МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни «**Internet-технології та мова Java**» для студентів напряму підготовки «Комп'ютерні науки» всіх форм навчання

Укладачі: Парфьонов Ю.Е.
Жукарев В.Ю.
Поляков А.О.

Відповідальний за випуск Пономаренко В. С.

Харків. Вид. ХНЕУ, 2008

Затверджено на засіданні кафедри інформаційних систем.
Протокол № 1 від 28.08.2007 р.

Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни «Internet-технології та мова Java» для студентів напряму підготовки «Комп'ютерні науки» всіх форм навчання / Укл. Ю. Е. Парфьонов, В. Ю. Жукарев, А. О. Поляков. – Харків: Вид. ХНЕУ, 2008. - 84 с. (Укр. мов.)

Наведено завдання до методичних рекомендацій, що сприяють поглибленню й закріпленню знань з теоретичних питань використання Internet-технологій та придбанню практичних навичок з розробки Java-додатків для роботи з Internet-ресурсами.

Рекомендовано для студентів напряму підготовки «Комп'ютерні науки» всіх форм навчання.

ВСТУП

Java – мова програмування, яка з самого початку була орієнтована на розробку Internet-додатків. Її синтаксис близький до синтаксису мови C++. Головна перевага мови Java полягає в тому, що за її допомогою можливо створення додатків, здатних працювати на різних платформах.

Як відомо, платформою називають сполучення апаратної архітектури, що визначається типом процесора, який використовується, та операційної системи. При написанні програм розробник застосовує засоби платформи для доступу до мережі, роботи з графічним інтерфейсом користувача та. ін. Різні платформи підтримують різні інтерфейси програмування додатків – API (Application Programming Interface).

Слід зазначити, що до мережі Internet підключені комп'ютери найрізноманітніших типів – Pentium PC, Macintosh, робочі станції Sun та ін. У цих комп'ютерах використовуються різні процесори, головним чином виробництва фірм Intel і AMD. У свою чергу, для процесорів Intel та AMD існує кілька операційних систем, наприклад, Microsoft Windows, Sun Solaris, різні різновиди операційної системи UNIX. Тому Java-технології, які призначені для роботи на різних платформах і не залежать від конкретного типу процесору й операційної системи, мають істотні переваги над аналогами.

Додаток Java не звертається прямо до API операційної системи, тому що використовує стандартні бібліотеки класів, які містять компоненти для організації інтерфейсу користувача, звертання до файлів, роботи в мережі тощо. Внутрішня реалізація бібліотек класів залежить від платформи. Всі завантажувальні модулі, що реалізують можливості цих бібліотек, поставляються разом з віртуальною машиною Java. Наприклад, для операційної системи Windows поставляються бібліотеки динамічного завантаження (Dynamic Link Library), всередині яких реалізована вся функціональність стандартних класів Java.

Таким чином, Java дозволяє створювати додатки, які не потребують перекомпіляції при перенесенні їх на різні платформи, що широко використовується для розробки Internet-додатків.

Курс лабораторних робіт, що пропонується, призначений для вивчення основних технологій розробки автономних і мережних додатків мовою Java. Для цього рекомендується використовувати інтегроване

середовище розробки, наприклад NetBeans, або програмні засоби розробника – Java Development Kit (JDK) компанії Sun Microsystems. Розглянуті в курсі приклади додатків сумісні з JDK версії 1.5.

У даній роботі розглядаються основні технології розробки додатків мовою Java, які призначені для створення автономних та розподілених додатків, аплетів і Web-додатків. Особливістю більшості лабораторних робіт є їхня орієнтація на розробку графічного користувальницького інтерфейсу, що дає можливість надати користувачу мережі Internet розвинуті засоби діалогу для роботи з інформацією на Web-сторінках.

Загальні рекомендації до виконання лабораторних робіт

Звіт з будь-якої лабораторної роботи повинен містити:

1. Титульний лист:
   назва дисципліни;
   тема лабораторної роботи;
   дата виконання роботи;
   П.І.Б. студента, курс, номер групи;
   П.І.Б., посада викладача.
2. Зміст (нумерований перелік назв пунктів роботи (див. пункт з) із зазначенням номерів сторінок).
3. Опис виконаних завдань:
   а) умова завдання;
   б) **опис архітектури програми** (склад, структура класів, зв‘язки між ними, алгоритми тощо);
   в) **вихідний код програми з документаційними коментарями JavaDoc** для класів (призначення класу) та методів (призначення, опис параметрів кожного методу, значення, що він повертає);
   г) приклади результатів роботи програми на тестових вихідних даних;
4. **Висновки з лабораторної роботі з** урахуванням усіх завдань, що були виконані:
   аналіз результатів, які були одержані за кожним пунктом завдання;
   аналіз результатів тестування програм;

ступінь відповідності програм, що були розроблені, постановці завдання;

інша інформація.

Вимоги до оформлення звіту:

1. Звіт з лабораторної роботи виконується у друкований спосіб на аркушах формату А4 відповідно до державних стандартів.
2. Аркуші звіту з'єднуються скріпками або іншим загальноприйнятим способом.

ЛАБОРАТОРНА РОБОТА 1
РОЗРОБКА ДОДАТКІВ З ВИКОРИСТАННЯМ БАЗОВИХ ЕЛЕМЕНТІВ ООП

Мета лабораторної роботи – одержати загальне уявлення про мову програмування Java.

Перед виконанням лабораторної роботи студент повинен знати:

Основи об'єктно-орієнтованого програмування.

Основні джерела інформації з мови Java, якы необхідні при розробці програм.

Загальну структуру Java-програми.

Після виконання лабораторної роботи студент повинен вміти:

Працювати з електронною документацією Java-розроблювача.

Виконувати основні технологічні операції з розробки програм в інтегрованому середовищі NetBeans.

Розробляти найпростіші консольні Java-програми.

Загальні відомості про мову програмування Java і використання в ній базових елементів об'єктно-орієнтованого програмування (ООП)

Особливості використання Jama программ наведено на рис 1.

Основні засоби JDK (операційна система Windows, рис. 2):

компілятор (javac.exe);

налагоджувач (jdb.exe);

завантажник додатків (java.exe);

генератор документації (javadoc.exe);

архіватор (jar.exe).

Основні засоби JRE (операційна система Windows, рис. 2):

завантажник додатків (java.exe);

віртуальна машина (jvm.dll).

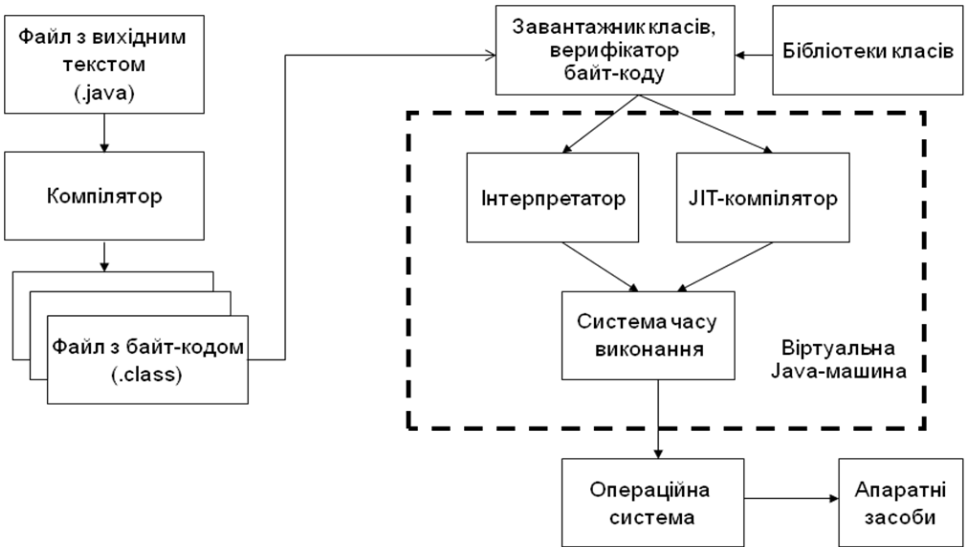


Рис. 1. **Особливості створення та виконання Java-програми**

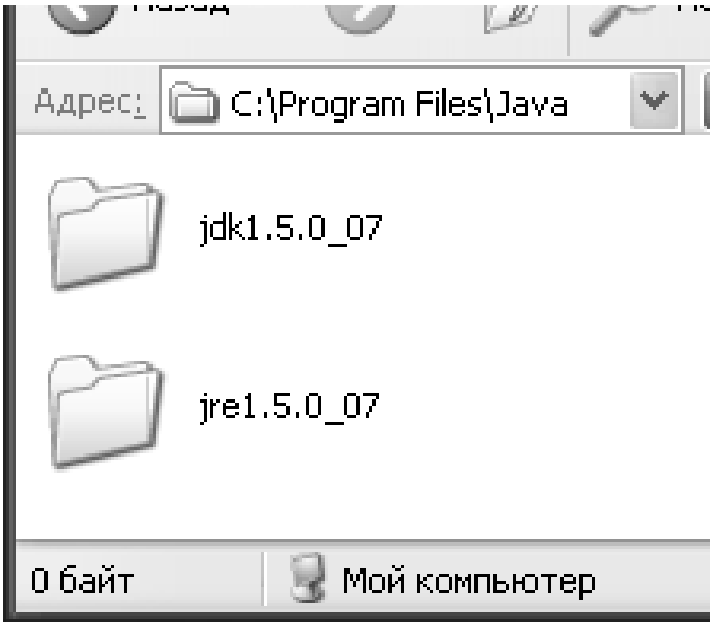


Рис. 2. **Java Development Kit (JDK) та Java Runtime Environment (JRE)**

Найпростіша Java-програма:

```
public class HelloApp {
        public static void main(String arg[ ]) {
```

```
            System.out.println("Hello, Java world");
        }
    }
```

Послідовність компіляції та запуску програми:

1. Набрати текст програми в будь-якому текстовому редакторі.
2. Зберегти його у файл (для класу **HelloApp** ім'я файлу повинне бути **HelloApp.java**), наприклад у каталозі **C:\Temp**.
3. Запустити стандартну програму «Командний рядок» (операційна система Windows).
4. Відкомпілювати вихідний код програми за допомогою компілятора (виконати команду **javac C:\Temp\HelloApp.java**).
5. Запустити програму на виконання (виконати команду **java -cp C:\Temp HelloApp**).

Алфавіт мови Java:

букви (a - z; A - Z);

цифри (0 - 9);

спеціальні символи (**~ % * ( ) - +** тощо);

керуючі символи **(\n \t** тощо).

Ключові слова Java наведені в табл. 1.

Таблиця 1

Основні ключові слова мови Java

abstract	continue	finally	interface	return	throws
boolean	default	float	long	short	transient
break	do	for	native	static	try
byte	double	if	new	super	void
case	else	implements	package	switch	volatile
catch	enum	import	private	synchronized	while
char	extends	instanceof	protected	this	
class	final	int	public	throw	

Вбудовані типи даних мови Java подані в табл. 2.

Функції в мові Java:

будь-яка функція повинна бути членом класу (методом);

методи можуть приймати або не приймати параметри, повертати або не повертати значення;

параметри можуть передаватися в методи за значенням або за посиланням;

методи можуть бути статичними або методами екземплярів класу (об'єктів).

Таблиця 2

Розрядність та діапазони значень типів даних мови Java

Ім'я	Розрядність	Діапазон значень
char	16	0……65536
byte	8	-128…+127
short	16	-32768…+32767
int	32	$-2^{31}…+2^{31}-1$
long	64	$-2^{63}…+2^{63}-1$
float	32	$3,4\times10^{-38}…3,4\times10^{38}$
double	64	$1,7\times10^{-308}…1,7\times10^{308}$
boolean		true; false

Простори імен у Java (пакети):

пакет – спосіб логічної організації коду класів у єдину групу;

пакету відповідає каталог файлової системи;

пакет задається за допомогою ключового слова **package**:

package mypackage;

звертання до класу пакета: **mypackage.MyClass;**

потрібно не вказувати пакета явно.

Взаємозв’язок каталогів файлової системи та Java-пакетів наведена на рис. 3.

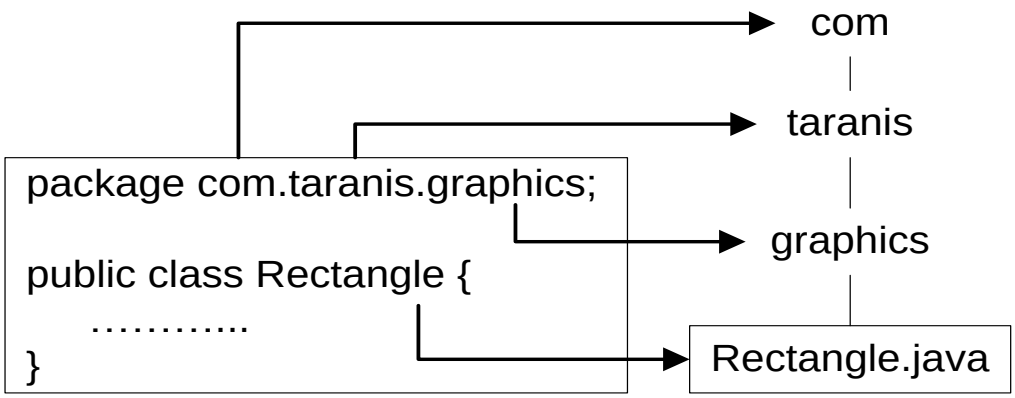


Рис. 3. **Пакети й каталоги файлової системи**

Структура вихідного файлу (.java):

package *packagename*; // тільки одна інструкція на файл
import *packagename1.ClassName1*; // імпорт бібліотек
……
import *packagename1.ClassNameM*; // імпорт бібліотек
public class *One* { // тільки один public-клас на файл
……
}
class Two {
……
}
……
class N {
……
}

Масиви в Java:

1. Одновимірні:

- **int array1[ ] = new int[10];**
  for (int i = 0; i < array1.length ; i++)
  array1[i] = 2 * i;
- **int array2[ ] = {10, 20, 30, 40, 50};**
- **int array3[ ] = new int[ ] {10, 20};**

Багатовимірні:

- **int array1[ ][ ] = new int[2][3];**
  for (int i = 0; i < array1.length ; i++)
  for (int j = 0; j < array1[i].length ; j++)
  array1[i][j] = 2 * I + j;
- **int array2[ ][ ] = { {10,20,25}, {30,40,45} };**
- **int array3[ ][ ] = new int[ ] [ ]{ {10,20,25}, {30,40,45} };**

Масиви об'єктів:

- **MyClass arrayOfObject[ ] = new MyClass[2];**
  for (int i = 0; i < arrayOfObject.length; i++)
  arrayOfObject[i] = new MyClass();

Види специфікаторів доступу Java для елементів класу подані в табл. 3.

Таблиця 3

Специфікатори доступу до елементів класів

Тип елемента класу	← Джерело доступу				
	Цей же клас	Підклас із пакета	Звичайний клас пакета	Підклас з іншого пакета	Звичайний клас іншого пакета
private	Так	Ні	Ні	Ні	Ні
«friendly»	Так	Так	Так	Ні	Ні
protected	Так	Так	Так	Так	Ні
public	Так	Так	Так	Так	Так

Приклад програми з документаційними коментарями JavaDoc:

```
package arithmetic;
import java.io.*;

/**
 *  Клас для підтримки арифметичних дій
 * @author Викладач
 */
public class Calculator {
    // Назва фірми-постачальника (однаково для всіх калькуляторів)
    private static String VendorName = "Casio";
    // Перший операнд
    private double numberOne;
    // Другий операнд
    private double numberTwo;

    /** Конструктор без параметрів */
    Calculator() { numberOne = numberTwo = 0; }

    /**
     * Обчислює суму двох чисел
     * @return сума.
     */
    public double getSum() {
```

```
        return numberOne + numberTwo;
    }

    /** Установлює значення першого операнду
     * @param numberOne перший операнд.
     */
    void setNumberOne(double numberOne) {
        this.numberOne = numberOne;
    }

    /** Установлює значення другого операнду
     * @param numberTwo другий операнд.
     */
    void setNumberTwo(double numberTwo) {
        this.numberTwo = numberTwo;
    }

    /**
     * Статичний метод. Повертає   назву фірми-постачальника
     * @return назва фірми-постачальника.
     */
    public static String getVendorName() { return VendorName; }

    /** Головний метод програми */
    public static void main(String args[ ]) throws IOException {
        // Результат обчислень
        double result;
        // Створення об'єкта класу Calculator
        Calculator mashine = new Calculator();
        // Створення потоку введення, де System.in – стандартний
потік введення
        BufferedReader stdin = new BufferedReader(new
                                InputStreamReader(System.in));
        System.out.println("Введіть перше число: ");

mashine.setNumberOne(Double.parseDouble(stdin.readLine());
        System.out.println("Введіть друге число: ");
```

```
mashine.setNumberTwo(Double.parseDouble(stdin.readLine());
            result = mashine.getSum();
            // Виклик статичного методу getVendorName()
            System.out.println("Калькулятор" + Calculator.
getVendorName());
            System.out.println("Результат= " + result);
        }
    }
```

Порядок виконання лабораторної роботи

1. Ознайомлення з електронним підручником з Java – **The Java Tutorial**.

1.1. Відкрити файл **index.html**

1.2. Перейти по посиланнях **Getting Started** → **The "Hello World Application"**.

1.3. Вивчити структуру найпростішої Java-програми.

1.4. Повернутися на головну сторінку підручника.

1.5. Перейти по посиланню **Learning the Java Language** (вивчення мови Java) → **Language Basics** (основи мови).

1.6. Вивчити розділи **Variables** (змінні), **Operators** (операції), **Expressions**, **Statements and Blocks** (вираз, оператори й блоки), **Control Flow Statements** (оператори керування обчислювальним процесом).

2. Ознайомлення з електронною документацією по Java – **JDK 5.0 Documentation**.

2.1. Відкрити файл **index.html**

2.2. Переглянути зміст головної сторінки Java-документації.

2.3. Перейти на сторінку опису класів стандартної бібліотеки Java по посиланню **Java 2 Platform API Specification** розділу **API & Language Documentation** головної сторінки документації.

2.4. Ознайомитися з призначенням складових частин бібліотеки (таблиця **Java 2 Platform Packages**).

2.5. Перейти на сторінку опису класів пакета **java.lang** (по відповідному посиланню).

2.6. Ознайомитися з описом класів **Object**, **Math**, **String**.

3. Самостійна розробка простих Java-програм.

Загальні вимоги до програм:

наявність найпростішого текстового інтерфейсу користувача;

забезпечення коректної обробки вихідних даних з метою запобігання появі помилок при виконанні програми.

3.1. Використовуючи елементи класу **Math** (Java API) розробити програму, що обчислює та виводить на консоль значення одного з виразів:

Номер варіанта	Вираз
1	$a^{1/2}$
2	$\|a\|$
3	exp(a)
4	lg (a)
5	sin(a)
6	cos(a)
7	tg(a)
8	ctg(a)
9	a^b
10	перетворення a (радіани $\rightarrow$ градуси)
11	перетворення a (градуси $\rightarrow$ радіани)
12	ln(a)

Зазначені обчислювальні дії повинні бути реалізовані у відповідних статичних методах класу. Вихідні дані мають вводитися з консолі.

Примітка: *потрібно врахувати, що стандартні методи класу Math для обчислення тригонометричних функцій очікують параметр у радіанах, а вихідні дані й результат повинні бути подані в градусах.*

3.2. Перетворити попередню програму так, щоб вона могла приймати значення вихідних даних як з консолі, так і з параметра методу main (тобто **args[0]**).

Примітка: *параметр методу* main *становить рядок. Для перетворення рядка в число використовуйте метод parseDouble() стандартного класу* ***Double*** *Java API.*

3.3. Розробити програму для обробки відомості (див. варіанти завдань). Для цього необхідно описати клас, поля якого відповідають вихідним полям відомості. Для встановлення значень полів повинен

використовуватися конструктор. Обчислення значень розрахункових полів відомості, одержання значень вихідних полів повинне виконуватися за допомогою відповідних нестатичних методів класу.

Програма повинна забезпечувати створення масиву об'єктів цього класу, введення вихідних даних з консолі та виведення на консоль значень вихідних полів і полів, що розраховуються, для кожного із записів відомості, а також підсумкової інформації з відомості.

