft Access в таблиці MYSQL. Включається в Access у вигляді функції, що дозволяє зберігати таблиці у форматі, що дозволяє експортувати їх в MYSQL.

dbf2mysql – конвертує файли dBASE (DBF) в таблиці MYSQL.

Exportsql/Importsql – конвертує бази даних Microsoft Access в MYSQL і назад. Ці утиліти є функціями Access, які можна використовувати для експорту таблиць Access у форматі, придатному для читання MYSQL. З їх допомогою можна також перетворювати SQL-вихід MYSQL у вигляд, придатний для читання Access.

4. Інтерфейси CGI:

PHP – створює HTML-сторінки з використанням спеціальних тегів, розпізнаваних аналізатором PHP. PHP має інтерфейси до більшості основних баз даних, включаючи MYSQL і mSQL.

Mysql-webadmin – здійснює веб-сервер-адміністрування баз даних MYSQL. Використовуючи цей засіб, можна переглядати таблиці і змінювати їх вміст за допомогою HTML-форм.

Mysqladm – здійснює веб-сервер-адміністрування баз даних MYSQL. Ця CGI-програма дозволяє переглядати таблиці через WWW, добавлять таблиці і змінювати їх вміст.

www-sql – створює HTML-сторінки з таблиць баз даних MYSQL. Ця програма здійснює розбір HTML-сторінок у пошуках спеціальних тегів і використовує дані, що витягують, для виконання команд SQL на сервері.

5. Клієнтські застосування:

Mysqlwinadmn – дозволяє адмініструвати MYSQL з Windows. За допомогою цього засобу можна виконувати функції mysqladmin з графічного інтерфейсу.

Xmysql – забезпечує повний доступ до таблиць баз даних MYSQL для клієнта X Window System. Підтримує групові вставки і видалення.

Xmysqladmin – дозволяє здійснювати адміністрування MYSQL з X Window System. Це інструмент для графічного інтерфейсу, що дозволяє створювати і видаляти бази даних і управляти таблицями.

6. Інтерфейси програмування:

MYODBC – реалізує ODBC API до MYSQL в Windows.

mm.mysql.jdbc – реалізує стандартний API JDBC (Java Database Connectivity – доступ до баз даних з Java).

TwzJdbcForMysql – реалізація JDBC API для Java.

IV. Мова SQL.

Для читання і запису в базах даних MYSQL використовується структурована мова запитів (SQL). Використовуючи SQL, можна здійснювати пошук, вводити нові дані або видаляти дані. SQL є просто інструментом, необхідним для взаємодії з MYSQL. Навіть якщо для доступу до бази даних користуєтеся якимсь застосуванням або графічним інтерфейсом користувача – це застосування всеодно генерує SQL-команди.

У IBM винайшли SQL на початку 1970-х, незабаром після введення Е. Коддом поняття реляційної бази даних. Із самого початку SQL була легкою у вивченні, але потужною мовою. Вона нагадує природну мову, таку як англійська і тому не стомлює тих, хто не є технічним фахівцем.

SQL дійсно була настільки популярною серед користувачів, для яких призначалася, що в 1980-х компанія Oracle випустила першу в світі загальнодоступну комерційну SQL-систему. Oracle SQL була надзвичайно популярною і породила довкола SQL цілу індустрію. Sybase, Informix, Microsoft і ряд інших компаній вийшли на ринок з власними розробками реляційних систем управління базами даних (РСУБД), заснованих на SQL.

У той час коли Oracle і її конкуренти вийшли на сцену, SQL була новинкою, і для неї не існувало стандартів. Лише у 1989 році комісія із стандартів ANSI випустила перший загальнодоступний стандарт SQL. Сьогодні його називають SQL89. Цей новий стандарт не дуже заглиблювався у визначення технічної структури мови. Тому, лише у 1992 році стандарт ANSI SQL вступив в свої права.

Стандарт 1992 року позначають як SQL92 або SQL2. Стандарт SQL2 включив максимально можливу кількість розширень, доданих в комерційних реалізаціях мови. SQL2 – не останнє слово в стандартах SQL. У зв'язку із зростанням популярності об'єктно-орієнтованих СУБД (ООСУБД) і об'єктно-реляційних СУБД (ОРСУБД) постійно зростають можливості мови з метою оптимізації її до об'єктно-орієнтованого доступу до баз даних.

З появою MYSQL з'явився новий підхід до розробки серверів баз даних. Замість створення чергової гігантської СУБД з ризиком не запропонувати нічого нового, були запропоновані невеликі і швидкі реалізації найбільш часто використовуваних функцій SQL.

Функції мови SQL будь-якої СУБД включають:

1. створення, видалення, зміна бази даних (БД);
2. додавання, зміна, видалення, призначення прав користувача;
3. внесення, видалення і зміна даних в БД (таблиць і записів);
4. вибірку даних з БД.

Перш ніж що-небудь робити з даними, потрібно створити таблиці, в яких ці дані зберігатимуться, навчитися змінювати структуру цих таблиць і видаляти їх, якщо буде потрібно. Для цього в мові SQL існують оператори CREATE TABLE, ALTER TABLE і DROP TABLE.

Оператор CREATE TABLE створює таблицю із заданим ім'ям в поточній базі даних. Якщо немає активної поточної бази даних або вказана таблиця вже існує, то виникає помилка виконання команди.

Кожна таблиця представлена набором певних файлів в директорії бази даних.

Синтаксис:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]
ім'я_таблиці
  [(означення_стовпця), ... ]
  [опції_таблиці]
  [select_вираз]
```

У виразі *означення_стовпця* перераховують, які стовпці повинні бути створені в таблиці. Кожен стовпець таблиці може бути порожнім (NULL), мати значення за замовчуванням, бути ключем або автоінкрементом. Крім того, для

кожного стовпця обов'язково вказується тип даних, який в ньому зберігатиметься. Якщо не вказується ні NULL, ні NOT NULL, то стовпець інтерпретується так, як ніби вказано NULL. Якщо поле позначають як автоінкремент (AUTO_INCREMENT), то його значення автоматично збільшується на одиницю кожного разу, коли відбувається додавання даних в таблицю і в це поле записується порожнє значення (NULL, тобто нічого не записується) або 0. Автоінкремент в таблиці може бути тільки один, і при цьому він обов'язково повинен бути проіндексований. Послідовність AUTO_INCREMENT починається з 1. Формальний опис стовпця (означення_стовпця) виглядає так:

```
ім'я_стовпця тип [NOT NULL|NULL] [DEFAULT
значення_за_замовчуванням] [AUTO_INCREMENT] [PRIMARY KEY]
[reference_definition]
```

Тип стовпця може бути одним з наступних:

- цілий: INT[(length)] [UNSIGNED] [ZEROFILL];
- дійсний: REAL[(length,decimals)] [UNSIGNED] [ZEROFILL];
- символьний: CHAR(length) [BINARY] або VARCHAR(length) [BINARY];
- дата і час: DATE або TIME;
- для роботи з великими об'єктами: BLOB;
- текстовий: TEXT;
- множина: ENUM(value1, value2, value3 ...) або SET(value1, value2 ...).

Останній (опційний) елемент команди CREATE – це вираз SELECT (select_вираз).

Синтаксис:

```
[IGNORE | REPLACE] SELECT (будь-який вираз SELECT)
```

Якщо при створенні таблиці в команді CREATE вказується вираз SELECT, то всі поля, отримані вибіркою, додаються в створювану таблицю.

За допомогою специфічної для MySql команди SHOW можна проглянути існуючі бази даних, таблиці в базі даних і поля в таблиці.

Показати всі бази даних: `mysql>SHOW databases;`

Зробити поточною базу даних book і показати всі таблиці в ній:

