Міністерство освіти і наук України

Національний університет “Львівська політехніка”

Кафедра ЕОМ

КЛАСИ ТА ПАКЕТИ

Методичні вказівки
до лабораторної роботи № 2 з курсу “Кросплатформні засоби програмування”
для студентів базового напряму 6.123 - “Комп’ютерна інженерія”

Затверджено
на засіданні кафедри
”Електронні обчислювальні машини”
Протокол № 1 від 28.08.2023 р.

Львів – 2023

Класи та пакети: Методичні вказівки до лабораторної роботи № 2 з курсу “Кросплатформні засоби програмування” для студентів базового напряму 6.123 - “Комп’ютерна інженерія” / Укладач: Олексів М.В. – Львів: Національний університет “Львівська політехніка”, 2023, 23 с.

Укладач Олексів М. В., к.т.н.

Рецензент

Відповідальний за випуск: Мельник А. О., професор, завідувач кафедри

КЛАСИ ТА ПАКЕТИ

Мета: ознайомитися з процесом розробки класів та пакетів мовою Java.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Класи

Мова Java є повністю об’єктно-орієнтованою мовою програмування, тому вона дозволяє писати програми лише з використанням об’єктно-орієнтованих парадигм програмування, що базуються на понятті класів.

Синтаксис оголошення простого класу в мові Java має наступний вигляд:

```
[public] class НазваКласу
{
	[конструктори]
	[методи]
	[поля]
}
```

Приклад оголошення загальнодоступного класу:

```
public class StartClass
{
	public StartClass()
	{
		str = "Hello";
	}
	public StartClass(String initString)
	{
		str = initString;
	}

	public void showMessage()
	{
		System.out.print(str);
	}

	private String str;
}
```

Необов’язковий специфікатор доступу `public` робить клас загальнодоступним. У кожному файлі з кодом програми може бути лише один загальнодоступний клас, ім’я якого співпадає з назвою файлу, та безліч класів без специфікатора `public`.

Створення об’єкту класу складається з двох етапів: оголошення та ініціалізації посилання на об’єкт. Оголошення посилання на об’єкт класу має синтаксис:

```
НазваКласу назваПосилання;
```

Приклад оголошення посилання на об’єкт класу `StartClass`:

```
StartClass obj;
```

Ініціалізація посилання на об’єкт класу здійснюється за допомогою оператора `new` і вказування конструктора, який має збудувати об’єкт. Одержаний в результаті цих операцій об’єкт розташується у області оперативної пам’яті що зветься *”куча”*. Ініціалізація посилання на об’єкт класу за допомогою конструктора за замовчуванням має такий синтаксис:

```
назваПосилання = new НазваКонструтора();
```

Приклад ініціалізації посилання на об’єкт класу `StartClass`:

```
obj = new StartClass();
```

При створенні об’єктів дозволяється суміщати оголошення та ініціалізацію об’єктів, а також створювати анонімні об’єкти. Якщо посилання на об’єкт не посилається на жоден об’єкт, то йому слід присвоїти значення `null`. На відміну від полів-посилань на об’єкти, локальні змінні-посилання на об’єкти не ініціалізуються значенням `null` при оголошенні. Для них ініціалізацію посилання слід проводити явно.

Методи

Метод – функція-член класу, яка призначена маніпулювати станом об’єкту класу. Методи можуть бути перевантаженими. Перевантаження методів відбувається шляхом вказування різної кількості параметрів та їх типів методам з однаковими назвами. Синтаксис оголошення методу наступний:

```
[СпецифікаторДоступу] [static] [final] Тип назваМетоду([параметри]) [throws класи]
{
	[Тіло методу]
	[return [значення]];
}
```

Конструктори, методи, та поля класу можуть бути відкритими (`public`), закритими (`private`) та захищеними (`protected`), що визначається специфікатором доступу.

Специфікатор доступу `public` робить елемент класу загальнодоступним в межах пакету (набору класів, з яких складається програма).

Специфікатор доступу `private` робить елемент класу закритим (недоступним) для всіх зовнішніх відносно даного класу елементів програми (включаючи похідні класи).

Специфікатор доступу `protected` робить елемент класу закритим (недоступним) для всіх зовнішніх відносно даного класу елементів програми, проте цей елемент буде загальнодоступним для похідних класів.

Якщо будь-який елемент класу не має специфікатора доступу, то цей елемент автоматично стає відкритим та видимим у межах пакету (не плутати з `public`).