Варіант 1

Відомість нарахування зарплати співробітникам підприємства:

№ з/п	Прізвище	Зарплата, грн.	Утримано, грн.	Видано, грн.
1	F	Z	P	S = Z – P
2				
…				
	Разом	∑	∑	∑

Варіант 2

Відомість витрат палива на автобазах міста:

№ з/п	Автобаза	Витрачено палива, кг.	Кількість автомашин, шт.	Середня витрата палива, кг.
1	A	T	K	C = T/K
2				
…				
n				
	Разом	∑	∑	∑/n

Варіант 3

Відомість використання машинного часу в обчислювальному центрі:

№ з/п	Кафедра	Використання машинного часу, годин		Відхилення від плану	
		за планом	фактично	у годинах	у %
1	K	P	F	O_1 = P-F	$O_2 = O_1 \times 100/P$
2					
…					
	Разом	∑	∑		

Варіант 4

Відомість споживання електроенергії на заводах міста:

№ з/п	Завод	Споживання електроенергії, кВт/год		Відхилення від плану	
		за планом	фактично	у кВт/год.	у %
1	Z	P	F	O_1= P-F	O_2=O_1×100/P
2					
…					
	Разом	∑	∑		

Варіант 5

Відомість руху матеріалів на складі підприємства за звітний період:

№ з/п	Склад	Рух матеріалів за період, грн.			Залишок на кінець періоду
		залишок на початок періоду	отримано	видано	
1	C	O_c	P	V	R = O_c + P – V
2					
…					
	Разом	∑	∑	∑	∑

Варіант 6

Відомість прибутку підприємства за звітний період за видами продукції:

№ з/п	Продукція	Кількість, шт.	Оптова ціна, грн.	Собівартість, грн.	Прибуток, грн.
1	Pr	K	Z	C	P = K(Z – C)
2					
…					
	Разом	∑			∑

Варіант 7

Відомість відвідування занять студентами:

№ з/п	Прізвище	Пропущено, годин		Пропуск	
		усього	виправдано	у годинах	у %
1	F	V	O	P_1 = V – O	P_2 = P_1·× 100/V
2					
…					
	Разом:	∑	∑	∑	

Варіант 8

Відомість обсягу поставок продукції в натуральному та вартісному виразах:

№ з/п	Продукція	Шифр	Об'єм постачань, шт.	Оптова ціна, грн.	Об'єм постачань, грн.
1	P	H	V	Z	O = V × Z
2					
…					
	Разом		∑	∑	∑

Варіант 9

Відомість розрахунку середньої вартості перевезення авіапасажирів:

№ з/п	Тип літаку	Рейс	Витрати на рейс, грн.	Кількість пасажирів	Середня вартість перевезення, грн.
1	T	R	Z	K	S = Z/K
2					
…					
n					
	Разом:		∑	∑	∑/n

Варіант 10

Відомість обліку часу роботи верстатів підприємства:

№ з/п	Тип верстата	Час роботи, год.		Відхилення від плану	
		за планом	фактично	у годинах	у %
1	Z	P	F	O_1 = P-F	$O_2 = O_1 \times 100/P$
2					
…					
	Разом	∑	∑		

Варіант 11

Відомість випуску деталей робітниками цеху:

№ з/п	Прізвище	Кількість деталей, шт.		Брак	
		виготовлено	прийнято	шт.	%
1	Z	P	F	O_1 = P-F	$O_2=O_1 \times 100/P$
2					
…					
	Разом	∑	∑	∑	

Варіант 12

Відомість наявності та руху основних фондів підприємства:

№ з/п	Назва фонду	Наявність на початок року, шт.	Поступило, шт.	Вибуло, шт.	Наявність на конець року, шт.
1	F	N_1	P	V	$N_2 = N_1 + P - V$
2					
…					
	Разом	$\sum$	$\sum$	$\sum$	$\sum$

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Структура найпростішої Java-програми.
2. Порядок створення нового проекту консольного додатка в середовищі NetBeans.
3. Порядок відкриття існуючого проекту в середовищі NetBeans.
4. Порядок додавання файлу до проекту в середовищі NetBeans.
5. Порядок компіляції та запуску програми в середовищі NetBeans.
6. Порядок налагодження програми у середовищі NetBeans.
7. Порядок генерації електронної документації за програмою в середовищі NetBeans.
8. Порядок передачі параметра методу main у середовищі NetBeans.
9. Поняття класу та об'єкта, співвідношення між ними.
10. Принципи об’єктно-орієнтованого програмування.
11. Життєвий цикл об'єкта в програмі.
12. Призначення та варіанти використання посилання **this**.
13. Що означає перевантаження методу?
14. Призначення конструктора.
15. Основні особливості конструктора.

ЛАБОРАТОРНА РОБОТА 2
РОЗРОБКА ДОДАТКІВ З ЗАСТОСУВАННЯМ ПРИНЦИПІВ СПАДКУВАННЯ ТА ПОЛІМОРФІЗМУ

Мета лабораторної роботи:

1. Придбання практичних навичок з розробки класів мовою Java.
2. Придбання практичних навичок застосування механізмів спадкування й поліморфізму в мові Java.
3. Удосконалення навичок роботи з електронною документацією Java-розроблювача.

Перед виконанням лабораторної роботи студент повинен знати:

1. Поняття класу та об'єкта, співвідношення між ними.
2. Принципи об’єктно-орієнтованого програмування.
3. Синтаксис опису класу.
4. Життєвий цикл об'єкта в Java-програмі.
5. Порядок використання конструкторів.
6. Синтаксис спадкування в Java.
7. Порядок виклику конструкторів при спадкуванні.
8. Принципи перевантаження та перевизначення методів.
9. Принципи реалізації поліморфізму в Java.
10. Порядок використання абстрактних класів та інтерфейсів.

Після виконання лабораторної роботи студент повинен вміти:

1. Самостійно розробляти класи мовою Java.
2. Застосовувати механізми спадкування й поліморфізму при розробці програм мовою Java.

Реалізація принципів спадкування й поліморфізму в мові Java

Принцип спадкування:

ієрархічне відношення між класами, при якому клас використовує структуру або поведінку іншого класу;

при встановленні між класами відношення спадкування між ними також встановлюється залежність «приватне – загальне»;

класи-спадкоємці доповнюють або перевизначають структуру й поведінку класів-предків, які успадковані.

Приклад відношення спадкування подано на рис. 4.

Усі герої казки мають властивості та методи класу Hero.

Людина (Human) вміє читати книги, тварина (Animal) вміє полювати, жінка (Woman) - пекти пироги, а чоловік (Man) - рубати дерева.

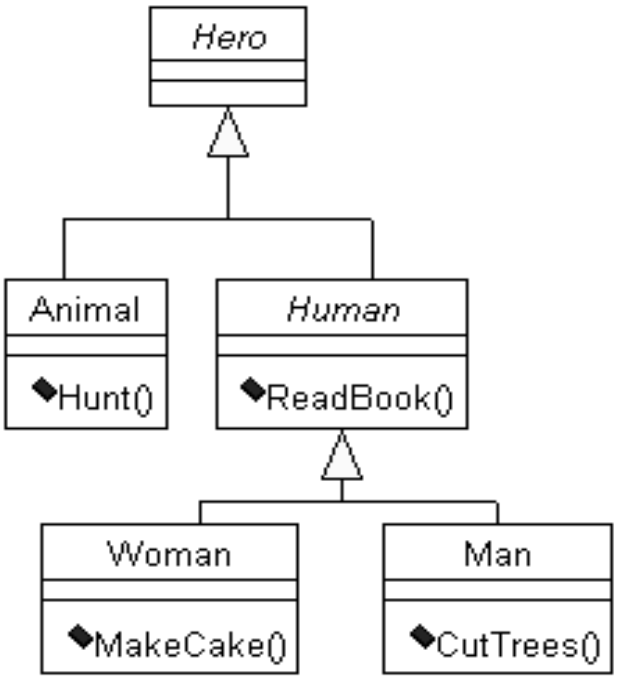


Рис. 4. **Діаграма спадкування**

Реалізація діаграми спадкування (рис. 4) мовою Java:

```
class Hero {...}
class Animal extends Hero {
  ……
   public void hunt() {...}
}
class Human extends Hero {
……
   public void readBook() {...}
}
class Woman extends Human {
……
   public void makeCake() {...}
}
```

```
class Man extends Human {
……….……….……….……
  public void cutTrees() {...}
}
```

Ініціалізація об'єктів при спадкуванні:

а) конструктори без параметрів

```
class Art {
 public Art() { System.out.println("Art");  }
}
class Drawing extends Art {
 public Drawing() { System.out.println("Drawing"); }
}
public class Cartoon extends Drawing {
 public Cartoon() {System.out.println("Cartoon"); }
 public static void main(String[ ] args) {
   Cartoon x = new Cartoon();
 }
}
```

Автоматичний виклик усіх конструкторів ієрархії: Cartoon() → Drawing() → Art().

Результат роботи програми:

Art
Drawing
Cartoon

б) конструктори з параметрами

```
class Box {
  private double width;
  private double height;
  private double depth;
  Box(double width, double height, double depth) {
    this.width = width;
    this.height = height;
    this.depth = depth;
```

```
  }
class WeightBox extends Box {
  private double weight;
  WeightBox(double width, double height, double depth, double
weight) {
    // Виклик конструктора найближчого базового класу
    super (width, height, depth);
    this.weight = weight;
  } }
public class BoxManager {
  public static void main(String[ ] args) {
    WeightBox box = new WeightBox(2.0, 3.0, 5.0, 125.5);
  }
}
```

Виклик усіх конструкторів ієрархії: WeightBox() → Box(). Конструктор Box() явно викликається з конструктора класу WeightBox за допомогою ключового слова ***super.***

Основні варіанти використання спадкування:

«B – це A»

```
class A {    // базовий клас
  private int field;
  public void f() { ... }
}
class B extends A { }
```

«B схожий на A»

```
class A {   // базовий клас
  private int field1;
  public void f() { ... }
}
class B extends A {
   private int field2;
   public void g() { ... }
}
```

перевантаження методу базового класу

```
class A {   // базовий клас
        private int field1;
        public void f() { ... }
}
class B extends A {
        private int field2;
        public void f(int argument) { ... }
}
```

перевизначення методу базового класу

```
class A {   // базовий клас
  private int field1;
  public void f() {...}
}
class B extends A {
  private int field2;
  public void f() {
     ...      // Наприклад, виклик методу базового класу
     super.f();
     ...
  } }
```

Заборона спадкування за допомогою ключового слова **final** на прикладі класу Java SE **String**:

public ***final*** **class String**
extends Object
implements Serializable**,** Comparable**<**String**>,** CharSequence **{...}**

Поліморфізм – здатність об'єктів з однієї ієрархії класів відповідати на один запит (рис. 5).

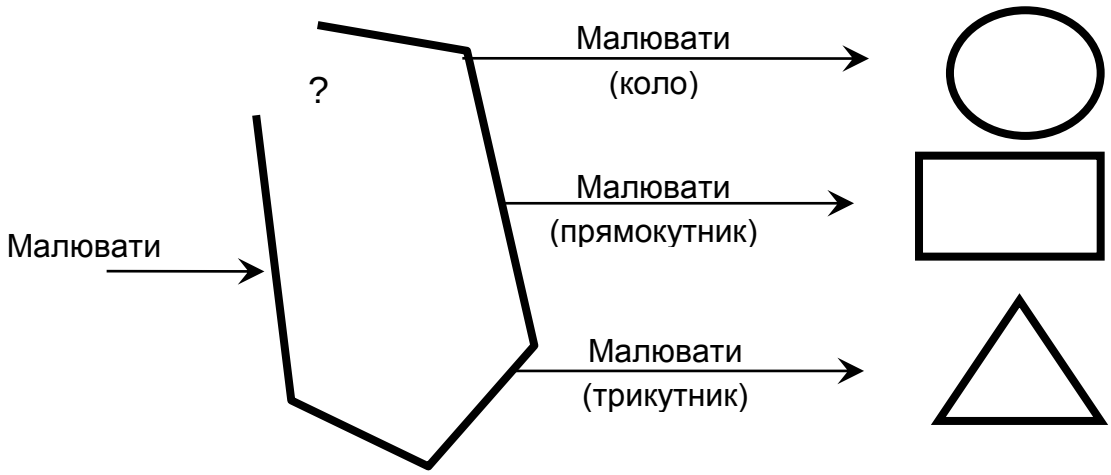


Рис. 5. **Приклад поліморфізму**

Використання принципу поліморфізму подано на рис. 6.

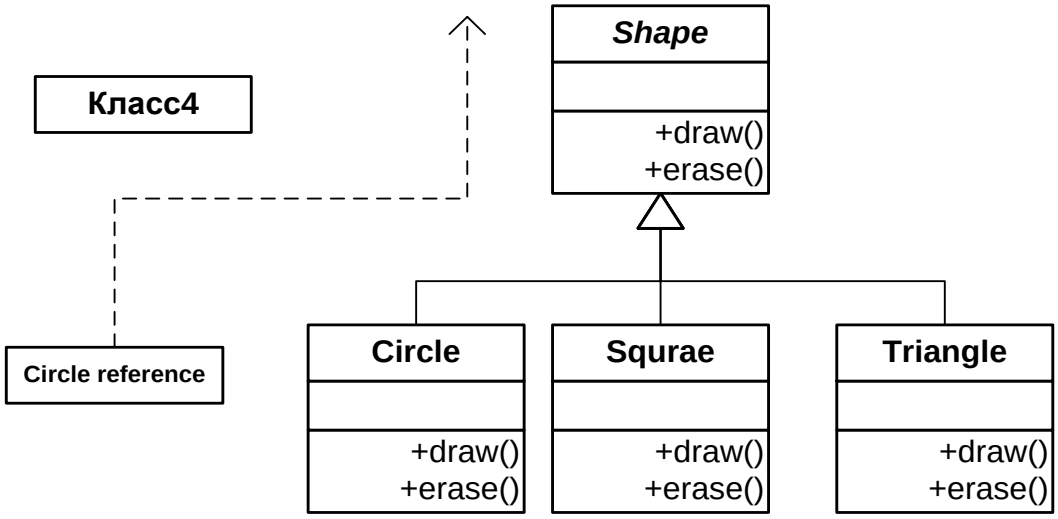


Рис. 6. **Діаграма класів – геометричних фігур**

Абстрактні класи становлять поняття предметної області, наділені всіма найбільш загальними властивостями та/або поведінкою, котрі передбачається перевизначати в нащадках (похідних класах).

Правила використання абстрактних класів:

реалізують вершину в ієрархії спадкування;

неможливо створити їхні об'єкти;

можливо оголосити посилання на абстрактний клас, що згодом буде використовуватися для роботи з об'єктами похідних класів;

можуть містити абстрактні й звичайні методи;

абстрактні методи не мають потреби в реалізації;

усі абстрактні методи повинні бути перевизначені в похідному класі або він має бути теж оголошений абстрактним.

Реалізація поліморфної поведінки з абстрактним класом:

```
abstract class Shape { abstract void draw(); }
class Circle extends Shape {
  void draw() { System.out.println("Circle.draw()"); }
}
class Square extends Shape {
  void draw() { System.out.println("Square.draw()"); }
  public static void main(String[ ] args) {
     //Shape s = new Shape();  // Помилка
     Shape s; // Немає помилки
     s = new Circle(); s. draw(); // викликається draw() з Circle
     s = new Square(); s. draw(); // викликається draw() з Square
  }
}
```

Інтерфейс (interface) – це іменований набір семантично зв'язаних абстрактних елементів. Інтерфейси використовуються для реалізації моделі поводження об'єкта.

Правила використання інтерфейсів:

реалізують вершину ієрархії спадкування;

неможливо створити їх об'єкти;

можливо оголосити посилання на інтерфейс, що згодом буде використовуватися для роботи з об'єктами класів реалізації;

елементи інтерфейсу неявно є public;

поля інтерфейсу автоматично є static і final;

можуть містити тільки абстрактні методи;

всі абстрактні методи повинні бути перевизначені в класах реалізації;

кожний член інтерфейсу (властивість або метод) автоматично є абстрактним;

клас може реалізовувати необмежену кількість інтерфейсів.

Реалізація поліморфної поведінки з інтерфейсом:

```
interface Shape { void draw(); }
class Circle implements Shape {
```

```
   void draw() { System.out.println("Circle.draw()"); }
}
class Square implements Shape {
  void draw() {System.out.println("Square.draw()");}
  public static void main(String[ ] args) {
     //Shape s = new Shape();  // Помилка
     Shape s; // Немає помилки
     s = new Circle(); s. draw(); // викликається draw() з Circle
     s = new Square(); s. draw(); // викликається draw() з Square
  } }
```

Порядок виконання лабораторної роботи

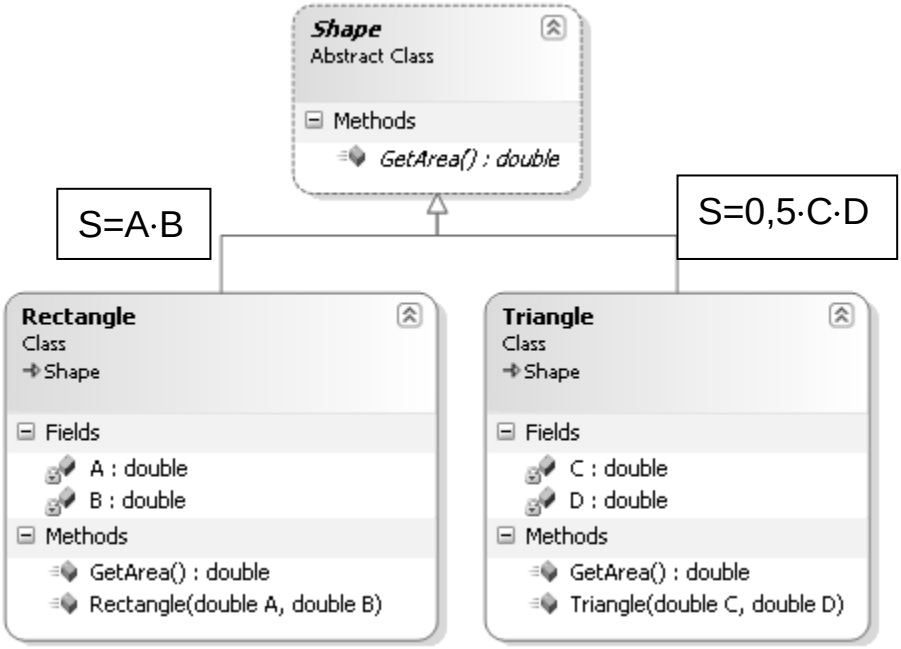
1. Розробити програму, що використовує принцип поліморфізму, на базі абстрактного класу відповідно до одного з варіантів завдань.
2. Перетворити розроблену програму так, щоб замість абстрактного класу використовувався інтерфейс.
3. У звіті навести тексти двох програм.

Варіант 1

Є діаграма класів: геометрична фігура (Shape) – абстрактний базовий клас, прямокутник (Rectangle) та прямокутний трикутник (Triangle) – похідні класи.

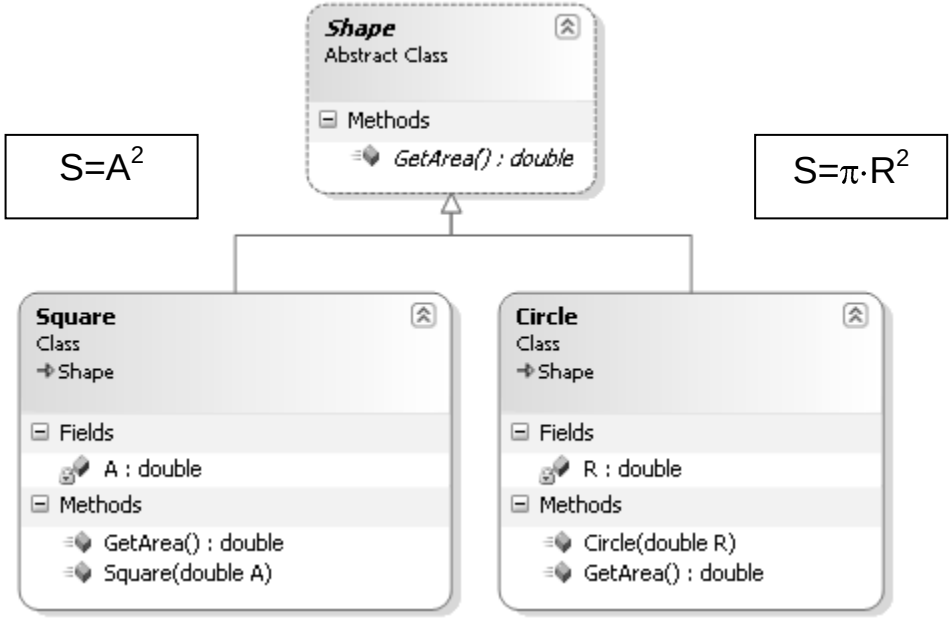
Клас Shape має абстрактний метод GetArea(), який обчислює та повертає значення площі геометричної фігури. Клас Rectangle має поля A – довжина прямокутника та B – ширина прямокутника. Клас Triangle – поля 3, D – катети прямокутного трикутника. Кожний з похідних класів перевизначає метод GetArea() базового класу.