```
mysql>use book; mysql>show tables;
```

Показати всі стовпці в таблиці Persons:

```
mysql> show columns from Persons;
```

Оператор DROP TABLE видаляє одну або декілька таблиць. Всі табличні дані і визначення видаляються, тому при роботі з цією командою слід дотримуватися обережності.

Синтаксис:

```
DROP TABLE [IF EXISTS] ім'я_таблиці [, ім'я_таблиці...]
                    [RESTRICT|CASCADE]
```

Опції RESTRICT і CASCADE дозволяють спростити перенесення програми з інших СУБД. В даний момент вони не задіяні.

Оператор ALTER TABLE забезпечує можливість змінювати структуру існуючої таблиці. Наприклад, можна додавати або видаляти стовпці, створювати або знищувати індекси, перейменовувати стовпці або саму таблицю. Можна також змінювати коментар таблиці і її тип.

Синтаксис:

```
ALTER [IGNORE] TABLE ім'я_таблиці alter_specification [,
                  alter_specification ...]
```

Оператор ALTER TABLE під час роботи створює тимчасову копію початкової таблиці. Необхідна зміна виконується на копії, потім початкова таблиця віддаляється, а нова перейменовується. Це робиться для того, щоб в нову таблицю автоматично потрапляли всі оновлення, окрім невдалих.

Оператор SELECT застосовується для витягування рядків, вибраних з однієї або декількох таблиць. Тобто з його допомогою ми задаємо стовпці або вирази, які треба витягувати (select_вираз), таблиці (table_references), з яких повинна проводитися вибірка, і, можлива умова (where_definition), якій повинні відповідати дані в цих стовпцях, і порядок, в якому ці дані потрібно видати.

Спрощено структуру оператора SELECT можна представити таким чином:

```
SELECT select_вираз1, select_вираз2 ...
  [FROM table_references
    [WHERE where_definition]
    [ORDER BY {число | ім'я_стовпця | формула [ASC
| DESC]} ...]]
```

Квадратні дужки [ ] означають, що використання оператора, що знаходиться в них, необов'язкове, вертикальна межа | означає перерахування можливих варіантів. Після ключового слова ORDER BY указують ім'я стовпця, число (ціле беззнакове) або формулу і спосіб впорядкування (за збільшенням – ASC, або зменшення – DESC). За замовчуванням використовується впорядкування за збільшенням. Коли в select_виразі ми пишемо «*», то це означає, що вибрати всі стовпці. Окрім «*» в select-виразі можуть використовуватися функції типу max, min і avg.

Оператор INSERT вставляє нові рядки в існуючу таблицю. Оператора має декілька форм. Параметр ім'я_таблиці у всіх цих формах задає таблицю, в

яку повинні бути внесені рядки. Стовпці, для яких задаються значення, вказуються в списку імен стовпців (ім'я_стовпця) або в частині SET.

Синтаксис:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE] [INTO] ім'я_таблиці
[(ім'я_стовпця ...)] VALUES (вираз ...) (...) ...
```

Оператор UPDATE оновлює значення існуючих стовпців таблиці відповідно до введених значень.

Синтаксис:

```
UPDATE [LOW_PRIORITY] [IGNORE] ім'я_таблиці SET
ім'я_стовпця1 = вираз1 [, ім'я_стовпця2 = вираз2 ...]
[WHERE where_definition] [LIMIT число]
```

У виразі SET вказується, які саме стовпці слід модифікувати і які величини повинні бути в них встановлені. У виразі WHERE, якщо він присутній, задається, які рядки підлягають оновленню. У решті випадків оновлюються всі рядки. Якщо заданий вираз ORDER BY, то рядки оновлюватимуться у вказаному в нім порядку.

Якщо вказується ключове слово LOW_PRIORITY, то виконання даної команди UPDATE затримується до тих пір, поки інші клієнти не завершать читання цієї таблиці. Якщо вказується ключове слово IGNORE, то команда оновлення не буде перервана, навіть якщо виникне помилка дублювання ключів. Рядки, із-за яких виникають конфліктні ситуації, оновлені не будуть.

Якщо у виразі, який задає нове значення стовпця, використовується ім'я цього поля, то команда UPDATE використовує для цього стовпця його поточне значення. Наприклад, наступна команда встановлює стовпець death_date в значення, на одиницю більше його поточної величини:

```
mysql> UPDATE Persons SET death_date = death_date + 1;
```

Оператор DELETE видаляє з таблиці `ім'я_таблиці` рядки, що задовольняють задані в `where_definition` умови, і повертає число видалених записів.

Якщо оператор DELETE запускається без визначення WHERE, то віддаляються всі рядки. Синтаксис:

```
DELETE [LOW_PRIORITY] FROM ім'я_таблиці [WHERE
where_definition] [LIMIT rows]
```

Специфічна для MYSQL опція LIMIT для команди DELETE вказує серверу максимальну кількість рядків, які слід видалити до повернення управління клієнтові. Ця опція може використовуватися для гарантії того, що дана команда DELETE не зажадає дуже багато часу для виконання.

ЛЕКЦІЯ 16

Тема: *Функції та масиви в PHP*.

План:

1. Функції в PHP.
2. Робота з масивами даних.

І. Функції в PHP.

У PHP функція – це іменований блок програмного коду, призначений для багаторазового виконання певної послідовності дій. Використання функцій дає змогу структурувати програму, уникати повторення однакових фрагментів коду, спрощувати супровід програмних систем і підвищувати читабельність програмного тексту. Функціональний підхід є одним із базових принципів побудови сучасних вебзастосунків.

Функції використовуються для:

– виконання типових обчислень;

– обробки даних;

– формування окремих логічних блоків програми;

– повторного використання коду.

У програмуванні, як і в математиці, функція є відображення безлічі її аргументів на безліч значень. Функція може бути визначена за допомогою наступного синтаксису:

```
function Ім’я_функції (параметр1, параметр2 ... параметрN)
{ Блок_дій…  return "Значення, повернене функцією"; }
```

Ім’я_функції і *імена параметрів* функції (*параметр 1*, *параметр 2* і так далі) повинні відповідати правилам найменування в PHP. Імена функцій нечутливі до регістра. Параметри функції – це змінні мови, тому перед назвою кожного з них повинен стояти знак $. Замість слів *блок_дій* в тілі функції повинен знаходитися будь-який правильний PHP-код (не обов’язково залежний від параметрів). Після ключового слова *return* повинен йти коректний php-вираз. Загалом, у функції може і не бути параметрів та значення, яке вона повертає.

Для виклику функції вказується ім’я функції і в круглих дужках список значень її параметрів, якщо такі є:

```
<?php
  Ім’я_функції("значення_для_параметра1",
"значення_для_параметра2" ...);
?>
```

Не дивлячись на те, що імена функцій нечутливі до регістру, краще викликати функцію по тому ж імені, яким вона була задана у визначенні.

У кожної функції може бути список аргументів. За допомогою цих аргументів у функцію передається різна інформація. Кожен аргумент є змінною або константою.

За допомогою аргументів дані у функцію можна передавати трьома різними способами – це передача аргументів за значенням (використовується за замовчуванням), за посиланням та заданням значення аргументів за замовчуванням. Розглянемо ці способи докладніше.