Всі елементи класу, що оголошені без використання ключового слова `static`, належать об’єкту класу. Тобто, кожен об’єкт класу містить власну копію цих елементів класу. Ключове слово `static` робить поле або метод членом класу, а не об’єкту, тобто вони є спільними для всіх об’єктів класу. Оскільки клас існує завжди, на відміну від об’єктів, які створюються в процесі роботи програми, то статичні елементи класу доступні

навіть тоді, коли ще не створено жодного об'єкту класу. Цей підхід використовується при написанні методу `main` з якого починається виконання консольної програми, бо на момент її запуску ще не існує жодного об'єкту.

Метод може генерувати виключну ситуацію. Якщо виключна ситуація не опрацьовується у тілі методу, то вона повинна бути описана в оголошені методу після ключового слова `throws`. Якщо виключна ситуація опрацьовується у тілі методу, то цього робити не потрібно.

Передача параметрів у метод відбувається по значенню шляхом копіювання значень реальних параметрів у формальні параметри методу. Якщо ці значення є простими типами, то відбудеться копіювання значень. Якщо ці значення є посиланнями, то копіюватимуться не об'єкти, а посилання на об'єкти. Таким чином зміна значення посилання формального параметру в середині методу не вплине на значення посилання за його межами.

Вихід та повернення значення з методу відбувається за допомогою оператора `return`. Якщо метод не повертає значення, то оператор `return` можна опустити. Перед поверненням з методу значення об'єкту, що може змінювати свій стан, слід обов'язково скористатися методом `clone` об'єкту, який створює його копію.

Синтаксис виклику нестатичного методу:

```
НазваОб'єкту.назваМетоду([параметри]);
```

Синтаксис виклику статичного методу має 2 види:

```
НазваОб'єкту.назваМетоду([параметри]); // через об'єкт класу
НазваКласу.назваМетоду([параметри]);   // через назву класу
```

Конструктори

Конструктор – спеціальний метод класу, який не повертає значення, має ім'я класу та призначений для початкової ініціалізації об'єктів класу. Синтаксис оголошення конструктора:

```
[СпецифікаторДоступу] НазваКласу([параметри])
{
      Тіло конструктора
}
```

Конструкторів може бути кілька. Конструктор без параметрів називається *конструктором за замовчуванням*. Якщо у класі не визначено жодного конструктора, то конструктор за замовчуванням генерується автоматично при компіляції (неявно). Він здійснює ініціалізацію полів об'єкту класу *значеннями за замовчуванням* (оскільки поля об'єкту на момент створення обов'язково мають бути ініціалізованими). Ці значення рівні:

- для полів чисел простих типів – `0`;
- для логічних полів – `false`;
- для посилань на об'єкти – `null`.

Якщо ж клас має хоч один конструктор з параметрами, то конструктор за замовчуванням має бути визначений явно.

У мові Java допускається виклик одного конструктора з іншого конструктора. Для цього у тілі першого конструктора першим оператором має бути виклик іншого конструктора:

```
this(сигнатура необхідного конструктора);
```

Наприклад:

```
public class StartClass
{
	public StartClass(String initString)
	{
		str = initString;
	}

	public StartClass(int val)
	{
		this("Hello User " + val);
	}

	public void showMessage()
	{
		System.out.print(str);
	}

	private String str;
}
```

Поля

Поле (властивість) – це дані-члени класу, що призначені для зберігання стану об'єкту. Поле може бути статичним (в цьому випадку воно називається *полем класу*), незмінним (*константне поле*), простим типом чи об'єктом та мати різні рівні доступу, що визначаються специфікатором доступу. Допускається ініціалізація поля в місці оголошення. Синтаксис оголошення поля наступний:

```
[СпецифікаторДоступу] [static] [final] Тип НазваПоля [= ПочатковеЗначення];
```

Приклад оголошення поля:

```
private int i;
```

Приклад оголошення константного поля:

```
private final int i;
```

Ініціалізацію полів при створенні об'єкту можна здійснювати трьома способами:

- у конструкторі;
- явно при оголошені поля;
- у блоці ініціалізації (виконується перед виконанням конструктора).

Якщо поле не ініціалізується жодним з цих способів, то йому присвоюється значення за замовчуванням.