Розробити консольну програму, яка використовує принцип поліморфізму при обчисленні площ прямокутника та прямокутного трикутника. Значення довжини і ширини прямокутника, катетів прямокутного трикутника ввести з консолі. Вивести значення площ прямокутника та прямокутного трикутника на консоль.



Варіант 2

Є діаграма класів: геометрична фігура (Shape) – абстрактний базовий клас, квадрат (Square) та коло (Circle) – похідні класи:



Клас Shape має абстрактний метод GetArea(), який обчислює та повертає значення площі геометричної фігури. Клас Square має поле A – довжина сторони квадрата. Клас Circle – поле R – радіус кола. Кожний з похідних класів перевизначає метод GetArea() базового класу.

Розробити консольну програму, яка використовує принцип поліморфізму при обчисленні площ квадрата та кола. Значення довжини

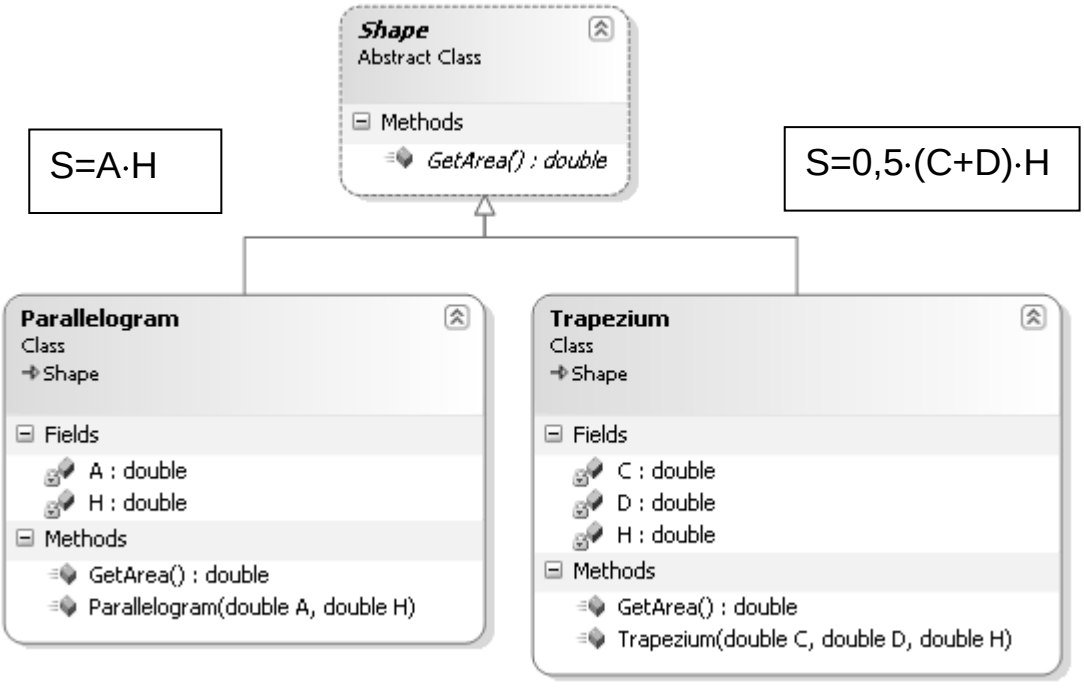
сторони квадрата, радіуса кола ввести з консолі. Вивести значення площ квадрата та кола на консоль.

Варіант 3

Є діаграма класів: геометрична фігура (Shape) – абстрактний базовий клас, паралелограм (Parallelogram) та рівнобедрена трапеція (Trapezium) – похідні класи.

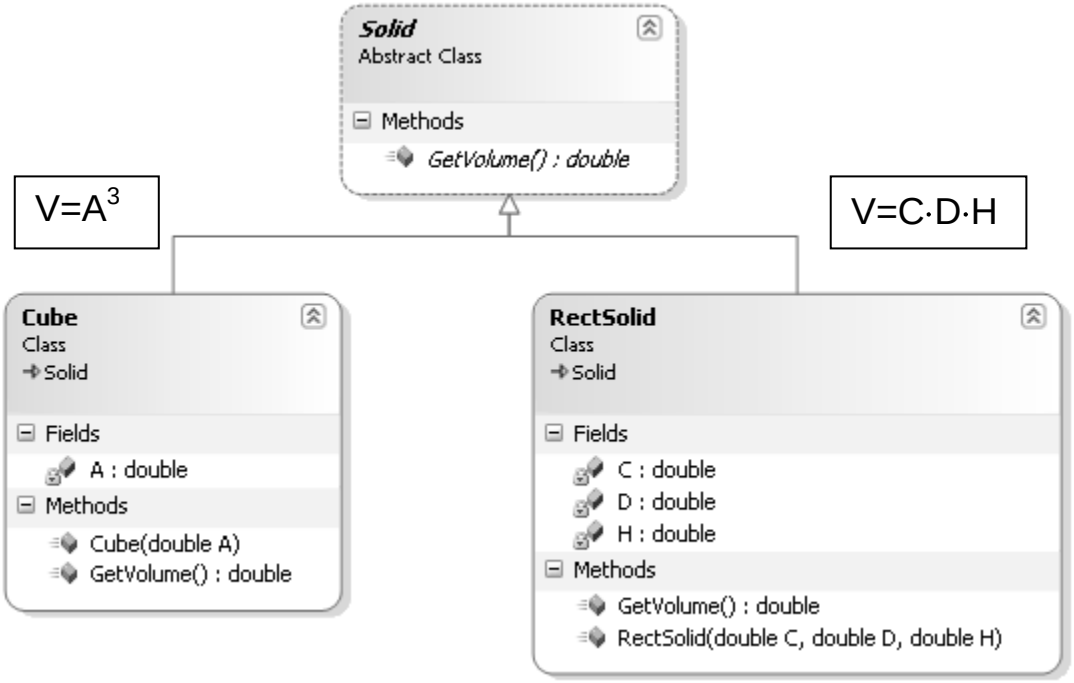
Клас Shape має абстрактний метод GetArea(), який обчислює та повертає значення площі геометричної фігури. Клас Parallelogram має поля A – основа паралелограма та H – висота паралелограма. Клас Trapezium – поля 3, D – основи трапеції та H – висота трапеції. Кожний з похідних класів перевизначає метод GetArea() базового класу.

Розробити консольну програму, яка використовує принцип поліморфізму при обчисленні площ паралелограма та трапеції. Значення основи і висоти паралелограма, основ та висоти трапеції ввести з консолі. Вивести значення площ паралелограма і трапеції на консоль.



Варіант 4

Є діаграма класів: геометричне тіло (Solid) – абстрактний базовий клас, прямокутний паралелепіпед (RectSolid) та куб (Cube) – похідні класи:



Клас Solid має абстрактний метод GetVolume(), який обчислює та повертає значення об'єму геометричного тіла. Клас Cube має поле A – довжина ребра куба. Клас RectSolid – поля 3, D – довжина та ширина основи прямокутного паралелепіпеда і H – його висота. Кожний з похідних класів перевизначає метод GetVolume() базового класу.

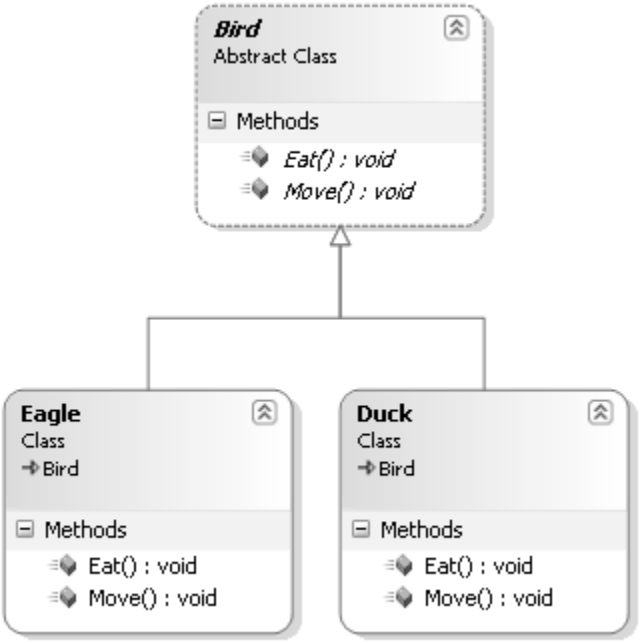
Розробити консольну програму, яка використовує принцип поліморфізму при обчисленні об'ємів куба та прямокутного паралелепіпеда. Значення довжини ребра куба; довжини і ширини основи прямокутного паралелепіпеда, його висоти ввести з консолі. Вивести значення об'ємів куба та прямокутного паралелепіпеда на консоль.

Варіант 5

Є діаграма класів: птиця (Bird) – абстрактний базовий клас, орел (Eagle) та качка (Duck) – похідні класи.

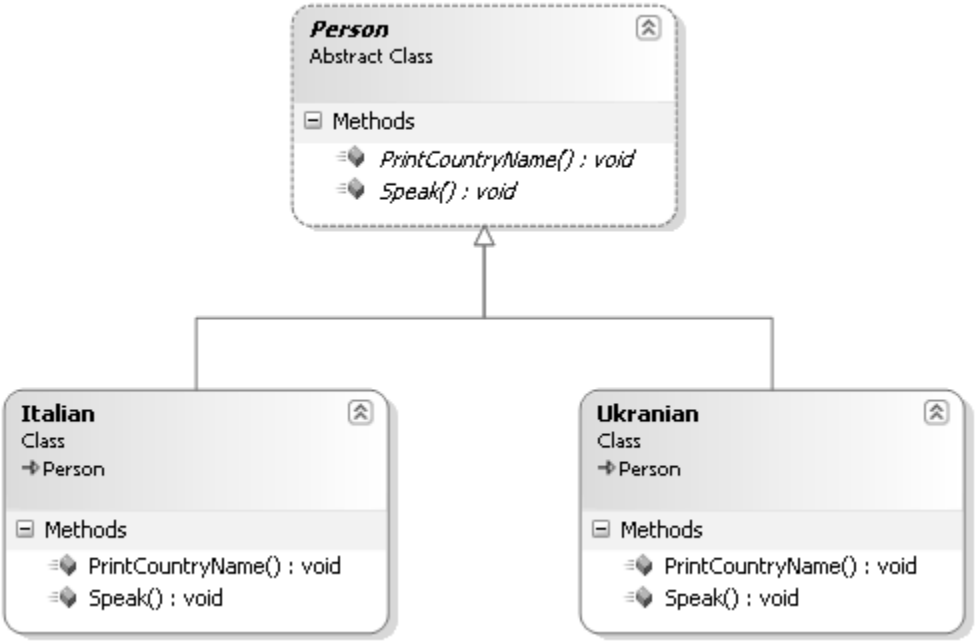
Клас Bird має абстрактні методи Eat() (їсти) та Move() (рухатися). Кожний з похідних класів перевизначає методи Eat(), Move() базового класу.

Розробити консольну програму, яка використовує принцип поліморфізму для виведення текстових повідомлень щодо руху та прийому їжі орлом і качкою.



Варіант 6

Є діаграма класів: персона (Person) – абстрактний базовий клас, італієць (Italian) та українець (Ukranian) – похідні класи.



Клас Person має абстрактні методи PrintCountryName() (надрукувати назву держави) та Speak() (говорити деякою мовою). Кожний з похідних класів перевизначає методи PrintCountryName(), Speak() базового класу.

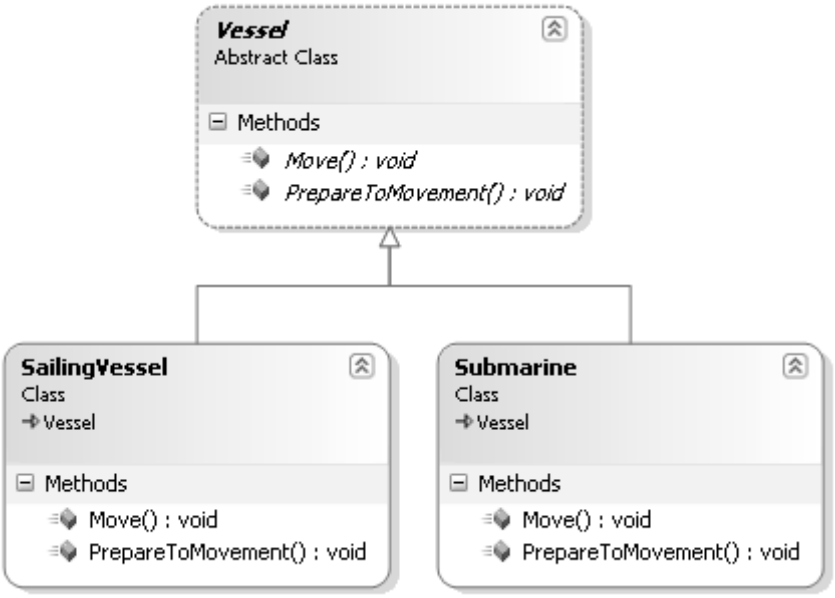
Розробити консольну програму, яка використовує принцип поліморфізму для виведення текстових повідомлень щодо назви держави та рідної мови італійця й українця.

Варіант 7

Є діаграма класів: судно (Vessel) – абстрактний базовий клас, вітрильник (SailingVessel) та підводний човен (Submarine) – похідні класи.

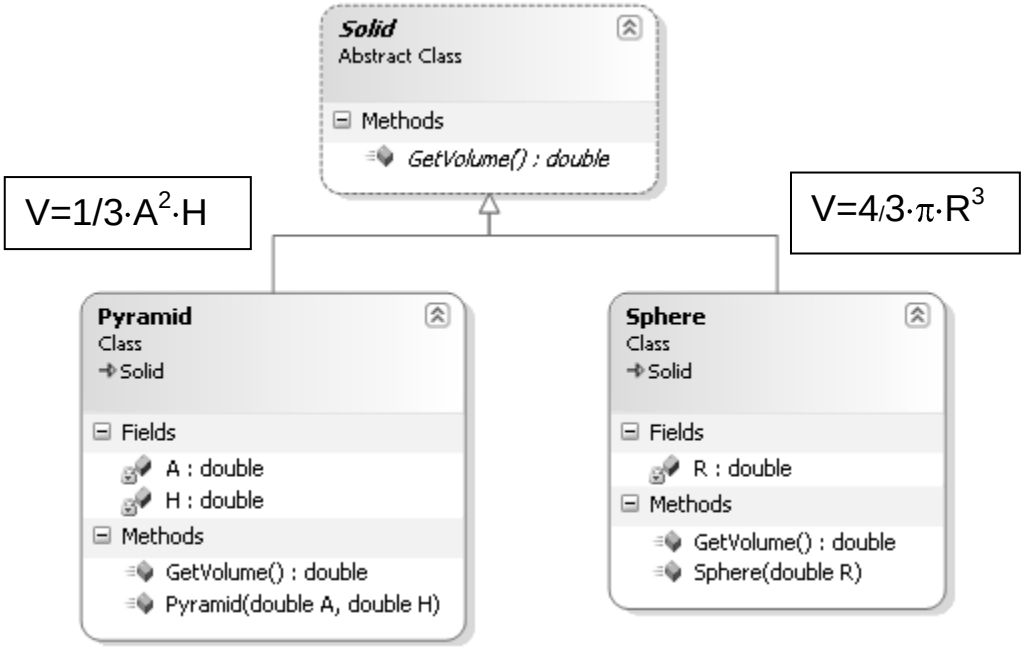
Клас Vessel має абстрактні методи PrepareToMovement() (підготуватися до руху) та Move() (рухатися). Кожний з похідних класів перевизначає методи PrepareToMoving() і Move() базового класу.

Розробити консольну програму, яка використовує принцип поліморфізму для виведення текстових повідомлень щодо підготовки до руху та руху вітрильника й підводного човна.



Варіант 8

Є діаграма класів: геометричне тіло (Solid) – абстрактний базовий клас, піраміда з квадратною основою (Pyramid) та куля (Sphere) – похідні класи.



Клас Solid має абстрактний метод GetVolume(), який обчислює та повертає значення об'єму геометричного тіла. Клас Pyramid має поля A – довжина сторони основи піраміди та H – висота піраміди. Клас Sphere – поле R – радіус кулі. Кожний з похідних класів перевизначає метод GetVolume() базового класу.

Розробити консольну програму, яка використовує принцип поліморфізму при обчисленні об'ємів піраміди та кулі. Значення довжини сторони основи піраміди, висоти піраміди і радіуса кулі ввести з консолі. Вивести значення об'ємів піраміди та кулі на консоль.

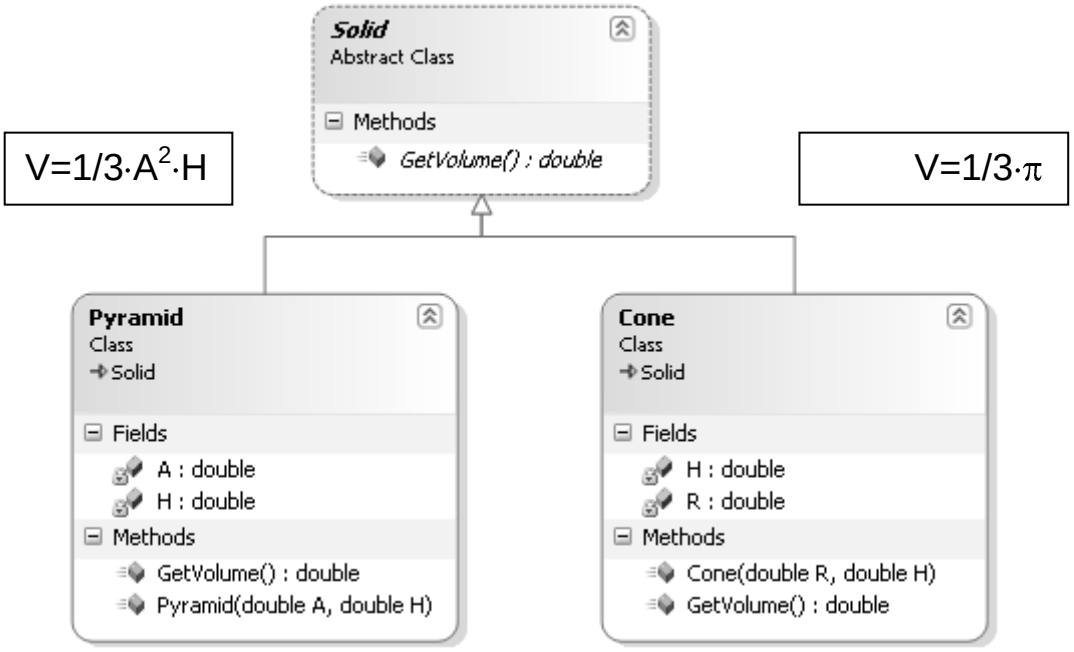
Варіант 9

Є діаграма класів: геометричне тіло (Solid) – абстрактний базовий клас, піраміда з квадратною основою (Pyramid) та конус (Cone) – похідні класи.

Клас Solid має абстрактний метод GetVolume(), який обчислює та повертає значення об'єму геометричного тіла. Клас Pyramid має поля A, – довжина сторони основи піраміди та H – висота піраміди. Клас Cone – поле R – радіус основи конуса і H – висота конуса. Кожний з похідних класів перевизначає метод GetVolume() базового класу.