Коли аргумент передається у функцію за значенням, зміна значення аргумента усередині функції не впливає на його значення поза функцією. Щоб дозволити функції змінювати її аргументи, їх потрібно передавати за посиланням. Для цього у визначенні функції перед ім'ям аргумента слід написати знак амперсанд «&».

```
<?php
/* напишемо функцію, яка б додавала до рядка слово checked */
function add_label(&$data_str)
{
$data_str .= "checked";}
$str = "<input type=radio name=article "; /* нехай є такий
рядок */
echo $str .="><br>"; /* виведе елемент форми – не відмічену
радіо кнопку */
add_label($str);  // викличемо функцію
echo $str."><br>"; // виведе вже відмічену радіо кнопку
?>
```

У функції можна визначати значення аргументів, використовувані за замовчуванням. Саме значення повинне бути константним виразом, а не змінною і не представником класу або викликом іншої функції.

Якщо у функції декілька параметрів, то ті аргументи, для яких задаються значення за замовчуванням, повинні бути записані після решти всіх аргументів у визначенні функції. Інакше з'явиться помилка, якщо ці аргументи будуть опущені при виклику функції.

Наприклад, ми хочемо внести опис статті до каталогу. Користувач повинен ввести такі характеристики статті, як її назва, автор і короткий опис. Якщо користувач не вводить ім'я автора статті, вважаємо, що це Шевченко Тарас.

```
<?php
function Add_article($title, $description, $author = "Шевченко
Тарас")
```

```
  {
    echo "Заносимо в каталог статтю: $title,";
    echo "автор $author";
    echo "<br>Короткий опис: ";
    echo "$description <hr>";
  }
  Add_article("Інформатика і ми", "Це стаття про інформатику
...", "Петренко Петро");
  Add_article("Хто такі хакери", "Це стаття про хакерів ...");
?>
```

Результат роботи скрипта:

```
Заносимо в каталог статтю: Інформатика і ми, автор Петренко Петро.
Короткий опис: Це стаття про інформатику...
```

```
Заносимо в каталог статтю: Хто такі хакери, автор Шевченко Тарас.
Короткий опис: Це стаття про хакерів...
```

У PHP можна створювати функції із змінним числом аргументів. Для написання такої функції ніякого спеціального синтаксису не потрібно. Все робиться за допомогою вбудованих функцій *func_num_args(), func_get_arg(), func_get_args().*

Функція func_num_args() повертає число аргументів, переданих в поточну функцію. Ця функція може використовуватися тільки усередині визначення, призначеного для користувача функції. Якщо вона з'явиться поза функцією, то інтерпретатор видасть попередження.

```
<?php
function DataCheck()
{
$n = func_num_args();
echo "Число аргументів функції $n";}
DataCheck(); /* виведе рядок "Число аргументів функції 0" */
DataCheck(1, 2, 3); /* виведе рядок "Число аргументів
функції 3" */
?>
```

Функція func_get_arg повертає аргумент із списку переданих у функцію аргументів, порядковий номер якого заданий параметром *номер_аргумента*. Аргументи функції нумеруються, починаючи з нуля. Як і *func_num_args()*, ця функція може використовуватися тільки усередині визначення якої-небудь функції.

Номер_аргумента не може перевищувати числа аргументів, переданих у функцію. Інакше згенерується попередження, і функція *func_get_arg()* поверне значення *False*.

Створимо функцію для перевірки типу даних її аргументів. Вважаємо, що перевірка пройшла успішно, якщо перший аргумент функції – ціле число, другий – рядок.

```
<?php
function DataCheck()
{
$check =true;  // число аргументів, переданих у функцію
$n = func_num_args(); /* перевіряємо, чи є перший переданий
аргумент цілим числом */
if ($n >= 1) if (!is_int(func_get_arg(0))) $check = false;
/* перевіряємо, чи є другий, переданий аргумент рядком */
if ($n >= 2) if (!is_string(func_get_arg(1))) $check = false;
return $check;
}
if (DataCheck(123, "text")) echo "Перевірка пройшла успішно
<br>";
else echo "Дані не задовольняють умови<br>";
if (DataCheck(324)) echo "Перевірка пройшла успішно<br>";
else echo "Дані не задовольняють умови<br>";
?>
```

Результатом роботи програми буде наступне:

```
Перевірка пройшла успішно
Перевірка пройшла успішно
```

Функція func_get_args() повертає масив, що складається із списку аргументів, переданих функції. Кожен елемент масиву відповідає аргументу, переданому функції. Якщо функція використовується поза визначенням призначеної для користувача функції, то генерується попередження.

Перепишемо попередній приклад, використовуючи дану функцію. Перевірятимемо, чи є цілим числом кожен парний аргумент, передаваний функції:

```
<?php
function DataCheck()
{
$check =true;
$n=func_num_args(); /*число аргументів переданих у функцію*/
$args = func_get_args(); /* масив аргументів функції */
for ($i=0; $i < $n; $i++)
{
```

```
$v = $args[$i];
if ($i % 2 == 0)
{ // перевіряємо, чи є парний аргумент цілим
if (!is_int($v)) $check = false;  }  }
return $check;
}
if(DataCheck(array("text",324)))echo "Перевірка пройшла
успішно <br>";
else echo "Дані не задовольняють умови<br>";
?>
```

Комбінації функцій *func_num_args()*, *func_get_arg()* і *func_get_args()* використовуються для того, щоб функції могли мати змінний список аргументів.

Щоб використовувати змінні тільки усередині функції, при цьому зберігаючи їх значення і після виходу з функції, потрібно оголосити ці змінні як *статичні*. Статичні змінні видно тільки усередині функції і не втрачають свого значення, якщо виконання програми виходить за межі функції. Оголошення таких змінних проводиться за допомогою ключового слова *static*:

```
static $var1, $var2;
```

Статичній змінній може бути привласнене будь-яке значення, але не посилання.

```
<?php
  function Test_s()
  {
    static $a = 1;
    $a = $a*2;
    echo $a;
  }
  Test_s(); // виведе 2
 echo $a; /* нічого не виведе, оскільки $a доступна
тільки усередині функції */
  Test_s(); /* усередині функції $a=2, тому
результатом роботи функції буде число 4 */
?>
```

Всі функції, приведені вище як приклади, виконували які-небудь дії. Окрім подібних дій, будь-яка функція може повертати як результат своєї роботи яке-небудь значення. Це робиться за допомогою *return*. Повертане значення може бути будь-якого типу, включаючи списки і об’єкти. Коли інтерпретатор зустрічає команду *return* в тілі функції, він негайно припиняє її виконання і переходить на той рядок, з якого була викликана функція.

Наприклад, складемо функцію, яка повертає вік людини. Якщо людина не померла, то вік вибирається щодо поточного року.

```
<?php
  /* якщо другий параметр обчислюється як true
     то він розглядається як дата смерті */
  function Age($birth, $is_dead){
    if ($is_dead) return $is_dead-$birth;
    else return date("Y")-$birth;
  }
  echo Age(1971, false);
  echo Age(1971, 2001);
?>
```

Коли функція повертає декілька значень для їх обробки в програмі, зручно використовувати мовну конструкцію *list()*, яка дозволяє однією дією привласнити значення відразу декільком змінним. Наприклад, в попередньому прикладі, залишивши без зміни функцію, обробити повернені нею значення можна було так:

```
<?php
  // завдання функції Full_age()
  list($day, $month,$year)= Full_age("07", "08", "2004");
  echo "Вам $year років $month місяців і $day днів";
?>
```

Взагалі конструкцію *list()* можна використовувати для привласнення змінним значень елементів будь-якого масиву.

```
<?php
  $arr = array("first", "second");
  list($a, $b)= $arr;
  // змінній $a привласнюється перше
  // значення масиву $b – друге
  echo $a",$b;
  // виведе рядок "first second"
?>
```

У результаті своєї роботи функція також може повертати посилання на яку-небудь змінну. Це може стати в нагоді, якщо потрібно використовувати функцію для того, щоб визначити, якій змінній повинне бути привласнена посилання. Щоб отримати з функції посилання, потрібно під час оголошення перед її ім’ям написати знак амперсанд (&) і кожного разу при виклику функції перед її ім’ям теж писати амперсанд (&). Зазвичай функція повертає посилання на яку-небудь глобальну змінну, посилання на статичну змінну або посилання на один з аргументів, якщо він був також переданий за посиланням.

```
<?php
  $a = 3;
  $b = 2;
  function & ref($par)
  {
    global $a, $b;
    if ($par % 2 == 0) return $b;
    else return $a;
  }
  $var =& ref(4);
  echo $var, " і ", $b"<br>";
  // виведе 2 і 2
  $b = 10;
  echo $var, " і ", $b"<br>";
  // виведе 10 і 10  ?>
```

Під час використання синтаксису посилань в змінну *$var* не копіюється значення змінної *$b* поверненою функцією *$ref*, а створюється посилання на цю змінну. Тобто тепер змінні *$var* і *$b* ідентичні і змінюватимуться одночасно.

Внутрішні (вбудовані) функції. З деякими з вбудованих функцій, такими як *echo(), print(), date(), include(),* ми вже познайомилися. Насправді всі перераховані функції, окрім *date()*, є мовними конструкціями. Вони входять в ядро PHP і не вимагають ніяких додаткових налаштувань і модулів. Функція *date()* теж входить до складу ядра PHP і не вимагає налаштувань. Але є і функції для роботи з якими потрібно встановити різні бібліотеки та підключити відповідний модуль. Наприклад, для використання функцій роботи з базою даних MySql слід скомпілювати PHP з підтримкою цього розширення. Останнім часом найбільш поширені розширення і їх функції спочатку включають до складу PHP так, щоб з ними можна було працювати без будь-яких додаткових налаштувань інтерпретатора.

ІІ. Робота з масивами даних.

Масив – це тип даних, з якими повинні бути визначені конкретні зазначені операції.

Масив можна створити двома способами:

1. За допомогою конструкції *array*:
```
$array_name = array("key1"=>"value1", "key2"=>"value2");
```
2. Безпосередньо задаючи значення елементам масиву:
```
$array_name["key1"] = value1;
```

Масиви можна об’єднувати і порівнювати. Об’єднують масиви за допомогою стандартного оператора «+». Якщо у нас є два масиви $a і $b, то

результатом їх об’єднання буде масив $c, що складається з елементів $a, до яких справа дописані елементи масиву $b. Причому, якщо зустрічаються співпадаючі ключі, то в результуючий масив включається елемент з першого масиву, тобто з $a. Таким чином, якщо складаються масиви в мові PHP, то від зміни місць доданків сума змінюється. Наприклад:

```
<?php
  $a = array("i"=>"Інформатика", "м"=>"Математика");
  $b = array("и"=>"Історія", "м"=>"Біологія", "ф"=>"Фізика");
  $c = $a + $b;
  $d = $b + $a;
  print_r($c);
// отримаємо: Array([i]=>Інформатика [м]=>Математика[ф]=>Фізика)
  print_r($d);
// отримаємо: Array([i]=>Історія [м]=>Біологія [ф]=>Фізика)
?>
```

Порівнювати масиви можна, перевіряючи їх рівність-нерівність або еквівалентність-нееквівалентність. Рівність масивів – це коли співпадають всі пари ключ-значення елементів масивів. Еквівалентність – коли окрім рівності значень і ключів елементів потрібно ще, щоб елементи в обох масивах були записані в одному і тому ж порядку. Рівність значень в PHP позначається символом «==», а еквівалентність – символом «===»:

```
<?php
  $a = array("и"=>"Інформатика", "м"=>"Математика");
  $b = array("м"=>"Математика", "I"=>"Інформатика");
  if ($a == $b) echo "Масиви рівні і";
  else echo "Масиви НЕ рівні і ";
  if ($a === $b) echo " еквівалентні";
  else echo " НЕ еквівалентні";
  // отримаємо echo "Масиви рівні і НЕ еквівалентні"
?>
```

Розглянемо ще одну важливу операцію з масивом – підрахунок кількості його елементів. Для її реалізації в PHP є спеціальна функція *count*. Дана функція обчислює число елементів в змінній взагалі. Якщо застосувати її до будь-якої іншої змінної, вона поверне 1. Виняток становить змінна типу NULL – count(NULL) має значення 0. Крім того, застосовуючи цю функцію до багатовимірного масиву, щоб отримати число його елементів, потрібно використовувати додатковий параметр *COUNT_RECURSIVE*.

```
<?php
  $del_items = array("langs" => array("10"=>"Python",
"12"=>"Lisp"), "other"=>"Інформатика");
  echo count($del_items)."<br>";
```

```
    // виведе 2
    echo count($del_items, COUNT_RECURSIVE);
    // виведе 4
  ?>
```

Функція ***in_array("шукане значення", "масив" ["обмеження на тип"]);*** – дозволяє встановити чи міститься в заданому масиві шукане значення. Якщо третій аргумент заданий як true, то в масиві потрібно знайти елемент, співпадаючий з шуканим не тільки за значенням, але і за типом. Якщо шукане значення – рядок, то порівняння чутливе до регістру.

Наприклад, є масив не вивчених нами мов програмування. Ми хочемо дізнатися, чи міститься в цьому масиві мова PHP. Напишемо для цього програму:

```
<?php
$langs = array("Lisp", "Python", "Java", "PHP", "Perl");
if (in_array("PHP" $langs)) echo "Треба б вивчити
PHP<br>";
// виведе повідомлення "Треба вивчити PHP"
if (in_array("php" $langs)) echo "Треба вивчити php<br>";
// нічого не виведе, оскільки в масиві є рядок "PHP", а не "php"
?>
```

Функція **array_search** також призначена для пошуку значення в масиві. На відміну від in_array в результаті роботи array_search повертає значення ключа, якщо елемент знайдений. Синтаксис обох функцій однаковий:

```
array_search("шукане значення", "масив" ["обмеження на тип"]);
```

Нехай у нас є масив мов програмування, які ми знаємо, причому ключем кожного елемента є номер, вказуючий, якою за рахунком була вивчена ця мова:

```
<?php
$langs = array("Lisp", "Python", "Java", "PHP", "Perl");
if (!array_search("PHP" $langs)) echo "Треба б вивчити PHP<br>";
  else
  {
    $k = array_search("PHP" $langs);
    echo "PHP вивчений $k – м";
  }
?>
```

У результаті ми отримаємо стрічку: `PHP вивчений 3 – м.`

Функція ***array_keys()*** вибирає всі ключі масиву. Але у неї є додатковий аргумент, за допомогою якого можна отримати список ключів елементів з конкретним значенням. Синтаксис цієї функції такий:

```
array_keys ("масив" ["значення для пошуку"]);
```

Функція *array_keys()* повертає як стрічкові, так і числові ключі масиву, організовуючи всі значення у вигляді нового масиву з числовими індексами.

Якщо є функція для отримання всіх ключів масиву, то можна припустити, що існує і функція для набуття всіх значень масиву. Це функція `array_values(масив)`. Всі значення переданого нею масиву записуються в новий масив, проіндексований цілими числами, тобто всі ключі масиву втрачаються, залишаються тільки значення.

Функція ***array_unique(масив)*** повертає новий масив, в якому елементи, що повторюються, фігурують в одному екземплярі. Таким чином, замість декількох однакових значень і їх ключів ми маємо одне значення.