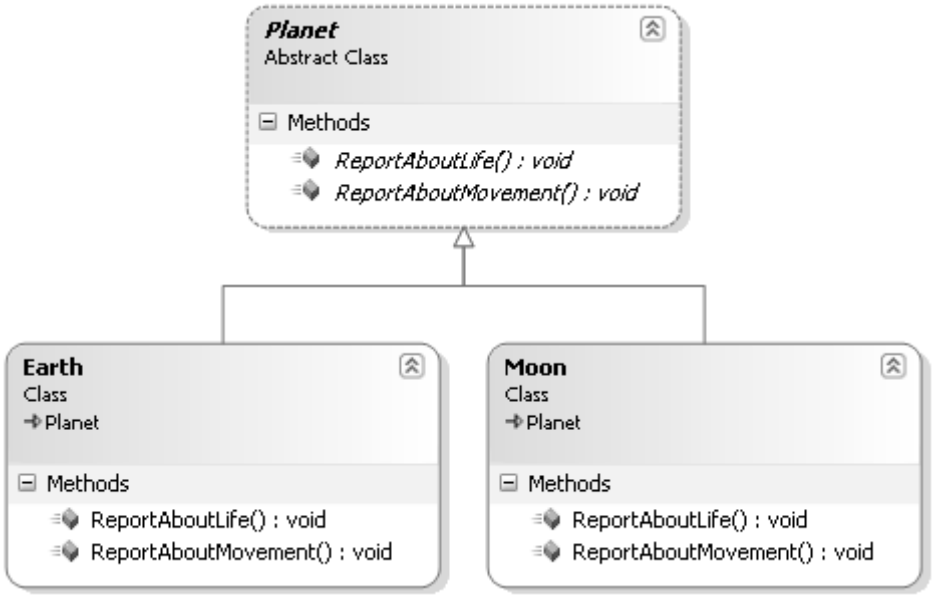
Розробити консольну програму, яка використовує принцип поліморфізму при обчисленні об'ємів піраміди та конуса. Значення довжини сторони основи піраміди і висоти піраміди, радіуса основи

конуса та висоти конуса ввести з консолі. Вивести значення об'ємів піраміди і конуса на консоль.



Варіант 10

Є діаграма класів: планета (Planet) – абстрактний базовий клас, Земля (Earth) та Місяць (Moon) – похідні класи:



Клас Planet має абстрактні методи ReportAboutMovement() (повідомити навколо якого небесного тіла рухається планета) та ReportAboutLife() (повідомити про наявність життя на планеті). Кожний з

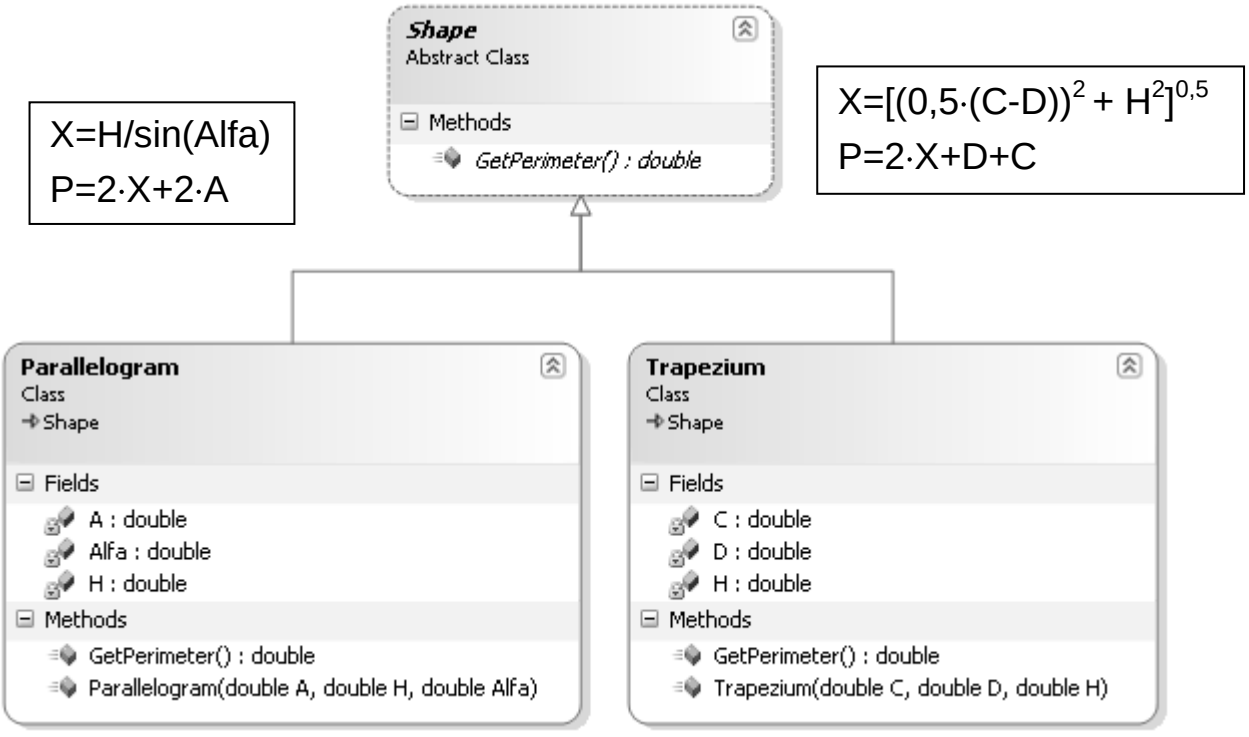
похідних класів перевизначає методи ReportAboutMovement() і ReportAboutLife() базового класу.

Розробити консольну програму, яка використовує принцип поліморфізму для виведення текстових повідомлень щодо руху та наявності життя на Землі і Місяці.

Варіант 11

Є діаграма класів: геометрична фігура (Shape) – абстрактний базовий клас, паралелограм (Parallelogram) та рівнобедрена трапеція (Trapezium) – похідні класи.

Клас Shape має абстрактний метод GetPerimeter(), який обчислює та повертає значення периметру геометричної фігури. Клас Parallelogram має поля A – основа паралелограма, H – висота паралелограма та Alfa – кут між основою й боковою стороною паралелограма. Клас Trapezium – поля 3, D – основи трапеції та H – висота трапеції. Кожний з похідних класів перевизначає метод GetPerimeter() базового класу.



Розробити консольну програму, яка використовує принцип поліморфізму при обчисленні периметрів паралелограма та трапеції. Значення основи і висоти паралелограма, основ та висоти трапеції

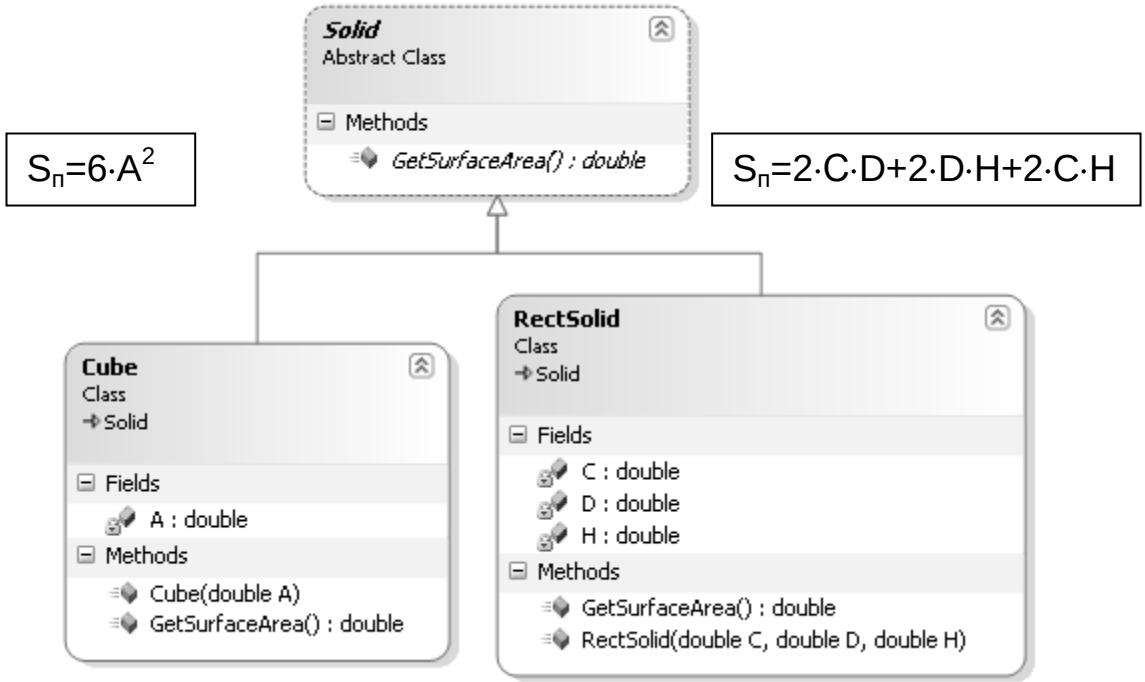
ввести з консолі. Вивести значення периметрів паралелограма і трапеції на консоль.

Варіант 12

Є діаграма класів: геометричне тіло (Solid) – абстрактний базовий клас, прямокутний паралелепіпед (RectSolid) та куб (Cube) – похідні класи.

Клас Solid має абстрактний метод GetSurfaceArea(), який обчислює та повертає значення площі поверхні геометричного тіла. Клас Cube має поле A – довжина ребра куба. Клас RectSolid має поля 3, D – довжина та ширина основи прямокутного паралелепіпеда і H – його висота. Кожний з похідних класів перевизначає метод GetSurfaceArea() базового класу.

Розробити консольну програму, яка використовує принцип поліморфізму при обчисленні площ поверхонь куба та прямокутного паралелепіпеда. Значення довжини ребра куба; довжини та ширини основи прямокутного паралелепіпеда, його висоти ввести з консолі. Вивести значення площ поверхонь куба та прямокутного паралелепіпеда на консоль.



КОНТРОЛЬНІ ЗАПИТАННЯ

1. Пояснити визначення методу: *public static void main(String[ ] args) { }*
2. Принципи об’єктно-орієнтованого підходу.
3. Синтаксис опису класу.
4. Особливості статичних елементів класу.
5. Специфікатори доступу в Java.
6. Порядок ініціалізації об'єкта класу.
7. Агрегація й спадкування.
8. Синтаксис спадкування в Java.
9. Порядок виклику конструкторів при спадкуванні.
10. Перевантаження «базового» методу.
11. Перевизначення «базового» методу.
12. Поліморфізм у Java.
13. Переваги концепції поліморфізму.
14. Абстрактні класи.
15. Абстрактні класи – правила.
16. Інтерфейси.
17. Інтерфейси – правила.

ЛАБОРАТОРНА РОБОТА 3
ВИКОРИСТАННЯ ОСНОВНИХ БІБЛІОТЕК JAVA SE ПРИ РОЗРОБЦІ ДОДАТКІВ

Мета лабораторної роботи:

1. Удосконалення практичних навичок з розробки класів мовою Java.

2. Придбання практичних навичок використання бібліотечних класів виключень, введення-виведення, підтримки багатопотоковості, контейнерів Java SE.

3. Удосконалення навичок роботи з електронною документацією Java-розроблювача.

Перед виконанням лабораторної роботи студент повинен знати:

1. Організацію системи введення-виведення в Java SE.
2. Принципи обробки виключень в Java-програмі.
3. Принципи розробки багатопотокових програм.
4. Структуру бібліотеки контейнерів Java SE і принципи їх використання.

Після виконання лабораторної роботи студент повинен вміти:

1. Самостійно розробляти Java-програми з використанням основних бібліотек Java SE.

Загальні відомості про використання основних бібліотек Java SE

Проблеми «традиційного підходу» до обробки помилок під час виконання програми:

1. «Помилки можуть траплятися з іншими, але не в *моєму* коді».
2. «Перевірочний» код повинен перебувати всередині «корисного» коду.
3. Код із численними перевірками помилок легко стає нечитабельним.

Переваги обробки виключень у порівнянні з «традиційним підходом» до обробки помилок:

1. Відділення «корисного» коду від «перевірочного».
2. Групування типів помилок.
3. Можливість передачі помилки в інший контекст.

Послідовність обробки виключення подана на рис. 7:

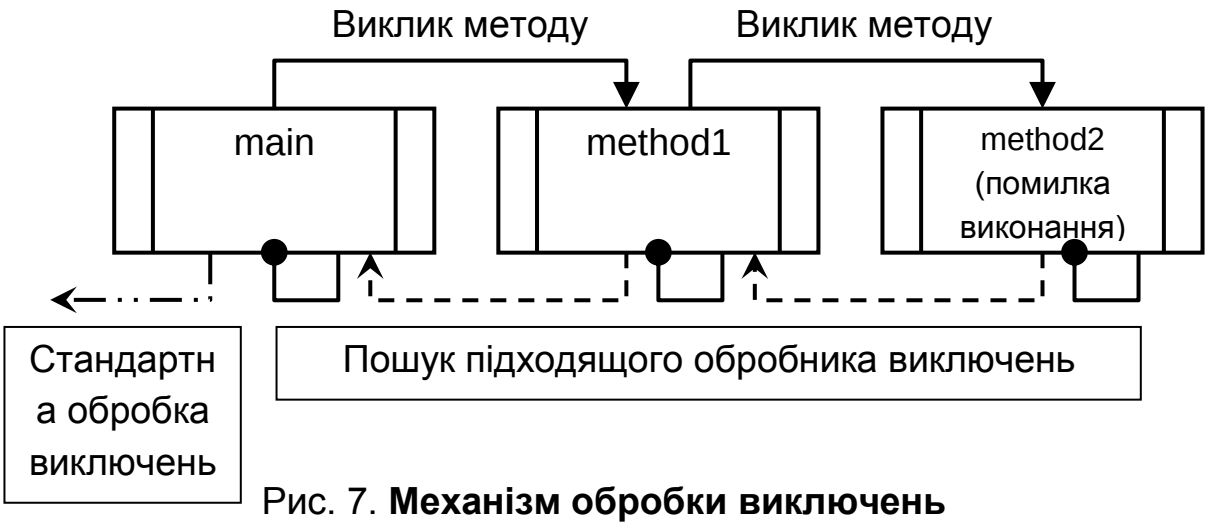


Рис. 7. **Механізм обробки виключень**

Синтаксис обробки виключень:

```
try {
    // блок, у якому може виникнути помилка \
}
catch(ExceptionType1 ex) {
    //блок обробки виключення типу ExceptionType1
}
……..
catch(ExceptionType ex) {
    // блок обробки виключення типу ExceptionType
}
// необов'язковий блок
finally {
// оператори, які виконуються в будь-якому разі
}
```

Ієрархія класів системи обробки виключень Java SE наведена на рис. 8.

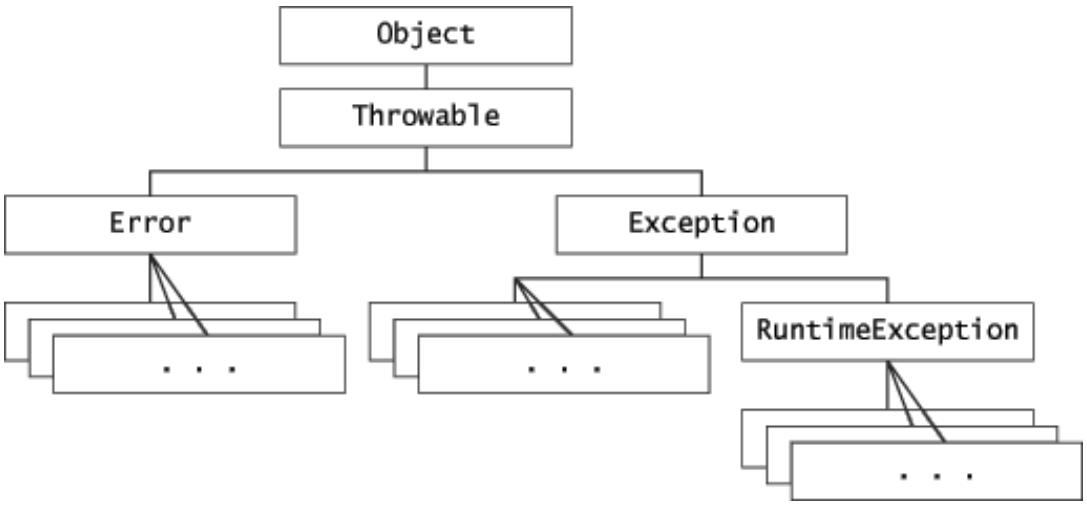


Рис. 8. **Базові класи системи обробки виключень Java SE**

Ключове слово **throws** – передача об'єкта виключення в інший контекст:

public void inputFile(String fName)
***throws* FileNotFoundException { . . . }**

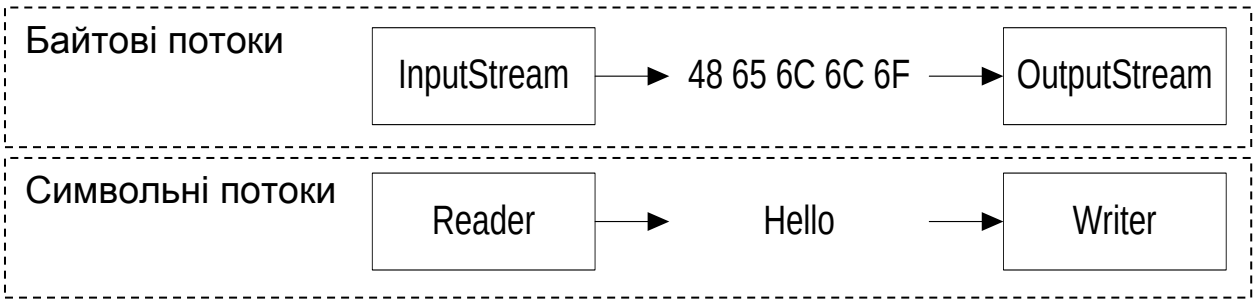


Рис. 9. **Базові класи потоків введення-виведення Java SE**

Класи файлового введення-виведення:
FileInputStream – байтовий потік введення з файлу;
FileOutputStream – байтовий потік виведення до файлу;
FileReader – символьний потік введення з файлу;
FileWriter – символьний потік виведення до файлу.
Класи-мости:
InputStreamReader – читає байти з **InputStream** і перетворює їх у символи, використовуючи зазначене кодування;
OutputStreamWriter – перетворює символи в байти, використовуючи зазначене кодування, і записує їх в **OutputStream**.

Приклад використання класів-мостів:

Байти → *Символи* (символьний буферизований потік введення з консолі)

```
BufferedReader in = new BufferedReader(new
                                    InputStreamReader(System.in));
```

Символи ← *Байти* (символьний буферизований потік виводу до файлу в заданому кодуванні)

```
Writer destination = new BufferedWriter(new OutputStreamWriter(
            new FileOutputStream(outputFile), encoding));
```

Приклади файлового введення-виведення:

посимвольне введення-виведення файлів

```
File inputFile = new File(“in.txt");
File outputFile = new File("out.txt");
FileReader in = new FileReader(inputFile);
FileWriter out = new FileWriter(outputFile);
int c;
while ((c = in.read()) != -1)
    out.write(c);
in.close();
out.close();
```

порядкове читання файлу:

```
BufferedReader in = new BufferedReader(new
                                FileReader("IOStreamDemo.java"));
String s;
while((s = in.readLine()) != null)
   System.out.println(s);
in.close();
```

Склад бібліотеки контейнерів Java SE:

Інтерфейси – забезпечують уніфікацію доступу до елементів незалежно від реалізації контейнера.

Класи реалізації – в основному це структури даних.

Ітератори – забезпечують послідовний доступ до елементів контейнера.

Алгоритми – методи, що реалізують розповсюджені алгоритми обробки даних.

Основні інтерфейси бібліотеки контейнерів Java SE, наведені на рис. 10.

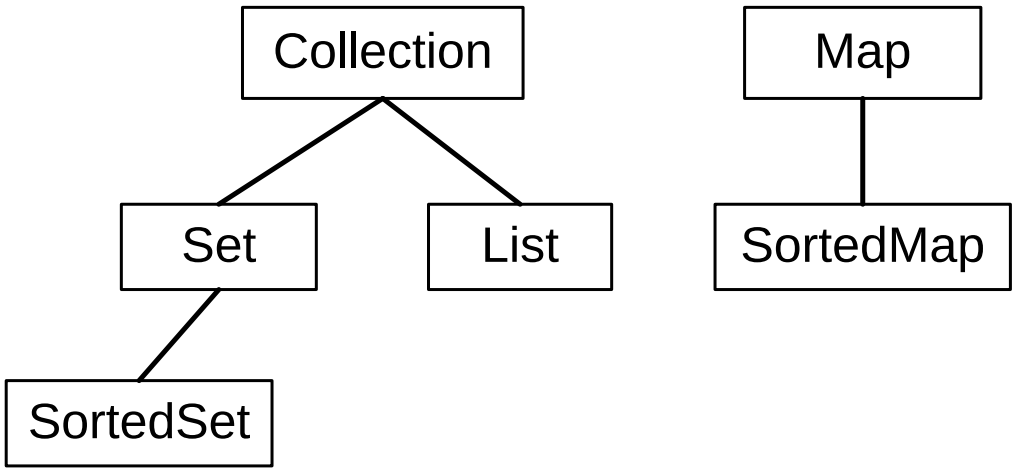


Рис. 10. **Основні інтерфейси бібліотеки контейнерів Java SE**

Коротка характеристика інтерфейсів бібліотеки контейнерів Java SE:

Collection – «кореневий» інтерфейс;

Set – «множина» (однакові елементи неприпустимі);

SortedSet – множина, елементи якої впорядковані по зростанню;

List – «список»;

Map – «асоціативний масив» (таблиця «ключ-значення»);

SortedMap – таблиця, ключі якої впорядковані по зростанню.