Всі елементи масиву перетворяться в рядки і сортуються. Потім обробник запам'ятовує перший ключ для кожного значення, а решту ключів ігнорує.

Сортування масивів. Необхідність сортування даних, у тому числі і даних, що зберігаються у вигляді масивів, дуже часто виникає при вирішенні найрізноманітніших завдань. В PHP це робиться однією простою командою.

Функція ***sort*** має наступний синтаксис: ***sort (масив [, прапорці]);*** – сортує масив, тобто упорядковує його значення за збільшенням. Ця функція видаляє всі ключі, що існували в масиві, замінюючи їх числовими індексами, відповідними новому порядку елементів. У разі успішного завершення роботи вона повертає true, в інакшому випадку – false.

Нехай у нас є два масиви: ціни товарів – їх назви і, навпаки, назви товарів – їх ціни. Упорядкуємо ці масиви за збільшенням:

```
<?php
  $items = array(10 => "хліб", 20 => "молоко", 30 =>
"бутерброд");
  sort($items); /* рядки сортуються в алфавітному порядку,
ключі втрачаються */
  print_r($items);
  $rev_items = array("хліб" => 10, "бутерброд" => 30,
"молоко" => 20);
  sort($rev_items); /* числа сортуються за збільшенням,
ключі втрачаються */
  print_r($rev_items);
?>
```

Отримаємо:

```
Array ( [0] => бутерброд [1] => молоко [2] => хліб )
Array ( [0] => 10 [1] => 20 [2] => 30 )
```

Як додатковий аргумент прапорця може використовуватися одна з наступних констант:

– `SORT_REGULAR` – порівнювати елементи масиву звичайним способом;

– SORT_NUMERIC – порівнювати елементи масиву як числа;

– SORT_STRING – порівнювати елементи масиву як рядки.

Функції asort, rsort, arsort. Якщо потрібно зберігати індекси елементів масиву після сортування, то потрібно використовувати функцію `asort (масив [, прапори])`. Якщо необхідно відсортувати масив у зворотному порядку, тобто від найбільшого значення до найменшого, то можна задіювати функцію `rsort (масив [, прапорці])`. А якщо при цьому потрібно ще і зберегти значення ключів, то слід використовувати функцію `arsort(масив [, прапорці])`. Синтаксис у цих функцій абсолютно такий же, як у функції sort. Відповідно і значення прапорів можуть бути такими ж.

Може виникнути необхідність в сортуванні масиву за значеннями його ключів. Наприклад, якщо у нас є масив даних про книги, як в приведеному вище прикладі, то цілком ймовірно, що ми захочемо відсортувати книги по іменах авторів. Для цього в PHP також не потрібно писати багато рядків коду – можна просто скористатися функцією *ksort()* для сортування за збільшенням (прямий порядок сортування) або *krsort()* – для зворотнього порядку сортування.

PHP пропонує користувачеві можливість самому задавати критерії для сортування даних. Критерій задається за допомогою функції, ім'я якої вказується як аргумент для спеціальних функцій сортування *usort()* або *uksort()*. За назвами цих функцій можна здогадатися, що *usort()* сортує значення елементів масиву, а *uksort()* – значення ключів масиву за допомогою визначеної користувачем функції. Обидві функції повертають *true*, якщо сортування пройшло успішно, і *false* – в інакшому випадку. Їх синтаксис виглядає таким чином:

```
usort (масив, сортуюча функція);
uksort (масив, сортуюча функція);
```

Сортуюча функція повинна мати два аргументи. У них інтерпретатор передає пари значень елементів для функції *usort()* або ключів масиву для функції *uksort()*.

Як і для інших функцій сортування, для функції *usort()* існує аналог, що не змінює значення ключів, – функція *uasort()*.

Виділення підмасиву. Оскільки масив – це набір елементів, то цілком імовірно, що іноді потрібно виділити з нього який-небудь підмасив. У PHP для цих цілей використовується функція *array_slice*. Її синтаксис такий:

```
array_slice (масив, номер_елемента [, довжина]);
```

Ця функція виділяє підмасив довжини. Довжиною є масив, починаючи з елемента, номер якого заданий параметром *номер_елемента*. Позитивний *номер_елемента* вказує на порядковий номер елемента відносно початку масиву, негативний – на номер елемента з кінця масиву.

```
<?php
  $arr = array(1, 2, 3, 4, 5);
  $sub_arr = array_slice($arr, 2);
  print_r($sub_arr);
  /* виведе Array ( [0] => 3  [1] => 4 [2] => 5 ),
     тобто підмасив, що складається з елементів 3, 4, 5 */
  $sub_arr = array_slice($arr,-2);
  print_r($sub_arr);
  // виведе Array ( [0] => 4 [1] => 5 )
  // тобто підмасив, з елементів 4, 5
?>
```

Якщо задати параметр *довжина* під час використання *array_slice*, то буде виділений підмасив, що має рівно стільки елементів, скільки задано цим параметром.

Функція ***array_chunk*** розбиває масив на декілька підмасивів заданої довжини. Синтаксис її такий:

```
array_chunk ( масив, розмір [, зберігати_ключі]);
```

У результаті роботи *array_chunk()* повертає багатовимірний масив, елементами якого є отримані підмасиви. Якщо задати параметр зберігати ключі як true, то під час розбиття будуть збережені ключі початкового масиву. В інакшому випадку ключі елементів замінюються числовими індексами, які починаються з нуля.

Сума елементів масиву. Функція, що обчислює суму всіх елементів масиву називається ***array_sum()*** і як параметр їй передається тільки ім’я масиву, суму значень елементів якого потрібно обчислити.

Як приклад використання цієї функції наведемо рішення складнішої задачі, ніж просто обчислення суми елементів. Цей приклад також ілюструє застосування функції *array_slice()* згадуваної раніше.

Здійснивши грунтовний аналіз використання функцій в мові програмування PHP можемо виокремити основні їх сфери застосування, зокрема вони використовуються для: побудови модульних програм; повторного використання алгоритмів; організації логіки програмування сучасних вебзастосунків.

ЛЕКЦІЯ 17

Тема: *Об'єктно-орієнтоване програмування в PHP.*

План:

1. Класи і об'єкти в PHP. Властивості класів. Визначення методів класу.
2. Використання конструкторів, деструкторів та інтерфейсів. Інкапсуляція, наслідування, агрегація та композиція в PHP.

І. Класи і об'єкти в PHP. Властивості класів. Визначення методів класу.

У PHP класи й об'єкти становлять основу об'єктно-орієнтованого програмування (ООП). Такий підхід дає змогу описувати програму як сукупність взаємодіючих об'єктів, кожен із яких має власні властивості та поведінку. Використання класів забезпечує модульність, повторне використання коду, логічну організацію програмних компонентів і спрощує розроблення складних вебзастосунків.

Клас – це базове поняття в об'єктно-орієнтованому програмуванні (ООП). Класи утворюють синтаксичну базу ООП. Їх можна розглядати як свого роду «контейнери» для логічно пов'язаних даних і функцій.

Клас – це шаблон або модель, за якою створюються об'єкти. У класі визначаються: властивості (дані), методи (дії).

Клас оголошується за допомогою ключового слова class. Загальний синтаксис:

```
class Student {
    public $name;
}
```

Технологія ООП в мові PHP володіє трьома головними перевагами:

- *вона проста для розуміння* – ООП дозволяє мислити категоріями повсякденних об'єктів;
- *підвищено надійна і проста для супроводу* – правильне проектування забезпечує простоту розширення і модифікації об'єктно-орієнтованих web-додатків. Модульна структура дозволяє вносити незалежні зміни в різні частини додатку, зводячи до мінімуму ризик помилок програмування;
- *прискорює цикл розробки* – модульність і тут відіграє важливу роль, оскільки різні компоненти об'єктно-орієнтованих web-додатків можна легко використовувати в інших web-застосунках, що зменшує надмірність коду і знижує ризик внесення помилок при копіюванні.

Об’єкти представляють собою часткову інформацію про певну сутність, а власне певну модель, що адекватна завданню що треба вирішити. Цей варіант представлення називається абстракція даних. При такому представленні з об'єктом працювати набагато простіше, ніж з низькорівневим представленнями з описом всіх можливих властивостей і методів.

В парадигмі об’єктно-орієнтованого програмування в PHP основою є певний об’єкт чи сукупність об’єктів, їхні властивості, методи і події. Власне з появою ООП і з’явилися такі терміни як клас, наслідування, поліморфізм, інкапсуляція. Якщо розглядати поняття об’єкту концептуально, то об’єкт є лише екземпляром певного класу об’єктів.

Об’єкт – окремий примірник структури даних, визначається класом. Визначення класу формулюється тільки один раз, після чого створюються всі необхідні об’єкти, які до нього належать.

Створення об’єкта здійснюється через оператор new.

```
$st = new Student();
```

Після цього об’єкт має доступ до властивостей і методів класу.

Властивість – один з іменованих компонентів визначення даних у визначенні класу.

Усередині об’єкту дані і код (члени класу) можуть бути або відкриті, або ні. Відкриті дані і члени класу є доступними для інших частин програми, які не є частиною об’єкта. А ось закриті дані і члени класу доступні тільки усередині цього об’єкта.

Властивості описують стан об’єкта.

```
class Student {
    public $name;
    public $age;
}
```

Заповнення властивостей:

```
$st->name = "Іван";
$st->age = 20;
```

Для звернення використовується оператор ->

```
echo $st->name;
```

Метод – компонент класу, який за своїм призначенням є функцією.

Метод описується так само, як і звичайна користувацька функція. Методу також можна передавати параметри. За загальноприйнятими правилами імена класів ООП починаються з великої літери, а всі слова в іменах методів, крім першого, починаються з великих літер (перше слово починається з малої літери). Зрозуміло, ви можете використовувати будь позначення, які вважаєте зручними; головне – виберіть стандарт і дотримуйтеся його.

ІІ. Використання конструкторів, деструкторів та інтерфейсів. Інкапсуляція, наслідування, агрегація та композиція в PHP.

Специфіка ООП помітно підвищує ефективність праці програмістів і дозволяє їм створювати більш потужні, масштабовані та ефективні програми. Об’єктно-орієнтоване програмування засноване на інкапсуляції, поліморфізмі, спадкуванні.

Інкапсуляцією називається включення різних дрібних елементів в більш великий об’єкт, внаслідок чого програміст працює безпосередньо з цим об’єктом. Це призводить до спрощення програми, оскільки з неї виключаються другорядні деталі.

Поліморфізм дозволяє використовувати одні й ті ж імена для схожих, але технічно різних завдань. Головним у поліморфізмі є те, що він дозволяє маніпулювати об’єктами шляхом створення стандартних інтерфейсів для схожих дій. Поліморфізм значно полегшує написання складних програм.

Успадкування дозволяє одному об’єкту набувати властивостей іншого об’єкта. При копіюванні створюється точна копія об’єкта, а при спадкуванні точна копія доповнюється унікальними властивостями, які характерні тільки для похідного об’єкта.

Досить часто при створенні об’єкта потрібно задати значення деяких властивостей. На щастя, розробники технології ООП врахували цю обставину і реалізували його в концепції конструкторів.

Конструктор являє собою метод, що задає значення деяких властивостей (а також може викликати інші методи). Конструктори викликаються автоматично при створенні нових об’єктів. Щоб це стало можливим, ім'я методу-конструктора має збігатися з ім’ям класу, в якому він міститься. Приклад конструктора:

```
<?
class Webpage {
var $ bgcolor;
 function Webpage ($ color) {
 $ this-> bgcolor = $ color;
 }
}
// Викликати конструктор класу Webpage
$ page = new Webpage ("brown");
?>
```

Раніше створення об’єкта і ініціалізація властивостей виконувалися роздільно. Конструктори дозволяють виконати ці дії за один етап.

В залежності від кількості переданих параметрів можуть викликатися різні конструктори. У розглянутому прикладі об'єкти класу Webpage можуть створюватися двома способами. По-перше, ви можете викликати конструктор, який просто створює об’єкт, але не ініціалізує його властивості:

```
$ page = new Webpage;
```

По-друге, об'єкт можна створити за допомогою конструктора, визначеного в класі – в цьому випадку ви створюєте об’єкт класу *Webpage* і привласнюєте значення його властивості *bgcolor*:

```
$ page = new Webpage ("brown");
```

Деструктори. У PHP відсутня безпосередня підтримка деструкторів. Тим не менш, ви можете легко імітувати роботу деструктора, викликаючи функцію PHP *unset ()*. Ця функція знищує вміст змінної і повертає займані нею ресурси системі. З об’єктами *unset ()* працює так само, як і зі змінними. Припустимо, ви працюєте з об’єктом *$ Webpage*. Після завершення роботи з цим конкретним об'єктом викликається функція:

```
unset($Webpage);
```

Ця команда видаляє з пам’яті весь вміст *$ Webpage*. Діючи в дусі інкапсуляції, можна помістити виклик *unset()* в метод з ім’ям *destroy ()* і потім викликати його:

```
$Website-> destroy();
```

Необхідність у виклику деструкторів виникає лише при роботі з об’єктами, що використовують великий обсяг ресурсів, оскільки всі змінні і об'єкти автоматично знищуються по завершенні сценарію.

Звернення до елементів класів здійснюється за допомогою оператора :: «подвійна двокрапка». Використовуючи «подвійна двокрапка», можна звертатися до методів класів.

При зверненні до методів класів, програміст повинен використовувати імена цих класів.

```
<? php
class A {
     function example () {
echo "Це первісна функція A :: example (). <br>";
     }}
class B extends A {
     function example () {
echo "Це перевизначення функція B :: example (). <br>";
         A :: example ();
     }}
```

```
// Не потрібно створювати об'єкт класу A.
// Це початкова функція A :: example ().
A :: example ();
// Створюємо об'єкт класу B.
$ b = new B;
// Виводить наступне:
// Це перевизначення функція B :: example ().
// Це початкова функція A :: example ().
$ b-> example ();
?>
```

У PHP об’єктна модель була повністю переписана для того, щоб відразу працювати з покажчиками на об’єкт. Вже не потрібно явно передавати об’єкти або привласнювати їх за посиланням, це робиться автоматично.

Нові можливості об’єктної моделі є занадто численними. Наведемо огляд головних змін: *public / private / protected*-модифікатори доступу для методів і властивостей. Дозволяють управляти доступом до методів і властивостей. Тепер видимість властивостей і методів може бути визначена ключовими словами: public, private, protected.

Модифікатор public дозволяє звертатися до властивостей і методів звідусіль.

Модифікатор private дозволяє звертатися до властивостей і методів тільки всередині поточного класу.

Модифікатор protected дозволяє звертатися до властивостей і методів тільки поточного класу і класу, який успадковує властивості і методи поточного класу.