Клас **ArrayList** – динамічний масив:

для зберігання елементів використовується масив;

швидкі операції проходу за всім списком (ітерації) і одержання даних;

конструктори:

ArrayList() – порожній масив, **initialCapacity = 10**.

ArrayList(int initialCapacity) – порожній масив з довільної initialCapacity.

ArrayList(Collection c) – масив, який ініціалізований елементами колекції **c**.

Клас **LinkedList** – список:

для зберігання елементів використовується двозв’язний список;

швидкі операції додавання елементів у початок (кінець) списку, проходу за всім списком (ітерації), видалення елементів із середини списку (постійний час);

конструктори:

LinkedList() – порожній список.

LinkedList(Collection c) – список, ініціалізований елементами колекції **c**.

Клас **HashSet** – множина:

для зберігання елементів використовується хеш-таблиця;

постійний час виконання основних операцій з елементами;

конструктори:

HashSet() – порожня множина, initialCapacity = 16, loadFactor = 0,75.

HashSet(Collection c) – множина, ініціалізована елементами колекції c, loadFactor = 0,75.

HashSet(int initialCapacity) – порожня множина, initialCapacity = …, loadFactor = 0,75...

HashSet(int initialCapacity, float loadFactor) - порожня множина, initialCapacity = … , loadFactor = …

Клас **TreeSet** – множина:

для зберігання елементів використовується бінарне дерево;

дані впорядковані, максимальне число кроків при пошуку по дереву дорівнює висоті даного дерева;

конструктори:

TreeSet() – порожня множина.

TreeSet(Collection c) – множина, ініціалізована елементами колекції **c.**

TreeSet(Comparator c) – порожня множина, відсортована відповідно до компаратора.

TreeSet(SortedSet s) – множини, що містять ті ж елементи, що й **s**.

Асоціативні масиви:

містять пари ключ-значення;

і ключ, і значення є об'єктами;

ключі повинні бути унікальними;

класи реалізації:

1. **HashMap**

2. **TreeMap**

внутрішні структури даних аналогічні відповідним реалізаціям множин;

конструктори аналогічні відповідним реалізаціям множин.

Ітератори:

ітератор – це допоміжний об'єкт, який використовується для проходу по колекції об'єктів;

ітератори базуються на інтерфейсі *Iterator (ListIterator);*

будь-який ітератор, як би він не був улаштований, має наступні три методи:

boolean hasNext() – перевіряє чи є ще елементи в колекції.

Object next() – видає черговий елемент колекції.

void remove() – видаляє останній обраний елемент із колекції.

ListIterator має додаткові методи:

boolean hasPrevious() – перевіряє чи є ще елементи в колекції (у зворотному напрямку).

Object previous() – видає попередній елемент колекції.

Використання ітератора з **ArrayList**:

```
ArrayList al = new ArrayList();
al.add("C");
al.add("A");
al.add("E");
al.add("B");
al.add("D");
al.add("F");
System.out.print("Original contents of al: ");
 Iterator itr = al.iterator();
 while(itr.hasNext()) {
    Object element = itr.next();
    System.out.print(element + " ");
 }
```

Використання ітератора з **HashMap**:

```
HashMap hm = new HashMap();
hm.put("John Doe", new Double(3434.34));
hm.put("Tom Smith", new Double(123.22));
```

```
hm.put("Jane Baker", new Double(1378.00));
Set set = hm.entrySet();
 Iterator i = set.iterator();
 while(i.hasNext()) {
     Map.Entry me = (Map.Entry)i.next();
     System.out.print(me.getKey() + ": ");
     System.out.println(me.getValue());
}
double balance =  ((Double)hm.get("John Doe")).doubleValue();
```

«Параметризований» контейнер для зберігання рядків:

```
ArrayList <String> list = new ArrayList<String>();
list.add("one");
list.add("two");
list.add("three");
list.add("four");
Iterator<String> itr = list.iterator();
while(itr.hasNext()) {
   String str = itr.next();
   System.out.println(str + " is " + str.length() + " chars long.");
}
```

Цикл **for...each** замість ітератора в «параметризованому» контейнері:

```
ArrayList<String> list = new ArrayList<String>();
list.add("one");
list.add("two");
list.add("three");
list.add("four");
for (String str:list)
  System.out.println(str + " is " + str.length() + " chars long.");
```

Приклад використання алгоритмів (методів класу **Collections**):

```
LinkedList list = new LinkedList();
list.add(new Integer(-8));
list.add(new Integer(20));
```

```
list.add(new Integer(-20));
list.add(new Integer(8));
// Створення компаратора для сортування у зворотному порядку
Comparator r = Collections.reverseOrder();
// Сортування списку з використанням компаратора
Collections.sort(list, r);
Collections.shuffle(list);
// Мінімальний елемент списку
System.out.println("Minimum: " + Collections.min(list));
// Максимальний елемент списку
System.out.println("Maximum: “ + Collections.max(list));
```

Клас java.util.Random – генератор випадкових чисел який використовується для одержання послідовності псевдовипадкових чисел.

У класі Random визначено кілька методів, які повертають псевдовипадкові величини для примітивних типів Java.

```
public class Test {
  public static void main(String[ ] args) {
    Random r = new Random();
    // Генерація восьми псевдовипадкових цілих чисел
    for (int i =0; i < 9; i++)
      System.out.println(r.nextInt() + " ");
    }
}
```

Основні поняття багатозадачності:

процес – об'єкт, що створюється операційною системою при запуску додатка;

процес – екземпляр додатка, завантажений в ОЗП;

процес має віртуальний адресний простір; код, який виконується; дані;

при створенні процесу автоматично стартує його первинний потік – послідовність команд для виконання;

первинний потік може створювати додаткові потоки, використовуючи API операційної системи.

Стани потоку виконання:

1. Готовність до виконання.
2. Виконання.
3. Пауза.
4. Поновлення.
5. Блокування.
6. Виконання завершене.

Потоки виконання в Java-програмі:

потік у Java представляється класом **Thread**:

public class Thread extends Object **implements** Runnable

клас **Thread** має поля, що відповідають мінімальному, максимальному й нормальному пріоритетам потоку;

клас **Thread** містить методи для управління потоками;

новий потік запускається методом start().

Способи створення вторинного потоку в Java-програмі:

1. Реалізація інтерфейсу **java.lang.Runnable**:

public interface Runnable { void run(); }

2. Спадкування від класу **java.lang.Thread**

Реалізація інтерфейсу **java.lang.Runnable**:

```
class NewThread implements Runnable {
  Thread t;
  NewThread() {
    // Створення об'єкта вторинного потоку виконання
    t = new Thread(this, "Demo Thread");
    System.out.println("Child thread: " + t);
    // Запуск вторинного потоку виконання
    t.start();
  }
// «Тіло» вторинного потоку виконання
  public void run() {
    try {
      for (int i = 5; i > 0; i-i--) {
        System.out.println("Child Thread: " + i);
        Thread.sleep(500);
      }
```

```
    } catch (InterruptedException e) { ... }
  } }
```

Спадкування від **java.lang.Thread**:

```
class NewThread extends Thread {
  NewThread() {
    super("Demo Thread");
    // Запуск вторинного потоку виконання
    start();
  }
  // «Тіло» вторинного потоку виконання
  public void run() {
    try {
      for(int i = 5; i > 0; i-i--) {
        System.out.println("Child Thread: " + i);
        Thread.sleep(500);
      }
    } catch (InterruptedException e) { ... }
  }
}
```

Необхідність синхронізації потоків виконання:

всі потоки, що належать одному процесу, розділяють деякі загальні ресурси;

кожний потік може працювати з цими ресурсами без яких-небудь обмежень (асинхронно);

потрібен механізм, який дозволяє потокам, якщо буде потреба, погоджувати з іншими потоками роботу із загальними ресурсами.

Структура механізму синхронізації:

1. Набір об'єктів операційної системи, які створюються й управляються програмно та використовуються для координування доступу до ресурсів.
2. Основні види об'єктів синхронізації:
   взаємовиключення (mutex);
   критична секція (critical section);
   подія (event);
   семафор (semaphore).

Об'єкти синхронізації:

кожен з них може перебувати в сигнальному стані;

потоки можуть перевіряти поточний стан об'єкта синхронізації та/або чекати зміни цього стану і в такий спосіб погоджувати свої дії;

об'єкти синхронізації лише «підказують» потокам, коли можна працювати з ресурсом, а коли потрібно почекати.

Об'єкт синхронізації «критична секція» – це частина коду, де потік одержує доступ до ресурсу, що поділяється.

Порядок використання критичної секції:

1. Виділення критичної секції.
2. «Вхід» першого потоку в критичну секцію, скидання сигнального стану.
3. Припинення конкуруючого потоку.
4. «Вихід» першого потоку з критичної секції, установка сигнального стану.
5. «Вхід» другого потоку в критичну секцію...

Порядок виконання лабораторної роботи

Загальні рекомендації

При розробці графічного інтерфейсу користувача рекомендується використовувати відповідні візуальні засоби, що належать до складу середовища NetBeans.

До звіту повинна додаватися електронна документація до програми у форматі HTML.

Завдання до виконання

Розробити програму відповідно до загальної постановки завдання та одного з варіантів завдань.

Постановка завдання 1.

У магазині обслуговується **k** покупців (чоловіків і жінок). Кожний покупець може купити не більше **m** різних товарів **x.** Через якийсь час Δt після входу покупця в магазин він стає в чергу до каси.

Необхідно розробити програму з графічним інтерфейсом користувача, яка складається з декількох класів та дозволяє:

1. Створювати текстовий файл (ім'я файлу вибирається користувачем) із записами про покупців, кожен з яких перебуває в

окремому рядку. Найменування полів записів і їх припустимих значень наведені в табл. 4.

Таблиця 4

Найменування полів записів і їх значення

Стать покупця	x	Δt
Ціле число (1 – чоловік, 0 – жінка)	Ціле число (x <= m)	Речовинне число (див. табл. 5)

Значення полів записів повинні бути випадковими числами, рівномірно розподіленими на заданому інтервалі і генеруватися в програмі.

***Примітка**: обмеження на параметри **k, m**, **Δt**, символи-роздільники полів запису необхідно вибрати відповідно до варіанта завдання (табл. 5).*

2. Завантажувати інформацію про покупців із сформованого файлу до типізованого контейнера (об'єкта стандартного контейнерного класу Java SE, див. табл. 5). Інформація про покупців, що зберігається в контейнері, повинна бути представлена об'єктами відповідного класу.

3. Відображати інформацію про покупців і їх кількість у графічному інтерфейсі.

4. Вибирати варіант відображення інформації про покупців: всі покупці, покупці-жінки, покупці-чоловіки.

5. Обробляти стандартні виключення, які можуть виникнути при виконанні програми.

Таблиця 5

Варіанти завдань

№ варіанта	k	m	Δt^*	Символ-роздільник полів записів	Тип контейнера
1	100	5	μ· m· rand(0,5; 1,5)	пробіл	ArrayList<T>
2	105	6	μ· m· rand(0,5; 1,4)	;	LinkedList<T>
3	110	7	μ· m· rand(0,5; 1,3)	:	HashSet<T>
4	115	8	μ· m· rand(0,6; 1,5)	|	TreeSet<T>
5	120	9	μ· m· rand(0,7; 1,5)	/	HashMap<K,V>
6	125	10	μ· m· rand(0,8; 1,5)	\	TreeMap<K,V>
7	130	8	μ· m· rand(0,9; 1,5)	$	Hashtable<K,V>
8	135	7	μ· m· rand(0,6; 1,4)	&	Vector<T>
9	140	9	μ· m· rand(0,7; 1,3)	!	Stack<T>
10	150	6	μ· m· rand(0,8; 1,2)	#	LinkedHashMap<K,V>
11	160	9	μ· m· rand(0,8; 1,3)	%	LinkedHashSet<K,V>
12	170	8	μ· m· rand(0,7; 1,4)	+	TreeMap<K,V>

Примітка: *Якщо покупець жінка, тоді μ=200, чоловік – μ=150; rand(a, b) – випадкове число, рівномірно розподілене на інтервалі [a; b].*

Постановка завдання 2.

Розробити багатопотоковий додаток із графічним інтерфейсом, який дозволяє прочитати частини текстового документа, що перебувають у трьох різних файлах (1-й рядок документа – в першому файлі, 2-й рядок – у другому файлі, 3-й рядок – у третьому файлі й та. ін.), і коректно відобразити вміст усього документа. Читання даних з кожного файлу повинне виконуватися в окремому вторинному потоці виконання.

Приблизний вид графічного інтерфейсу додатка наведено на рис. 11.

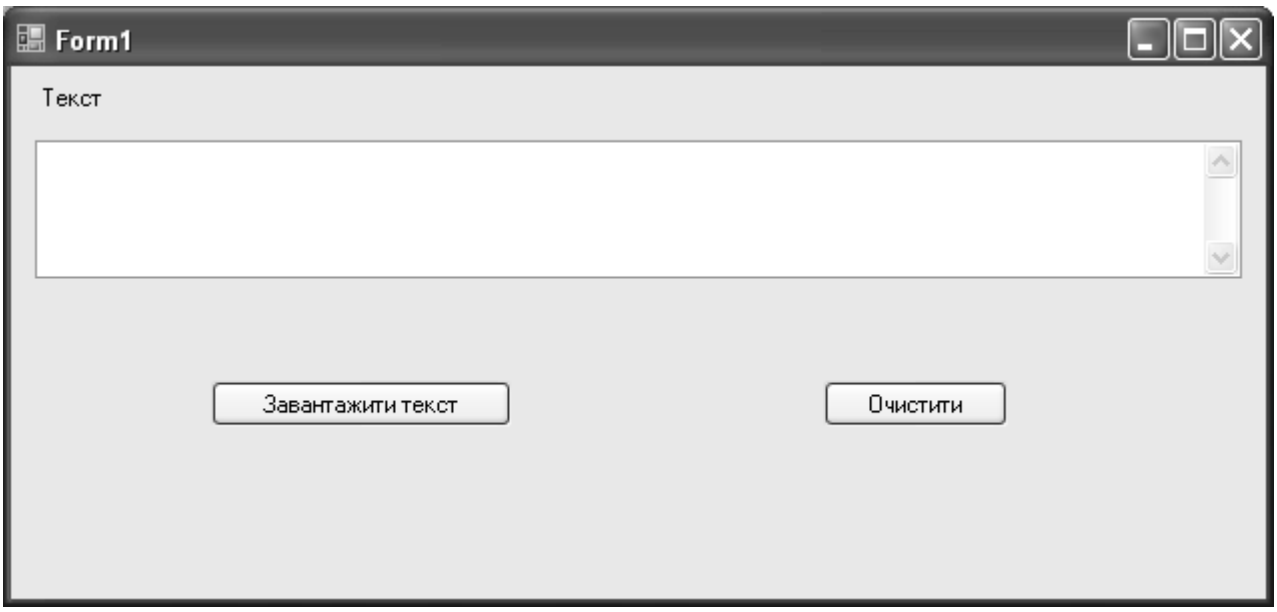


Рис. 11. **Графічний інтерфейс додатка**

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Проблеми «традиційного підходу» до обробки помилок під час виконання програми.
2. Переваги обробки виключень у порівнянні з «традиційним підходом» до обробки помилок.
3. Механізм обробки виключень.
4. Який клас є базовим для всіх помилок і виключень у Java SE?
5. У чому різниця між «перевірюваними» і «неперевірюваними» виключеннями?
6. Особливості байтових і символьних потоків введення-виведення.
7. Базові класи байтових потоків введення-виведення Java SE і їх основні методи.
8. Базові класи символьних потоків введення-виведення і їх основні методи.
9. Призначення основних класів символьних потоків введення-виведення Java SE.
10. Призначення основних класів байтових потоків введення-виведення Java SE.
11. Класи – «мости» між абстракціями символьних і байтових потоків введення-виведення Java SE та їх особливості.

12. У чому різниця між буферизованими й небуферизованими потоками введення-виведення Java SE?
13. Поняття процесу та потоку виконання.
14. Первинні й вторинні потоки виконання.
15. «Життєвий цикл» потоку виконання.
16. Способи створення нового потоку виконання в Java-програмі.
17. Для чого використовується синхронізація потоків виконання ?
18. Основні види об'єктів синхронізації.
19. Склад бібліотеки контейнерів Java SE.
20. Основні інтерфейси бібліотеки контейнерів Java SE.
21. Коротка характеристика структури даних «динамічний масив» і її реалізації в Java SE.
22. Коротка характеристика структури даних «двозв'язний список» і її реалізації в Java SE.
23. Поняття про хеш-таблицю та хеш-функцію.
24. Коротка характеристика структури даних «множина» та її реалізації в Java SE.
25. Коротка характеристика структури даних «асоціативний масив» і її реалізації в Java SE.
26. Ітератори та їх використання.
27. Основні «алгоритми» бібліотеки контейнерів Java SE.
28. Компаратори і їх використання.
29. Засоби розбору рядків на лексеми бібліотеки Java SE.

ЛАБОРАТОРНА РОБОТА 4
РОЗРОБКА ДОДАТКА, ЯКИЙ ВЗАЄМОДІЄ З БАЗОЮ ДАНИХ.

Мета лабораторної роботи:

1. Придбання практичних навичок з використання Java-технології доступу до баз даних (JDBC).

2. Придбання практичних навичок використання таблиць **javax.swing.JTable** і основних можливостей пакета **javax.swing.table** при розробці графічного інтерфейсу додатків, що взаємодіють з базою даних.

3. Удосконалення навичок роботи з інтегрованим середовищем розробки NetBeans і електронною документацією Java-розроблювача.

Перед виконанням лабораторної роботи студент повинен знати:

1. Порядок використання технології JDBC для доступу до реляційних баз даних.
2. Особливості застосування засобів розробки графічного інтерфейсу користувача в Java SE.
3. Порядок обробки подій у програмі з графічним інтерфейсом користувача.
4. Особливості використання таблиць **javax.swing.JTable.**

Після виконання лабораторної роботи студент повинен вміти:

1. Використовувати технологію JDBC для доступу до реляційних баз даних.
2. Застосовувати основні можливості таблиці **javax.swing.JTable** для відображення даних.

Основні елементи Java-технології доступу до баз даних Java Database Connectivity

Java Database Connectivity (JDBC) – стандартний API для доступу до баз даних з Java-додатків, який не залежить від особливостей конкретної системи управління базами даних (СУБД).

Архітектура JDBC наведена на рис. 12.

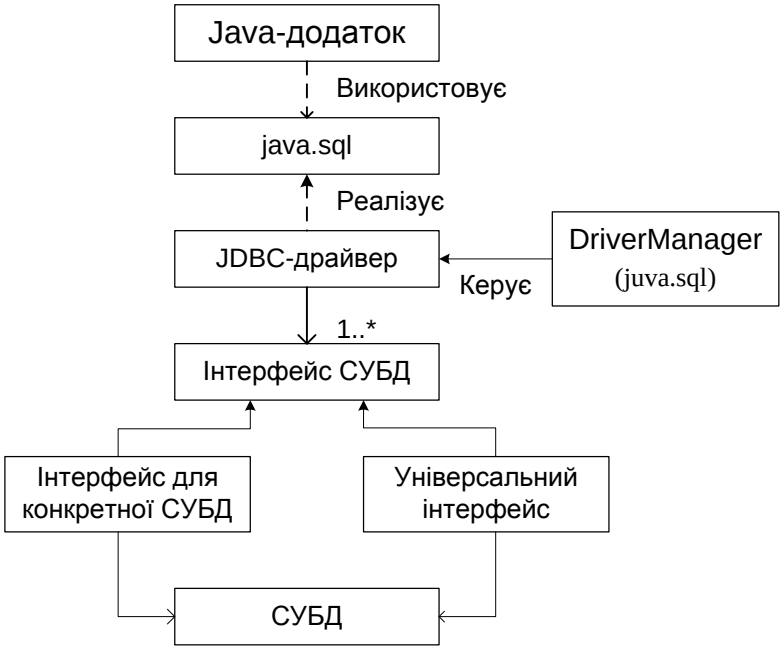


Рис. 12. **Архітектура JDBC**

Типи JDBC-драйверів подані в табл. 6.