У PHP ***інтерфейс*** – це клас, в якому всі методи є абстрактними і відкритими. Інтерфейс мови PHP являє собою інструмент для створення надійного коду, що визначає методи, які повинні бути реалізовані за допомогою класу без визначення обробки цих методів. Це потужна і проста концепція, яка використовується в об’єктно-орієнтованому PHP. Для його створення використовується ключове слово *interface*, він визначається так само, як клас PHP. Методи, що містяться в ньому, не мають ніякого функціоналу, але інтерфейс задає, які методи необхідні для інших класів. Будь-клас, спадщини від інтерфейсу, повинен містити ті ж методи, які містяться в інтерфейсі, інакше видається помилка.

В інтерфейсі ООП PHP ми задаємо тільки імена методів і їх параметри, а реалізовані вони можуть бути пізніше. Зазвичай оголошують все методи інтерфейсу як *public*. Для реалізації інтерфейсу використовується ключове слово *implements*. При необхідності клас може реалізувати більш одного інтерфейсу, між собою вони розділяються комою. Як і клас, інтерфейс може містити константи.

Єдина відмінність полягає в тому, що вони не можуть бути перевизначені в похідному класі.

Згідно основам ООП PHP, вони відрізняються один від одного наступним:

1. В інтерфейсі всі методи є абстрактними (без реалізації). В абстрактному класі лише деякі методи є абстрактними. Абстрактний клас повинен містити, принаймні, один абстрактний метод. Інакше це буде стандартний клас PHP.
2. В інтерфейсі PHP всі оголошені методи є відкритими, а в абстрактному класі методи можуть бути відкритими, приватними або захищеними. Отже, для інтерфейсів існує обмеження на використання модифікаторів доступу, а в абстрактному класі таких обмежень немає.
3. Клас може реалізувати необмежену кількість інтерфейсів. У той же час клас PHP може породити тільки один абстрактний клас.
4. Ви повинні перевизначати всі методи інтерфейсу, а в абстрактному класі у вас є вибір: перевизначати методи або залишити їх без змін.
5. Інтерфейси мають більш суворі правила, ніж абстрактні класи. Інтерфейс PHP покликаний забезпечити певну логіку, він виступає в якості порожній оболонки або шаблону для інших класів.

В об'єктно-орієнтованому програмуванні під ***агрегацією*** (або *делегуванням*) мають на увазі методику створення нового класу з уже існуючих класів шляхом їх включення. Вкладені об'єкти нового класу зазвичай оголошуються закритими, що робить їх недоступними для прикладних програмістів, які працюють з класом. З іншого боку, творець класу може змінювати ці об'єкти, не порушуючи при цьому роботи існуючого клієнтського коду. Крім того, заміна вкладених об'єктів на стадії виконання програми дозволяє динамічно змінювати її поведінку. Механізм успадкування такою гнучкістю не володіє, оскільки для похідних класів встановлюються обмеження, що перевіряються на стадії компіляції.

Ще однією особливістю об'єктно-орієнтованого програмування в PHP є можливість реалізовувати так званий ***композиційний підхід***. Полягає він у тому, що є клас-контейнер, він же агрегатор, який включає в себе виклики інших класів. В результаті виходить, що при створенні об'єкта класу-контейнера, також створюються об'єкти включених в нього класів.

Не слід плутати композицію зі спадкуванням, в тому числі множинним. Спадкування передбачає приналежність до якоїсь спільності (схожість), а композиція – формування цілого з частин. Успадковуються атрибути, тобто можливості іншого класу, при цьому об'єктів безпосередньо батьківського класу не створюється. При композиції же клас-агрегатор створює об'єкти інших класів.

ЛЕКЦІЯ 18

Тема: *Шаблони об'єктно-орієнтованого програмування.*

План:

1. Шаблони об'єктно-орієнтованого програмування в PHP.
2. Використання фреймворків при розробці програмних продуктів.

І. Шаблони об'єктно-орієнтованого програмування в PHP.

У середовищі PHP шаблони доцільно розглядати як інструмент розширення й упорядкування програмного коду, що сприяє ефективнішій організації процесу розроблення. Їх застосування не лише зменшує обсяг рутинних операцій під час програмування, а й забезпечує чіткий структурний розподіл завдань між учасниками проєкту. Значення такого підходу особливо зростає у великих програмних системах, де збільшується кількість розробників, ускладнюється архітектура програмного продукту та виникає потреба у подальшому супроводі й модернізації коду.

За своєю сутністю шаблони об'єктно-орієнтованого програмування в PHP є узагальненими моделями організації взаємодії класів і об'єктів, орієнтованими на розв'язання типових завдань проєктування в певних умовах. Вони не є готовими бібліотеками чи завершеними програмними компонентами, які можна безпосередньо інтегрувати в систему, а виступають концептуальними рішеннями, що адаптуються до конкретної програмної ситуації.

Використання шаблонів ООП дозволяє пришвидшити розроблення програмного забезпечення, оскільки більшість із них сформувалися на основі перевірених практик і вже довели свою ефективність у різних програмних проєктах. Крім того, шаблони сприяють спрощенню сприйняття складних архітектурних рішень, подаючи їх у формі зрозумілих моделей. Водночас їх застосування має бути обґрунтованим, оскільки використання шаблонів у ситуаціях, де вони не є необхідними, може призвести до надмірного ускладнення програмної структури.

На даний момент в PHP існують наступні шаблони проектування, які за їх призначенням можна розділити на три категорії:

1. **Породжуючі шаблони** – використовуються для створення об'єктів, які можна відокремлювати від їх системи реалізації:

Abstract Factory – дозволяє створювати цілі групи взаємопов'язаних об'єктів, які, будучи створеними однією фабрикою, реалізують загальну поведінку.

Builder – використовується для відділення процесу конструювання складного об’єкта від його уявлення, так що в результаті одного і того ж конструювання можуть виходити різні об’єкти.

Factory Method – надає подкласам інтерфейс для створення екземплярів деякого класу.

Prototype – використовується для завдання виду створюваних об’єктів на основі об’єкта прототипу, від якого відбувається передача внутрішнього стану (створює нові об’єкти шляхом копіювання прототипу).

2. **Структурні шаблони** – використовуються для формування великих об’єктних структур між безліччю розрізнених об’єктів.

Adapter – призначений для організації використання функцій об’єкта, недоступного для модифікації, через спеціально створений інтерфейс.

Bridge – використовується для відділення абстракції від її реалізації так, щоб і те й інше можна було змінювати незалежно.

Composite – використовується для компонування об’єктів в деревовидні структури для представлення ієрархій, дозволяючи однаково трактувати індивідуальні та складові об’єкти.

Decorator – використовується для динамічного розширення функціональності об’єкту. Є гнучкою альтернативою спадкування.

Facade – являє собою уніфікований інтерфейс замість набору інтерфейсів деякої підсистеми. Патерн фасад визначає інтерфейс більш високого рівня, який спрощує використання підсистем.

Flyweight – використовується для зменшення витрат при роботі з великою кількістю дрібних об’єктів.

Proxy – який надає об’єкт, який контролює доступ до іншого об’єкту, перехоплюючи всі виклики (виконує функцію контейнера).

3. **Поведінкові шаблони** – використовуються для управління алгоритмами, відносинами і обов’язками між об’єктами:

Chain of responsibility – служить для ослаблення зв’язку між відправником і отримувачем запиту. При цьому сам по собі запит може бути довільним.

Command – являє собою дію. Об’єкт команди містить в собі сама дія і його параметри.

Interpreter – вирішує таку поширену, але піддану змінам, завдання.

Iterator – являє собою об’єкт, що дозволяє отримати послідовний доступ до елементів об’єкта-агрегату без використання описів кожного з агрегованих об'єктів.

Mediator – забезпечує взаємодію безлічі об’єктів, формуючи при цьому слабку зв’язаність і позбавляючи об’єкти від необхідності явно посилатися один на одного.

Memento – дозволяє, не порушуючи инкапсуляцию, зафіксувати і зберегти внутрішній стан об’єкта так, щоб пізніше відновити його в цей стан.

Observer – створює механізм у класу, який дозволяє отримувати примірника об'єкта цього класу оповіщення від інших об’єктів про зміну їх стану, тим самим спостерігаючи за ними.

State – використовується в тих випадках, коли під час виконання програми об’єкт повинен міняти свою поведінку в залежності від свого стану.

Strategy – призначений для визначення сімейства алгоритмів, інкапсуляції кожного з них і забезпечення їх взаємозамінності. Це дозволяє вибирати алгоритм шляхом визначення відповідного класу. Шаблон Strategy дозволяє змінювати обраний алгоритм незалежно від об’єктів-клієнтів, які його використовують.

Template Method – визначає основу алгоритму і дозволяє спадкоємцям перевизначити деякі кроки алгоритму, не змінюючи його структуру в цілому.

Visitor – описує операцію, яка повинна бути виконана над кожним об’єктом з деякою довільній структури.

II. Використання фреймворків при розробці програмних продуктів.

WEB-фреймворк – це каркас, призначений для створення динамічних веб-сайтів, мережевих додатків, сервісів або ресурсів.

Він спрощує розробку і позбавляє від необхідності написання рутинного коду. Багато фреймворків спрощують доступ до баз даних, полегшують розробку інтерфейсу, а також зменшують дублювання коду.

Виділяють п’ять типів веб-фреймворків: Requestbased, Component-based, Hybrid, Meta and RIA-based.

Request-based – фреймворки, які безпосередньо обробляють вхідні запити. Збереження стану відбувається за рахунок серверних сесій. Приклади: Django, Ruby on Rails, Struts, Grails.

Component-based – фреймворки, які абстрагують обробку запитів всередині стандартних компонентів і самостійно стежать за станом. Дані каркаси мають багато спільного зі стандартними програмними графічними інтерфейсами. Приклади: JSF, Tapestry, Wicket.

Hybrid-based – фреймворки, які комбінують Request-based та Component-based фреймворки, беручи під свій контроль всі дані і логічний потік в заснованої на запиті моделі. Розробники мають повний контроль над URL, формами, параметрами, cookies і pathinfos. Гібридні фреймворки забезпечують об’єктну модель компонентів, яка поводиться тотожно в багатьох різних ситуаціях, таких як окремі сторінки, перервані запити, подібні порталу фрагменти сторінок та інтегровані віджети. Приклади: RIFE.

Meta-based – фреймворки, що мають ряд базових інтерфейсів для загального обслуговування і основу, яка легко розширюється з метою інтегрування компонентів і служб. Приклад: Keel.

RIA-based – фреймворки, що служать для розробки повноцінних додатків, які запускаються всередині браузера. Приклад: Flex.

Найбільш поширеними є Request-based і Component-based веб-фреймворки. Зібравши і проаналізувавши інформацію, мною було виділено такі характерні компоненти web-фреймворків:

- *шаблонизатор* (відповідає за незалежність верстки від програмного коду);
- *роутер* (розпізнає URL, за яким відбулося звернен- ня до сервера);
- *модуль доступу до бази даних*;
- *модуль кешування* (прискорює завантаження сторінок);
- *модуль безпеки* (аутентифікація і авторизація користувачів);
- *файли конфігурації* (WEB фреймворки також можуть керувати сесіями, вести логи, спрощувати використання технології Ajax та ін).

Переваги використання фреймворків наступні:

- гнучкість і масштабування – завжди має гнучке рішення нестандартних завдань і можливість далі розширення функціоналу шляхом підключення сторонніх бібліотек або окремих класів;
- ефективне використання ресурсів сервера;
- використання підходу модель-вид-контролер (MVC) суттєво розширює функціонування та гнучкість проекту, так як використовується під час проектування та розробки програмного забезпечення;
- наявність детальної документації з використання фреймворку;
- безпека – забираються всі проломи в безпеці, практично немає вузьких місць для SQL-ін’єкцій; фреймворк дозволяє сконцентруватися на вирішенні архітектурних завдань, а не базових;
- якість матеріалу на виході.

Головні переваги фреймворків те, що вони якнайкраще підходять для створення маштабованих і унікальних сайтів. Жоден маштабний проект не розроблений на готовій CMS – вони для цього не призначені. Майже всі унікальні web-додатки розробляються з використанням фреймворків. Web-проект, розроблений за допомогою фреймворку, розвивається динамічно. При зміні вимог змінюється і сайт, для створення нового розділу або внесення новизни в дизайні, достатньо змінити окремий модуль. Замінити окремий блок (модуль), створити новий розділ або внести новизну в дизайні.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Васильєв О.М. Програмування мовою PHP. Видавництво: Ліра-К. 2022. 368 с.
2. Васильєв О.М. Програмування мовою Java. Видавництво: Навчальна книга Богдан. 2020. 696 с.
3. Кеті Сьєрра, Берт Бейтс. Head First. Java. Легкий для сприйняття довідник. Видавництво : Фабула, 2022. 720 с.
4. Руденко Віктор. Вивчаємо Java у школі. У 2-х частинах. Ч.1. Синтаксис мови. Видавництво : Ранок, 2020. 96 с.
5. Duckett, J. (2022). PHP & MySQL: Server-side Web Development. Wiley. 1st ed. 672 p.
6. Java 23 for Absolute Beginners: Learn the Fundamentals of Java Programming (3rd Ed): Samoylov, N. (2022). Learn Java 17 Programming – Second Edition. 2nd ed. 748 pp.
7. Java: The Complete Reference, Eleventh Edition. Herbert Schildt. Oracle Press. 2019. 1871 p.
8. Java: A Beginner's Guide, 9th Edition. Herbert Schildt. Oracle Press. 2022. 752 p.
9. Java: The Complete Reference, 12th Edition: Schildt, H. (2022). Java: The Complete Reference. McGraw-Hill. 12th ed. 1280 p.
10. Nixon, R. (2024). Learning PHP, MySQL & JavaScript: A Step-by-Step Guide to Creating Dynamic Websites (7th ed.). O'Reilly Media.
11. Simon, M. (2024). An Introduction to PHP: Learn PHP 8 to Create Dynamic Websites. Published by Apress Copyright. 1st ed., 625 p.
12. Lengstorf, J., Hansen, T. B., & Prettyman, S. (2022). PHP 8 for Absolute Beginners: Basic Website and Web Application Development. 3rd ed. 429 p.
13. Powers, D. (2022). PHP 8 Solutions: Dynamic Web Design and Development Made Easy. Published by Apress Copyright. 5th ed. 558 p.
14. Learning Java 5th Edition. Marc Loy, Patrick Niemeyer, Daniel Leuck. O'Reilly Media. 2020. 1248 p.

Прикладне та Web-програмування: конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Професійна освіта (комп'ютерні технології)» галузі знань А Освіта спеціальності А5.39 Професійна освіта (Цифрові технології) денної та заочної форм навчання / уклад. В. В. Кабак. Луцьк: ЛНТУ, 2026. 156 с.

Комп'ютерний набір: В.В. Кабак.

Редактор: В.В. Кабак.

Підп. до друку «___»_____2026 р. Формат 60х84/16. Папір офс.
Гарн. Таймс. Ум. друк. арк. 9,75.
Тираж 50 прим.

Луцький національний технічний університет
43018, м. Луцьк, вул. Львівська,75