Таблиця 6

Типи JDBC-драйверів

Розташування інтерфейсу СУБД	Залежність інтерфейсу від виробника СУБД	
	Не залежить	Залежить
На боці клієнта	**Тип 1:** JDBC-ODBC Bridge	**Тип 2:** Native-API Partly Java Technology-Enabled
На віддаленому вузлі	**Тип 3:** Net-Protocol Fully Java Technology-Enabled	**Тип 4:** Native Protocol Fully Java Technology-Enabled

Конфігурація JDBC-драйвера типу 1 наведена на рис. 13.

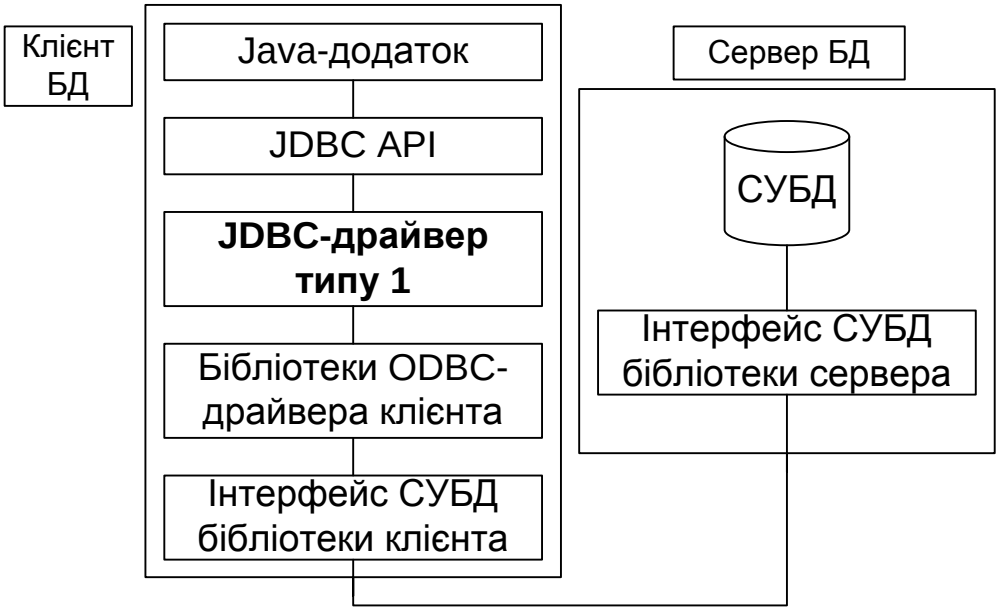


Рис. 13. **JDBC-драйвера типу 1**

Конфігурація JDBC-драйвера типу 2 наведена на рис. 14:

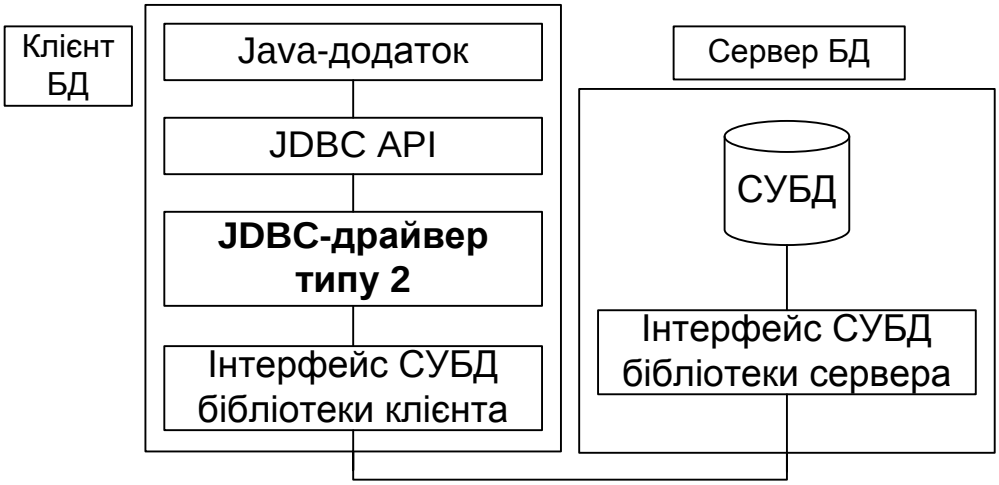


Рис. 14. **JDBC-драйвера типу 2**

Конфігурація JDBC-драйвера типу 3 наведена на рис. 15.

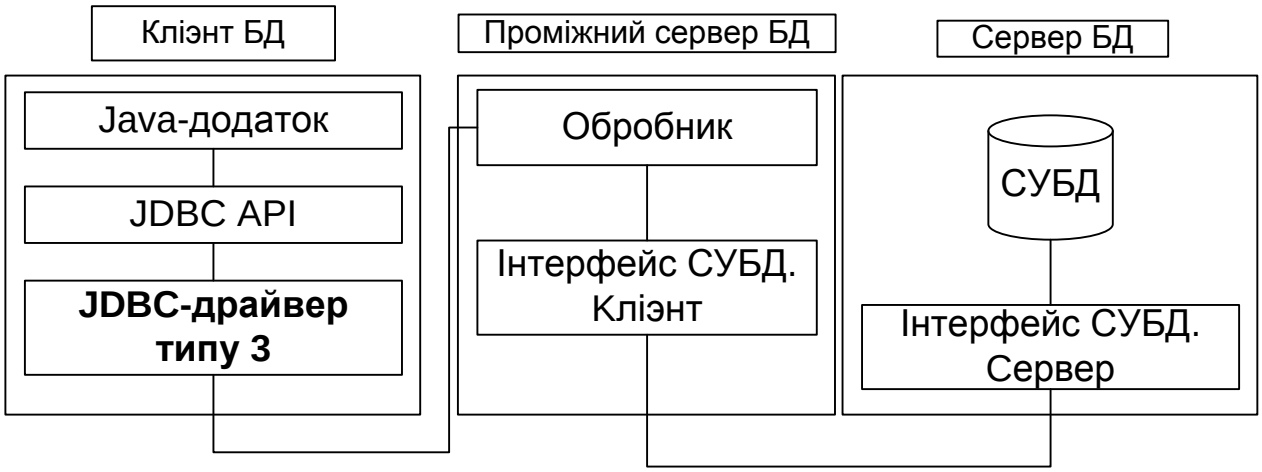


Рис. 15. **JDBC-драйвера типу 3**

Конфігурація JDBC-драйвера типу 4 наведена на рис. 16.

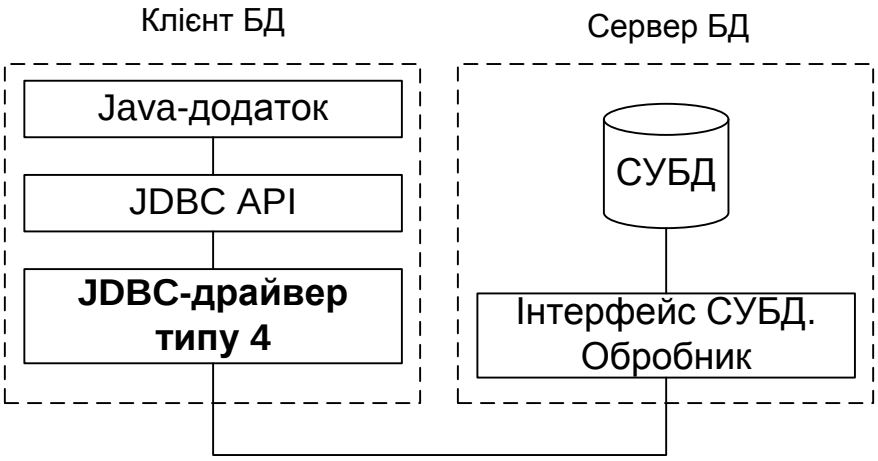


Рис. 16. **JDBC-драйвера типу 4**

Вибір JDBC-драйвера:

драйвер типу 1 не залежить від конкретного типу СУБД, але не забезпечує прийнятної продуктивності;

драйвер типу 2 забезпечує вищу продуктивність, але залежить від конкретного типу СУБД;

драйвер типу 3 більш універсальний, ніж драйвер типу 4, але має меншу продуктивність;

драйвер типу 4 забезпечує вищу продуктивність, ніж драйвер типу 3, але залежить від конкретного типу СУБД.

Основні абстракції JDBC наведені на рис. 17.

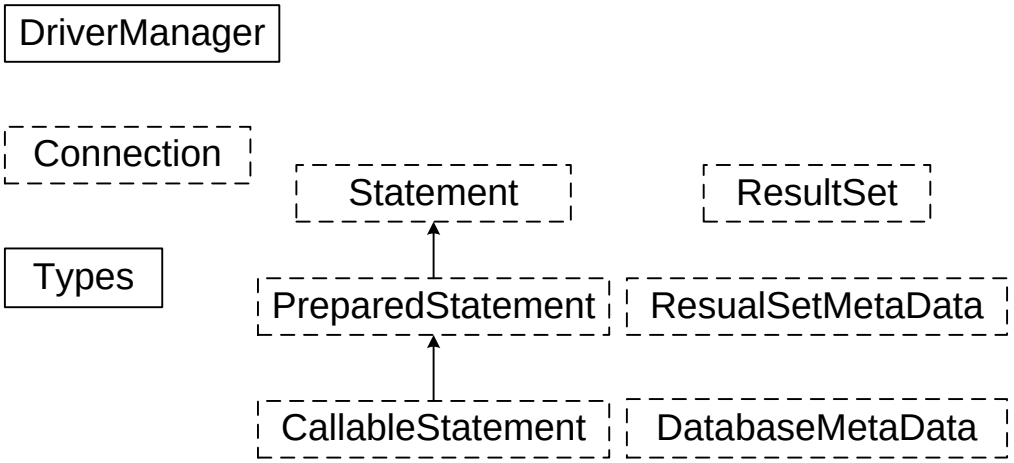


Рис. 17. **Основні абстракції JDBC**

Основні етапи взаємодії з БД за допомогою JDBC:

1. Завантаження JDBC-драйвера.
2. Визначення URL для установки з'єднання.
3. Установка з'єднання.
4. Створення посилання на Statement.
5. Виконання запиту.
6. Обробка результатів.
7. Закриття з'єднання.

Варіанти завантаження JDBC-драйвера (на прикладі JDBC-ODBC Bridge):

1. **java -D jdbc.drivers = sun.jdbc.odbc.JdbcOdbcDriver MainClass**
2. **Class theDriver = sun.jdbc.odbc.JdbcOdbcDriver.class;**
3. **Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");**
4. **sun.jdbc.odbc.JdbcOdbcDriver theDriver = new sun.jdbc.odbc.JdbcOdbcDriver();**

Формат рядка для завдання URL БД:

String url = "*jdbc:sub_protocol:db_identifier*"

де sub_protocol – ім'я драйвера або механізму доступу до бази даних (наприклад, ODBC);

db_identifier – "логічне" ім'я бази даних (наприклад, ODBC DSN).

Приклад завдання URL БД і установки з'єднання:

```
String url = "jdbc:oracle:thin:@//uni:1521/histest";
Connection con = DriverManager.getConnection(url, user,
password);
```

Способи створення об'єкта для зберігання SQL-запиту:

1. **Statement stmt = con.createStatement();**
2. **PreparedStatement pr_stmt = con.prepareStatement("PreparedQuery");**

Варіанти виконання SQL-запиту:

1. **ResultSet rs = stmt.executeQuery("SQL-Query");**
2. **int rowCount = stmt.executeUpdate("SQL-Query");**
3. **boolean returnValue = stmt.execute("SQL-Query");**

Обробка результатів, закриття з'єднання:

```
while (rs.next()) {
  String s = rs.getString("COF_NAME");
  float n = rs.getFloat("PRICE"); System.out.println(s + " " + n);
}
con.close();
```

Використання PreparedStatement:

```
PreparedStatement updateSales;
String updateString = "update COFFEES " + "set SALES = ? where
COF_NAME = ?";
updateSales = con.prepareStatement(updateString);
int [ ] salesForWeek = {175, 150, 60};
String [ ] coffees = {"Colombian", "French_Roast"};
int len = coffees.length;
for(int i = 0; i < len; i++){
  updateSales.setInt(1, salesForWeek[i]);
  updateSales.setString(2, coffees[i]);
  updateSales.executeUpdate();
 }
```

Використання таблиць у графічному інтерфейсі Java-додатків

Загальні відомості про компонент javax.swing.JTable:

є уявленням (View) для даних;

дає можливість відображати двовимірну інформацію, розташовану у вигляді рядків і стовпців;

дозволяє гнучко налаштовувати зовнішній вигляд;

забезпечує можливість редагування даних;

допоміжні класи для JTable – javax.swing.table

Прості таблиці мають наступній вигляд:

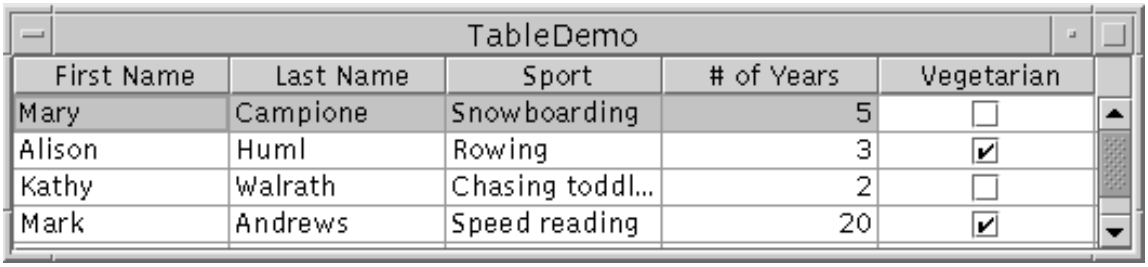


Рис. 18. **Проста таблиця TableDemo, що побудована за допомого класу JTable**

Конструктори:

JTable(Object[ ][ ] rowData, Object[ ] columnNames)
JTable(Vector rowData, Vector columnNames)

Особливості:

усі елементи є такими, що редагуються;

для відображення даних використовується їх рядкове уявлення;

дані повинні міститися у векторі або масиві.

Типовий код створення простої таблиці:

1. Таблиця в панелі прокрутки:

```
JTable table = JTable(rowData, columnNames);
JScrollPane scrollPane = new JScrollPane(table);
new JPanel().add(scrollPane);
```

2. Таблиця сама по собі:

```
JTable table = JTable(rowData, columnNames);
JPanel panel = new JPanel();
```

```
panel.setLayout(new BorderLayout());
panel.add(table.getTableHeader(),
    BorderLayout.NORTH);
panel.add(table, BorderLayout.CENTER);
```

Моделі (Models) таблиці JTable:

javax.swing.table.TableModel – зберігає дані елементів таблиці, а також додаткову службову інформацію про ці елементи;

javax.swing.table.TableColumnModel – дозволяє маніпулювати стовпцями, дізнаватися про зміни в їх даних, змінювати модель виділення стовпців і т. д.

javax.swing.ListSelectionModel – управляє виділенням рядків таблиці.

Модель даних таблиці подана на рис. 19.

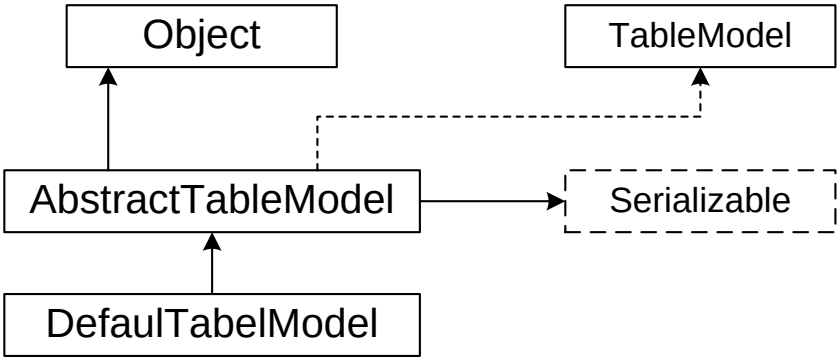


Рис. 19. **Модель даних таблиці**

Стандартна модель даних – DefaultTableModel:

є найшвидшим способом створити таблицю і заповнити її даними;

зберігає передані їй дані у двох об'єктах Vector;

дозволяє повно описати кожен елемент таблиці;

методи моделі таблиці дають можливість одержати значення довільного елементу і змінити його, отримати інформацію про стовпці та рядки;

має низьку продуктивність;

має обмежені можливості настройки типів даних стовпців.

Методи TableModel, які потрібно перевизначати обов'язково:

1. **public int getRowCount()** – повертає кількість рядків у таблиці.

2. **public int getColumnCount()** – повертає кількість стовпців у таблиці.

3. **public Object getValueAt(int row, int column)** – основний метод моделі даних таблиці (дозволяє вказати, які дані знаходяться в елементі, який визначається за вказаним рядком і стовпцем).

Приклад реалізації моделі даних:

```
class SimpleModel extends AbstractTableModel {
  public int getRowCount() { return 100000; }
  public int getColumnCount() {return 3; }
  public Object getValueAt(int row, int column){
    boolean flag = (row == 0) ? true : false;
    switch (column) {
      case 0: return "" + row;
      case 1: return new Boolean(flag);
      case 2: return new ImageIcon(“Table.gif”);
    }
    return "Порожньо“;
  } }
```

Модель для роботи з базою даних:

```
class DatabaseModel extends AbstractTableModel
{
  private ArrayList column Names = new ArrayList();
  private Arraylist columnTypes = new ArrayList();
  private ArrayList data = new ArrayList();
  public int getRowCount() { ... }
  public int getColumnCount() { ... }
  public Object getValueAt(int row, int column) { ... }
  .....................
  public void getData(ResultSet rs) { ... }
}
```

Таблиця бази даних має наступний вигляд

ID	SUP_NAME	STREET	CITY	ZIP
101	Acme, Inc.	99 Market Street	Groundsville	95199
49	Superior Coffee	1 Party Place	Mendocino	95460
150	The High Ground	100 Coffee Lane	Meadows	93966

Отримання даних бази і сповіщення таблиці:

```
public void getData(ResultSet rs){
  ResultSetMetaData metaData = rs.getMetaData();
  int columnCount = metaData.getColumnCount();
  for (int i = 0; i < columnCount; i++) {
    // Заповнення списків column Names, columnTypes
    .....
  }
  fireTableStructureChanged();
  while (rs.next() ) {
    // Заповнення списку  data
    .....
  }
  fireTableDataChanged();
}
```

Порядок виконання лабораторної роботи

Загальні рекомендації

При розробці графічного інтерфейсу користувача слід використовувати відповідні візуальні засоби, що належать до складу середовища NetBeans.

До звіту повинна додаватися електронна документація до програми у форматі HTML.

Основні джерела інформації з використання компонента javax.-swing.JTable та основних можливостей пакета javax.swing.table [3; 5; 6].

Основні джерела інформації щодо використання технології JDBC [2; 3].

Постановка завдання

Розробити додаток з графічним інтерфейсом, що складається з декількох класів, що взаємодіє з базою даних, створеною за допомогою СУБД Microsoft Access.

Загальні вимоги до функціональності додатка:

1. Імпортувати дані з текстового файлу (див. варіанти завдань до лабораторної роботи 3, постановка завдання 1) в таблицю бази даних.
2. Вибирати всі дані з таблиці бази даних і відображати їх у графічному інтерфейсі користувача.
3. Обробляти стандартні виключення.

Загальні вимоги до графічного інтерфейсу додатка:

1. Головне вікно додатка – фрейм.
2. Наявність меню "Дані" (пункти "Імпортувати", "Показати"), "Допомога" (пункт "Про програму").
3. Дані з бази повинні відображатися в табличному вигляді (у компоненті javax.swing.JTable).

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Типи JDBC-драйверів та їх коротка характеристика.
2. Основні етапи взаємодії з БД за допомогою JDBC.
3. Для чого призначений інтерфейс Connection?
4. Для чого призначений інтерфейс Statement?
5. Для чого призначений інтерфейс PreparedStatement?
6. Для чого призначений інтерфейс CallableStatement?
7. Для чого призначений інтерфейс ResultSet?
8. Для чого призначений інтерфейс ResultSetMetaData?
9. Для чого призначений інтерфейс DatabaseMetaData?
10. Де спочатку встановлений курсор у наборі результатів SQL-запиту?
11. Склад і коротка характеристика моделей таблиці JTable.
12. Архітектура моделі даних таблиці JTable.
13. У об'єкті якого з класів Java SE за замовчуванням зберігаються дані елементів таблиці JTable?
14. Який з інтерфейсів повинна реалізувати модель, що зберігає дані елементів таблиці JTable?

15. Які з методів необхідно обов'язково перевизначити в користувальницькій моделі даних таблиці JTable?

ЛАБОРАТОРНА РОБОТА 5
ВИКОРИСТАННЯ TCP/IP СОКЕТІВ У РОЗПОДІЛЕНИХ ДОДАТКАХ КЛІЄНТ-СЕРВЕР

Мета лабораторної роботи:

1. Придбання практичних навичок з розробки розподілених додатків клієнт-сервер із застосуванням TCP/IP сокетів.
2. Придбання практичних навичок з використання основних можливостей пакета javax.swing при розробці аплетів.
3. Удосконалення навичок роботи з інтегрованим середовищем розробки NetBeans і електронною документацією Java-розроблювача.

Перед виконанням лабораторної роботи студент повинен знати:

1. Призначення й організацію пакета java.net бібліотеки Java SE.
2. Структуру розподіленого Java-додатка який використовує TCP/IP сокети.
3. Особливості розробки і "життєвий цикл" аплетів.

Після виконання лабораторної роботи студент повинен вміти:

1. Розробляти прості розподілені Java-додатки клієнт-сервер з використанням TCP/IP сокетів.
2. Розробляти графічний інтерфейс користувача із застосуванням Java-аплетів.

Загальні відомості про розробку розподілених додатків на платформі Java SE

Рівні моделі взаємодії відкритих систем (Open System Interconnected - OSI) наведені на рис. 20:

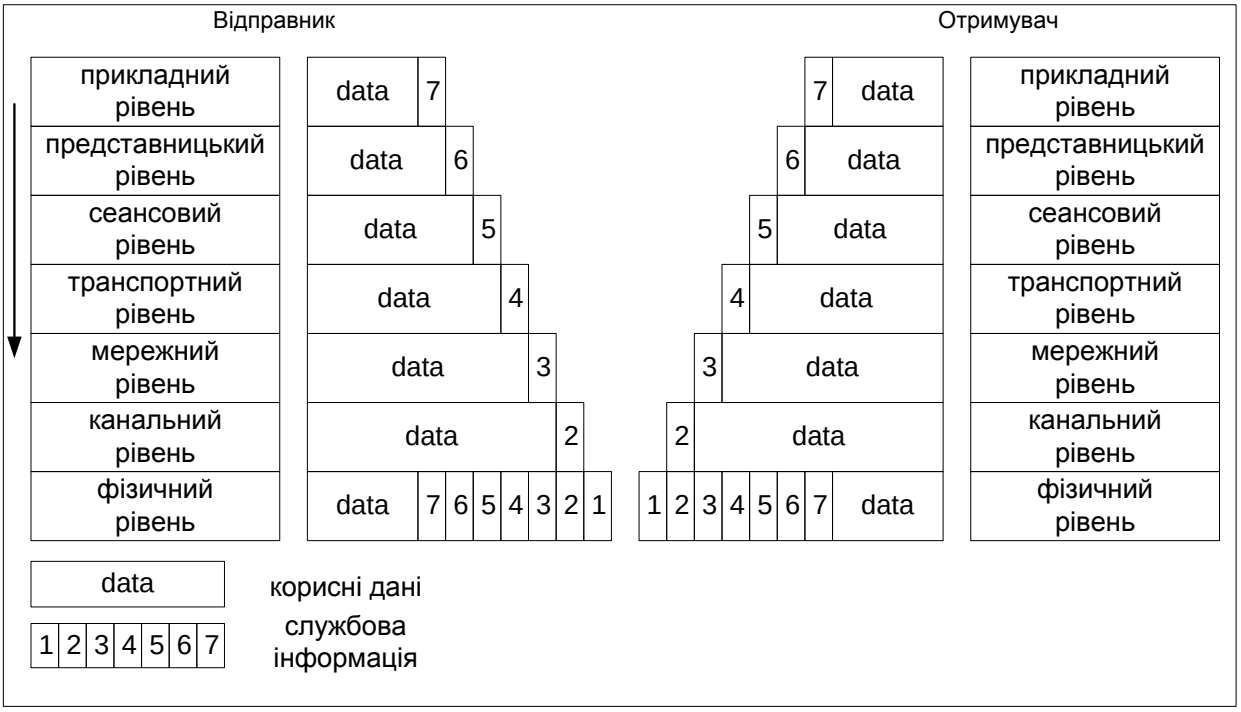


Рис. 20. **Рівні моделі OSI**

Мережний рівень:

найбільш поширеним протоколом передачі даних є Internet Protocol (IP);

для унікальної ідентифікації комп'ютерів у мережі використовують ієрархічну систему адресації (IP-адреса);

протокол IP забезпечує найкращу, але не гарантовану доставку даних крізь мережу;

основний пристрій, що використовується на цьому рівні, – маршрутизатор, який об’єднує віддалені локальні мережі в глобальну мережу;

основне завдання маршрутизатора – вибір шляху, яким буде пересилатися повідомлення.

IP-адресація

Формат IP-адреси: **xxx.xxx.xxx.xxx**, де xxx – 0…255. Класи та діапазоні комп’ютерних мереж подані у табл.. 7.

Транспортний рівень:

протоколи транспортного рівня – Transmission Control Protocol (TCP) і User Datagram Protocol (UDP);

протокол TCP – забезпечує віртуальні з'єднання між користувальницькими додатками та гарантує точну доставку даних;

протокол UDP призначений для швидкого обміну спеціальними повідомленнями (датаграмами) без підтвердження їх доставки.

Таблиця 7

Класи комп'ютерних мереж

Клас мережі	Мінімальний номер мережі	Максимальний номер мережі	Максимальна кількість вузлів
A	1.0.0.0	126.0.0.0	2^{24}
B	128.0.0.0	191.255.0.0	2^{16}
C	192.0.0.0	223.255.255.0	2^{8}
D	224.0.0.0	239.255.255.255	Multicast
E	240.0.0.0	247.255.255.255	Резерв

Основні характеристики протоколів TCP і UDP:

TCP	UDP
Для роботи встановлює з'єднання.	Працює без з'єднань.
Гарантована доставка даних.	Гарантії доставки даних немає.
Розбиває початкове повідомлення на сегменти.	Передає повідомлення цілком у вигляді датаграм.
На стороні одержувача повідомлення заново збирається із сегментів.	На стороні одержувача прийняті повідомлення не об'єднуються.
Втрачені сегменти пересилає заново.	Підтверджень про доставку не існує.
Контролює потік сегментів.	Не існує ніякого контролю потоку датаграм.

Порядок роботи за протоколом TCP:

робота починається з установлення з'єднання;

ініціатор з'єднання та одержувач обмінюються спеціальними пакетами в три етапи ("запит", "підтвердження", "підтвердження на підтвердження");

після успішного встановлення з'єднання учасники можуть почати обмінюватися даними;

якщо відправник протягом визначеного часу (тайм-ауту) не отримує підтвердження, тоді він вважає, що пакет n втрачений, і посилає його ще раз.

Поняття «порт»:

порт описується числом від 0 до 65535 і дозволяє операційній системі розподіляти пакети, що приходять на транспортний рівень, між різними прикладними програмами;

кажучи про встановлене TCP – з'єднання, мають на увазі 4 числа: IP-адресу та порт однієї сторони і ті ж параметри другої сторони.

Порядок розподілу портів:

порти з номерами менше 255 використовуються для публічних сервісів;

порти з номерами в діапазоні 255 – 1023 призначаються компаніями-розробниками для своїх додатків;

номери портів понад 1023 – не регулюються.

Сокет – сукупність IP-адреси і номера порту.

Архітектура розподіленого додатка подана на рис. 21.

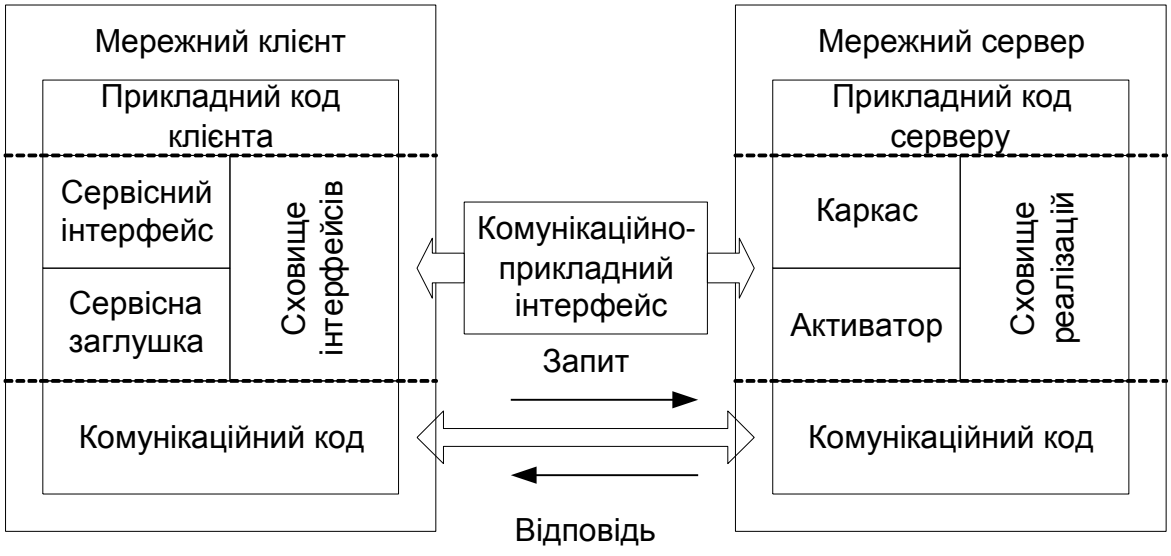


Рис. 21. **Архітектура розподіленого додатка**

Основні класи пакета java.net :

Основі класи які необхідні для розробки розподілених додатків об’єднані у пакеті java.net (рис. 22).

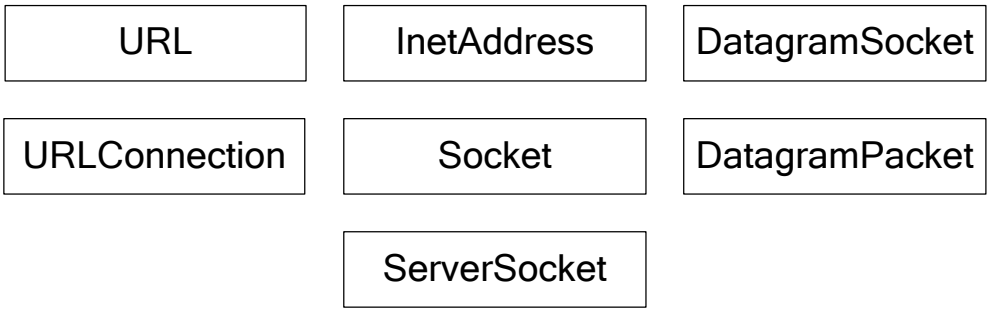


Рис. 22. **Класи пакету java.net**

Серверний додаток:

1. Створює серверний сокет (об'єкт класу ServerSocket).
2. Чекає запиту клієнта.
3. Створює і відкриває клієнтський сокет (об'єкт класу Socket).
4. Відкриває потік введення і потік виведення даних.
5. Читає дані з потоку введення й записує дані в потік виведення.
6. Закриває потоки введення та виведення даних.
7. Закриває сокет.

Клієнтський додаток:

1. Створює і відкриває клієнтський сокет (об'єкт класу Socket).
2. Відкриває потік введення і потік виведення даних.
3. Читає дані з потоку введення й записує дані в потік виведення відповідно до протоколу серверу.
4. Закриває потоки введення та виведення даних.
5. Закриває сокет.

Загальний вигляд коду серверного додатка:

```
import java.io.*;
import java.net.*;
. . .
int port = 3456;
ServerSocket s;
try {
    s = new ServerSocket(port);
    Socket c = s.accept();
    OutputStream out = c.getOutputStream();
    InputStream in = c.getInputStream();
```

```
        . . . . . .
  } catch (IOException e) { … }
```

Загальний вигляд коду клієнтського додатка:

```
import java.io.*;
import java.net.*;
. . .
int port = 3456;
String host = “localhost";
try
{
   s = new Socket(host, port);
   OutputStream out = s.getOutputStream();
   InputStream in = s.getInputStream();
      . . . . . .
}
catch (IOException e) { … }
```

Загальні відомості про аплети

Аплети – це невеликі програми, які працюють під управлінням Web-браузера.

Тут прикметник "невеликі" відображає типову практику використання аплетів, а не якусь формальну вимогу. Теоретично аплети можуть бути як маленькими, так скільки завгодно великими і складними. Але об'єм коду аплета впливає на час його запуску, оскільки зазвичай код аплета передається мережею Internet/Intranet. Відповідно великий аплет потребує багато часу на завантаження і його використання не буде виправданим.

Поточні версії основних Web-браузерів, зокрема *Microsoft Internet Explorer,* підтримують роботу аплетів. Для цього в кожен з них умонтована віртуальна Java-машина.

Проблема безпеки

Суть проблеми полягає в тому, що аплет – це, як правило, програма, яка отримана із зовнішнього джерела. Відповідно вона потенційно небезпечна. Через злий намір або через помилку вона може містити небезпечний програмний код.

З метою безпеки аплети за замовчуванням, дуже обмежені у своїх правах. Ці обмеження визначаються настройками конкретного браузера. Вони можуть бути ослаблені самим користувачем як відносно всіх аплетів, так і для конкретних аплетів.

Обмежень досить багато, наприклад, аплети не можуть ні читати, ні записувати на локальний диск. Вони мають можливість передавати інформацію тільки за тією мережною адресою, звідки був завантажений аплет. Вікно аплета спеціально виділяється або робиться якась інша позначка для того, щоб користувач уявляв собі, що він має справу з аплетом, а не із звичайною Web-сторінкою.

Створення аплетів

Техніка розробки аплетів базується на класі **JApplet** пакета **javax.swing**. Цей клас має багато своїх методів і ряд методів, успадкованих від класу **java.awt.Applet.**

Для побудови аплета потрібно створити клас – спадкоємець класу **JApplet –** і перевизначити в ньому ряд методів класу **Applet**. У класі JApplet ці методи реалізовані як порожні заглушки, які нічого не роблять. При роботі аплета всередині браузера він викликає дані методи в певних ситуаціях. Якщо перевизначити ці методи, то браузер викличе їх, а не методи класу **Applet**.

Далі розглянемо зазначені методи.

public void init()

Викликається браузером відразу після завантаження аплета перед першим викликом методу **start()**. Цей метод потрібно перевизначати практично завжди, якщо в аплеті необхідна ініціалізація даних.

public void start()

Викликається браузером при кожному "відвідуванні" даної сторінки. Використовується зазвичай у комбінації з методом **stop()** для економії ресурсів, наприклад, у тому разі, якщо аплет виконує деяку анімацію. Тоді stop() може її зупинити, а start() – запустити знову.

public void stop()

Викликається браузером при деактивізації даної сторінки як у разі завантаження нової сторінки без вивантаження даної, так і в разі її вивантаження. У останньому випадку stop() викликається перед destroy().

public void destroy()

Викликається браузером перед вивантаженням даної сторінки.

Зазвичай при створенні аплета перевизначають метод init() і реалізують у ньому графічний інтерфейс користувача. При цьому вся функціональність аплета забезпечується «слухачами» подій візуальних елементів управління.

Приклад аплета

У даному прикладі присутня мітка з написом "Перший аплет", кнопка і текстове поле, в яке при натисненні на кнопку виводиться текст "Привіт".

```
import java.awt.* ;
import java.awt.event.*;
import javax.swing.*;

public class AppDemo extends JApplet {
   JTextField txt = new JTextField(15);

   public AppDemo() { }

   public void init() {
    Container c = getContentPane();
    c.setLayout(new FlowLayout());
    JLabel lbl = new JLabel("Перший аплет");
    c.add(lbl);
    JButton btn = new JButton("Натиснути один раз");
    btn.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent e) {
         txt.setText("Привіт");
       }
    });
    c.add(btn);
    c.add(txt);
```

```
    }
}
```

Запуск аплетів

Аплет запускається браузером при перегляданні Web-сторінки, в якій є посилання на цей аплет. Для запуску аплета достатньо написати зовсім простий код у складі Web-сторінки.

Наприклад, для запуску наведеного вище аплета *AppDemo* можна створити такий файл:

```
<html>
<head><title>First Applet</title></head>
<body>
Якийсь- текст на сторінці
<applet code=AppDemo width=400 height=200>
</applet>
</body>
</html>
```

Тут для підключення аплета використаний HTML-тег **<applet>.** Він має параметри **code**, **width** і **height**. Посилання на аплет реалізоване за допомогою параметра **code**.

Якщо аплет заархівований у jar-файл, для посилання на архів потрібно використовувати параметр **archive** тега **applet**. Наприклад, для файлу MyApplet.jar:

```
<applet code="." width=400 height=200 archive=MyApplet.jar>
</applet>
```

Порядок виконання лабораторної роботи

Загальні рекомендації

При розробці графічного інтерфейсу користувача слід використовувати відповідні візуальні засоби, що належать до складу середовища NetBeans.

До звіту повинна додаватися електронна документація до програми у форматі HTML.

Основні джерела інформації щодо використання бібліотеки компонентів Swing [2; 3; 5].

Постановка завдання

1. Розробити розподілений додаток клієнт-сервер з використанням TCP/IP сокетів, який виконується на локальному комп'ютері.

2. Функціональність додатка повинна бути реалізована в декількох класах.

Загальні вимоги до функціональності клієнтського додатка:

1. Має становити swing-аплет.

2. Графічний інтерфейс аплета повинен дозволяти вводити дані про користувача (прізвище, ім'я, по батькові), ініціювати їх відправку серверному додатку шляхом натиснення відповідної кнопки, відображати відповідь серверу в стандартному діалоговому вікні MessageBox (див. клас **javax.swing.JOptionPane**).

3. Додаток має формувати рядок на основі вводу, виконаного користувачем, і відправляти його серверному додатку через сокет. Текстовий рядок повинен мати наступний формат:

Прізвище Ім'я По батькові Ідентифікатор користувача

Приклад текстового рядка:

Петренко Наталя Вікторівна ПНВ479248

4. Ідентифікатор користувача повинен формуватися на основі даних, що вводяться за допомогою графічного інтерфейсу аплета, і складатися з перших букв прізвища, імені та по батькові, а також випадкового цілого числа (див. приклад до п. 3). Параметри генерації випадкового числа наведені у варіантах завдань.

Загальні вимоги до функціональності серверного додатка:

1. Має становити однопотокову консольну програму.

2. Повинен приймати дані (рядок) від клієнтського додатка через сокет, додавати в кінець прийнятого рядка номер свого порту і відправляти цей рядок назад клієнтові. Текстовий рядок, переданий клієнтському додатку, матиме такий вигляд:

Прізвище Ім'я По батькові Ідентифікатор користувача: порт xxx

Приклад текстового рядка:

Петренко Наталя Вікторівна ПНВ479248: порт 3456

4. Має забезпечувати очікування підключення клієнта в нескінченному циклі.

5. Повинен виводити на консоль текстові повідомлення про старт серверу, підключення клієнта, отримані від клієнта і відправлені йому дані.

Рекомендація: *для введення даних доцільно в клієнтському та серверному додатках використовувати потік* ***PrintWriter****, а для виведення –* ***BufferedReader***

Варіанти завдань

№ варіанта	Параметри генерування випадкового числа для ідентифікатора користувача, (random[a, b] – випадкові цілі числа, рівномірно розподілені на інтервалі від a до b включно)
1	random[100, 1000]
2	random[1500, 2000]
3	random[2500, 3000]
4	random[3500, 4000]
5	random[4500, 5000]
6	random[5500, 6000]
7	random[6500, 7000]
8	random[7500, 8000]
9	random[8500, 9000]
10	random[9500, 10000]
11	random[10500, 11000]
12	random[11500, 12000]

КОНТРОЛЬНІ ПИТАННЯ

1. Призначення протоколу TCP.
2. Призначення протоколу UDP.
3. Основні відмінності протоколів TCP і UDP.
4. Що таке порт?
5. Сокети та їх основні характеристики.
6. Призначення класів **Socket**, **ServerSocket**.
7. Який клас є базовим для класу **JApplet**?
8. Для чого потрібні менеджери компонування?
9. Які методи визначають "життєвий цикл" аплета?
10. Порядок виклику методів "життєвого циклу" аплета?
11. Призначення методу **paint()** класу **java.awt.Container**?

ЛАБОРАТОРНА РОБОТА 6
РОЗРОБКА WEB-ДОДАТКУ З ВИКОРИСТАННЯМ JAVA-ТЕХНОЛОГІЙ

Мета лабораторної роботи:

1. Придбання практичних навичок розробки Web-додатків з використанням Java-технологій Servlets і JSP.
2. Удосконалення практичних навичок розробки Web-сторінок.
3. Удосконалення навичок роботи з інтегрованим середовищем розробки NetBeans і електронною документацією Java-розроблювача.

Перед виконанням лабораторної роботи студент повинен знати:

Основи технологій Dynamic HTML, Servlets, JSP.

Після виконання лабораторної роботи студент повинен вміти:

Використовувати основні елементи технологій Dynamic HTML, Servlets і JSP при розробці Web-додатків.

Основні відомості про Dynamic HTML, сервлети, JSP

Dynamic HTML:

становить спосіб створення інтерактивних Web-сайтів, що використовує поєднання статичної мови розмітки HTML, скриптової мови JavaScript, CSS (каскадних таблиць стилів) і DOM (об'єктної моделі документа);

дозволяє додати "динаміку" на статичні Web-сторінки;

використовується для управління всіма елементами Web-сторінки;

реалізація залежить від браузера.

Перевірка коректності даних, введених у форму, перед відправкою їх на сервер за допомогою сценарію мовою **JavaScript**:

зовнішній вигляд форми наведений на рис. 23:

Рис. 23. **Зовнішній вигляд форми**

HTML-код форми:

```
<form action="/WebSite/FormServlet" name="MyForm">
<input type="text" name="LastName" value="" width="20" />
<input type="submit" value="Відправити" name="send"
onClick="return ValidateData();"/>
 </form>
```

сценарій JavaScript на Web-сторінці:

```
<script type = "text/javascript">
    function ValidateData() {
if (MyForm.LastName.value == ' ') {
                alert('Ви повинні ввести прізвище користувача');
                return false;
              }
              else
                return true;
    }
</script>
```

Сервлети і JSP:

сервлет – самостійний програмний компонент, що функціонує на Web-сервері і дозволяє динамічно генерувати HTML-сторінку або інший документ у відповідь на отриманий від клієнта запит;

JSP – технологія, яка дозволяє додавати Java-код до статичного змісту Web-сторінки і динамічно генерувати HTML-, XML- документ та ін.

сервлети і JSP є складовою частиною єдиної Java-технології Java 2 Enterprise Edition (J2EE).

Архітектура J2EE подана на рис. 24.

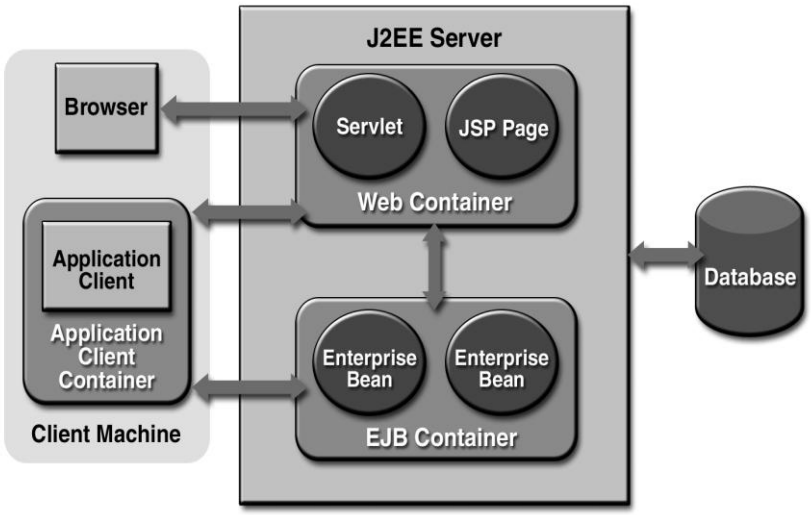


Рис. 24. **Архітектура J2EE**

Поняття Web-контейнеру:

WEB-контейнер – це програмне забезпечення, що управляє життєвим циклом сервлетів і JSP;

відповідає за взаємодію з Web-сервером;

існують Web-контейнери від різних виробників з різною функціональністю;

так зване “Reference Implementation” технологій сервлетів і JSP – Web-контейнер Apache TomCat;

Apache TomCat – програмне забезпечення, яке вільно розповсюджується (http://jakarta.apache.org).

Web-додаток:

становить повний набір ресурсів, що призначений для Web-сайта;

склад Web-додатка:

1. Файл конфігурації – web.xml (містить метадані для додатка й використовується Web-контейнером при його завантаженні).
2. Статичні файли і JSP.
3. Java-файли з байт-кодом (.class).

Загальні відомості про сервлети:

сервлет – це певним чином побудований Java-клас, що не має прив'язки до якої-небудь конкретної платформи або Web-серверу;

сервлет безпосередньо не зв'язується з клієнтом, а в ролі посередника, що підтримує зв'язок з віддаленим клієнтом, виступає Web-контейнер;

технологію сервлетів можливо використовувати на будь-якій платформі, для якої є віртуальна Java-машина, і на будь-якому Web-сервері, що має відповідну підтримку;

завдяки мові Java сервлети можливо переносити з однієї платформи на іншу без перекомпіляції.

«Життєвий цикл» HTTP-сервлета наведений на рис. 25:

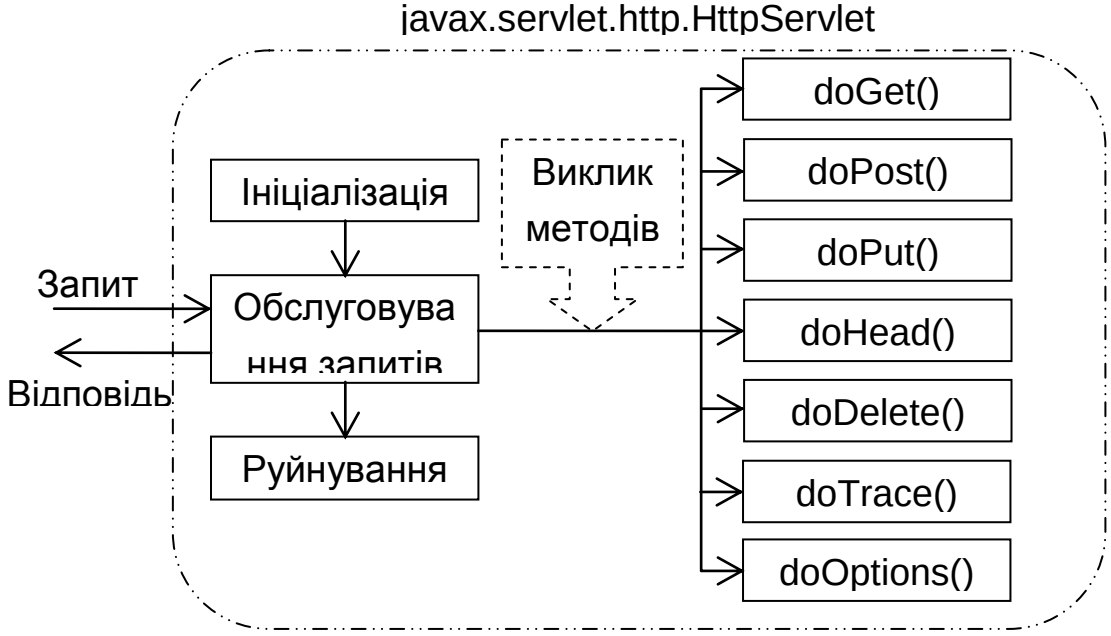


Рис. 25. **«Життєвий цикл» HTTP-сервлета**

Розробка сервлета:

для побудови сервлета розроблювач повинен написати клас, що реалізовує інтерфейс **javax.servlet.Servlet**;

даний інтерфейс містить три методи: **init()**, **service()** і **destroy()**, що визначають «життєвий цикл» сервлета:

1. Метод **init()** викликається контейнером у момент завантаження сервлета.

2. Метод **destroy()** викликається контейнером перед тим, як сервлет буде вивантажений із сервера.

3. Метод **service()** викликається контейнером при кожному зверненні клієнта до сервлета.

Приклад HTTP-сервлета, який генерує HTML-документ:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest request,
  HttpServletResponse response)
  throws IOException, ServletException {
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     out.println("<html>");
     out.println("<head>");
     out.println("<title>Hello World!</title>");
     out.println("</head>");
     out.println("<body>");
     out.println("<h1>Hello World!</h1>");
     out.println("</body>");
     out.println("</html>");
  }
}
```

Результат роботи сервлета подано на рис. 26.

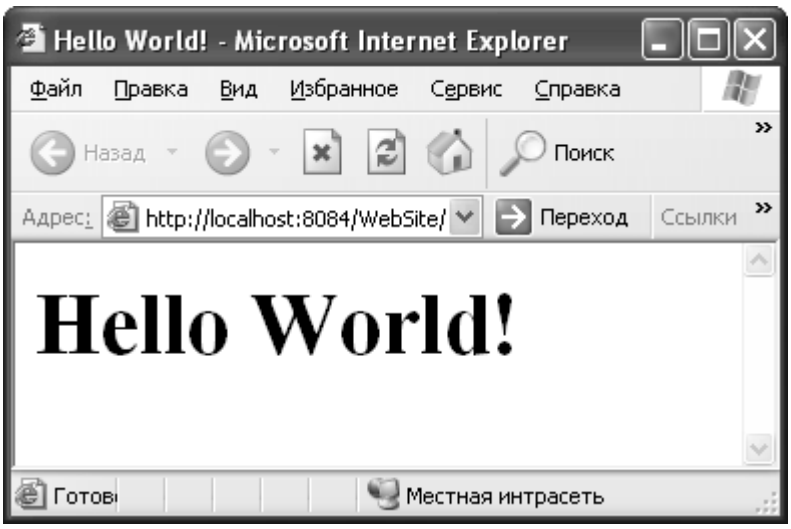


Рис. 26. **Результат роботи сервлета**

Читання даних форми сервлетом:

дані форми передаються на Web-сервер як параметри HTTP-запиту;

будь-яка інформація, передана в рядку запиту, доступна сервлету через об'єкт класу **HttpServletRequest** та його методи:

1. String getParameter(String parameterName).
2. String[ ] getParameterValues(String name).
3. java.util.Enumeration getParameterNames().

Приклад читання даних форми сервлетом і формування HTTP-відповіді:

```
public class FormServlet extends HttpServlet {

  protected void processRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String lastName = request.getParameter("LastName");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Відповідь серверу</title>");
        out.println("</head>");
        out.println("<body>");
       out.println("<h1>Введене прізвище: " + lastName + "</h1>");
```

```
            out.println("</body>");
            out.println("</html>");
            out.close();
              }

        protected void doGet(HttpServletRequest request,
                                    HttpServletResponse response)
                                        throws ServletException, IOException
    {
          processRequest(request, response);
        }

        protected void doPost(HttpServletRequest request,
                                        HttpServletResponse response)
                                            throws ServletException,
    IOException {
          processRequest(request, response);
        } }
```

JSP і сервлети:

вихідний код JSP має небагато спільного з вихідним кодом сервлета;

Web-додаток автоматично ставить у відповідність будь-якої JSP URL-адрес (ім'я файлу);

Web-контейнер обробляє JSP так само як і сервлет, але неявно;

за допомогою JSP простіше генерувати текстові документи;

JSP призначені для використання тільки з протоколом HTTP;

JSP згодом компілюється у сервлет.

Типи структурних конструкцій JSP:

1. Елементи – динамічні частини сторінки, що інтерпретуються Web-контейнером і визначають дії, які виконуються при генерації HTTP-відповіді:

елементи сценарію;

директиви;

дії.

2. Дані шаблону – текст, розташований на сторінці і призначений для безпосередньої передачі «клієнтові».

Елементи сценарію:

блоки Java-коду, які додаються до тексту шаблону;

типи елементів сценарію:

1. Скриптлети – забезпечують метод для вставки довільного Java-коду між фрагментами тексту шаблону:

<% .....%>

2. Вирази – забезпечують метод для вставки обчислюваних виразів у текст шаблону:

<%=.....%>

3. Декларації – подібні скриптлетам, але вставляються в тіло класу результуючого сервлета поза його методами:

<%!.....%>

Директиви:

це інформація для Web-контейнера, що впливає на загальну структуру результуючого сервлета;

не посилають дані «клієнтові»;

узагальнений синтаксис директив:

<%@ page | include | taglib {attribute="value"}* %>

деякі атрибути директиви **page**:

import – опис типів, доступних в елементах сценарію;

contentType – опис типу документу, що посилається «клієнтові», та кодування символів;

errorPage – задає JSP-сторінку, яка оброблює виключення, що генеруються на поточній сторінці.

Дії:

забезпечують метод зв'язування динамічного коду з текстом шаблону;

типи дій:

Стандартні.

Користувальницькі.

узагальнений синтаксис дій:

<prefix:element {attribute="value"}* />

деякі стандартні дії:

<jsp:include/> – підключення ресурсів під час виконання;

<jsp:forward/> – перенаправлення запиту до іншого ресурсу додатка.

Порядок виконання лабораторної роботи

Загальні рекомендації

При розробці графічного інтерфейсу користувача слід використовувати відповідні візуальні засоби, що належать до складу середовища NetBeans.

До звіту повинна додаватися електронна документація до програми у форматі HTML.

Інформація про сервлети, JSP, організацію взаємодії з базою даних [6 – 8].

Постановка завдання

Розробити Web-додаток, що становить персональний Web-сайт студента.

Web-додаток повинен розміщуватися на локальному Web-сервері Apache Tomcat (інстальований разом з NetBeans).

Додаток має включати HTML-сторінки (Web-інтерфейс клієнта), сервлет і сторінку JSP (бізнес-логіка сайта), а також взаємодіяти з локальною базою даних, створеною за допомогою СУБД Microsoft Access.

Загальний порядок функціонування Web-сайта:

1. Користувач завантажує головну Web-сторінку сайта.

2. З головної сторінки користувач переходить по посиланню "Залишити відгук" на HTML-сторінку з формою, заповнює дані і натискає кнопку відправки даних:

дані відгуку відправляються на Web-сервер і записуються в базу даних;

користувач може натиснути кнопку очищення форми. В цьому випадку передача даних на Web-сервер не відбувається.

3. З головної сторінки користувач переходить по посиланню "Переглянути відгуки". В результаті всі дані відгуків витягуються з бази даних і у вікні браузера з'являється HTML-сторінка з таблицею, в якій знаходяться всі відгуки відвідувачів сайта.

Функціональність Web-додатка і його складові частини:

Web-інтерфейс сайта повинен складатися з:

1. Головної Web-сторінки, що містить:

заголовок сторінки;

фотографію студента;

інформаційну частину (текст і графічні зображення);

посилання на Web-сторінку з HTML-формою (зразкова назва посилання "Залишити відгук");

посилання із приблизною назвою "Переглянути відгуки";

відомості про автора і дату створення сторінки;

іншу інформацію за бажанням студента.

2. Web-сторінки з формою для заповнення відгуку відвідувача про сайт, що містить:

заголовок сторінки;

прізвище (текстова мітка і відповідне їй текстове поле);

ім'я (текстова мітка і відповідне їй текстове поле);

стать (текстові мітки і радіокнопки);

вік (текстова мітка і випадний список з декількома заданими значеннями);

відгук (текстова мітка і випадний список з декількома заданими значеннями);

кнопку відправки даних форми (при натисненні на цю кнопку повинен викликатися сценарій JavaScript для перевірки введення обов'язкових для заповнення даних форми);

кнопку очищення даних форми (при натисненні на цю кнопку повинен викликатися сценарій JavaScript для очищення даних форми).

3. Взаємних посилань «головна сторінка – сторінка з формою».

Бізнес-логіка сайту

Сервлет, що витягує дані форми з HTTP-запиту і записує їх до таблиці бази даних:

при успішному додаванні даних до таблиці сервлет у HTTP-запиті Web-серверу повинен генерувати HTML-сторінку з підтвердженням запису даних;

при помилці додавання даних до таблиці сервлет у HTTP-відповіді Web-серверу повинен генерувати HTML-сторінку з описом помилки.

JSP, призначена для вибірки всіх записів з таблиці бази даних і генерації в HTTP-відповіді Web-серверу такої HTML-сторінки:

дані, вибрані з таблиці бази даних, повинні бути представлені у вигляді HTML-таблиці;

якщо в таблиці бази даних немає записів, то JSP в HTTP-відповіді Web-серверу повинна генерувати HTML-сторінку з відповідним повідомленням;

при помилці вибірки даних з бази JSP повинна в HTTP-відповіді Web-серверу генерувати HTML-сторінку з описом помилки.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Призначення мови HTML та її недоліки.
2. Призначення каскадних таблиць стилів.
3. Метод GET протоколу HTTP та особливості його застосування.
4. Метод POST протоколу HTTP та особливості його застосування.
5. Призначення сценаріїв JavaScript.
6. Яким чином виконуються сценарії JavaScript?
7. Використання сценаріїв JavaScript щодо перевірки даних форми.
8. Поняття про платформу J2EE.
9. Призначення та особливості сервлетів.
10. Призначення та особливості JSP.
11. Методи "життєвого циклу" сервлета.
12. "Життєвий цикл" JSP.
13. Поняття Web-додатка.
14. Структура Web-додатка.
15. Поняття про дескриптор розгортання Web-додатка.
16. Зв‘язок між сервлетами та JSP.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Брюс Эккель. Философия Java: Пер. с англ – СПб.: БХВ-Петербург, 2003. – 788 с.

2. Електронна документація з Java SE – JDK 5.0 Documentation // // http://www.java.sun.com

3. Електронний підручник компанії Sun Microsystems "The Java Tutorial" // http://www.java.sun.com;

4. Патрик Ноутон Java 2: Пер. с англ. Патрик Ноутон, Герберт Шилдт. – СПб.: БХВ-Петербург, 2000. – 1072 с.

5. Портянкин И. А. Swing: эффективные пользовательские интерфейсы. – СПб.: Питер, 2005. – 524 с.

6. Фленаган Д. Java в примерах. Справочник: Пер. с англ. 2-е изд. – СПб: Символ-плюс, 2003. – 664 с.

7. Холл М. Сервлеты и JSP: Пер. с англ. – СПб.: Питер, 2001. – 496 с.

8. Falkner J., Jones K. Servlets and Java Server Pages. – Addison-Wesley, 2005. – 760 p.

9. www.java.sun.com – сайт компанії Sun Microsystems.

10. www.netbeans.org – сайт проекту NetBeans.

11. www.javaworld.com – сайт журналу Java-розробників.

12. www.javable.com – документація з Java-технологій.

13. www.intuit.ru – Internet-інститут інформаційних технологій.

НАВЧАЛЬНЕ ВИДАННЯ

Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "INTERNET-ТЕХНОЛОГІЇ ТА МОВА JAVA"

для студентів напряму підготовки "Комп'ютерні науки" всіх форм навчання

Укладачі: **Парфьонов Юрій Едуардович**
Жукарєв Володимир Юрійович
Поляков Андрій Олександрович

Відповідальний за випуск **Пономаренко В. С.**

Редактор **Лященко Т. О.**

Коректор **Чистякова А. В.**

План 2008 р. Поз. №277.
Підп. до друку Формат 60 × 90 1/16. Папір MultiCopy. Друк Riso.
Ум.-друк. арк. 5,25. Обл.-вид. арк. 6,04. Тираж прим. Зам. №

Свідоцтво про внесення до Державного реєстру суб'єктів видавничої справи ***Дк №481 від 13.06.2001 р.***

Видавець і виготівник — видавництво ХНЕУ, 61001, м. Харків, пр. Леніна, 9а