Приклад ініціалізації поля `str` у конструкторі:

```
public class StartClass
{
      public StartClass()
      {
            str = "Hello";
      }
      ...
      private String str;
}
```

Приклад явної ініціалізації поля `str` константою при оголошенні:

```
public class StartClass
{
      ...
      private String str = "Hello";
}
```

Приклад явної ініціалізації поля `str` статичним методом при оголошенні:

```
public class StartClass
{
      ...
      static String assignUser()
      {
            String tmpStr = "Hello user " + nextId++;
            return tmpStr;
      }
      ...
      private String str = assignUser ();
      private static int nextId;
}
```

Приклад ініціалізації поля `str` у блоці ініціалізації:

```
public class StartClass
{
      ...
      private String str;

      // блок ініціалізації
      {
            str = "Hello";
      }
}
```

Ініціалізацію полів класу (статичних полів) можна здійснювати двома способами:

- явно при оголошені поля класу;
- у статичному блоці ініціалізації.

Статичний блок ініціалізації виконується коли клас завантажується вперше. Якщо поле класу не ініціалізується жодним з цих способів, то йому присвоюється значення за замовчуванням.

Приклад явної ініціалізації поля класу `str` константою при оголошенні:

```
public class StartClass
{
      ...
      private static String str = "Hello";
      ...
}
```

Приклад ініціалізації поля класу `str` у статичному блоці ініціалізації:

```
public class StartClass
{
      ...
      private String str;

      // статичний блок ініціалізації
      static
      {
            str = "Hello";
      }
}
```

Пакети

Пакет – це механізм мови Java, що дозволяє об'єднувати класи в простори імен. Об'єднання класів в пакети дозволяє відділяти класи, що розроблені одними розробниками, від класів, що розроблені іншими розробниками, забезпечуючи тим самим унікальність імен класів в межах програми та усуваючи можливі конфлікти імен класів. Пакети можуть бути вкладеними одні в одних, утворюючи цим самим ієрархії пакетів. Будь-який зв'язок між вкладеними пакетами відсутній. Всі стандартні пакети належать ієрархіям `java` і `javax`, наприклад, `java.lang`, `java.util`, `java.net` тощо.

Створення пакетів

Створення пакетів відбувається за допомогою оператора `package` з вказуванням назв пакету і під пакетів (за необхідності), що розділені крапкою. Оператор `package` вказується на початку тексту програми перед операторами `import` та визначенням класу. Синтаксис оператора `package`:

```
package НазваПакету{.НазваПідпакету};
```

Приклад:

```
package mypack.subpack;
```

```
public class StartClass
{
      ...
}
```

Якщо оператор `package` в файлі з кодом програми не вказаний, то класи, що описані в цьому файлі розміщуються у *пакеті за замовчуванням*. Пакет за замовчуванням не має імені.

Використання пакетів вимагає, щоб файли і каталоги проекту та їх ієрархія були строго структурованими. Так назви пакету і його підпакетів мають співпадати з назвами каталогів, де вони розміщуються. Назви загальнодоступних класів мають співпадати з назвами файлів, де вони розміщуються. Ієрархія каталогів і файлів проекту має співпадати з ієрархією пакетів. Після компіляції ієрархія каталогів, де містяться файли класів, співпадає з ієрархією каталогів проекту. Наприклад, для пакету:

```
package mypack.subpack;

public class StartClass
{
      ...
}
```

ієрархія каталогів і файлів буде наступною:

. (кореневий каталог проекту)
- (файли пакету за замовчуванням)
- mypack/
    -- subpack/
        --- StartClass.java
        --- StartClass.class

У будь-якому випадку, використовуємо ми пакети чи ні, для компіляції проекту необхідно перейти в кореневий каталог проекту (в той, що містить каталог кореневого пакету) та викликати компілятор `javac`, передавши йому в параметрах шлях до файлу з методом `main` програми. Якщо програма використовує інші пакети, то, завдяки строгій ієрархії каталогів пакетів, шлях до їх файлів, що містять код відповідних класів, буде визначений компілятором автоматично. Наприклад, для компіляції згаданого вище пакету слід виконати команду:

```
javac mypack/subpack/StartClass.java
```

Для запуску програми на виконання слід в параметрах команди `java` передати повну назву пакету і клас, який слід запустити на виконання. Наприклад, для запуску згаданого вище пакету слід виконати команду:

```
java mypack.subpack.StartClass
```

Зверніть увагу, що компілятор працює з *файлами*, а інтерпретатор з *класами*.

Використання пакетів

Клас може використовувати всі класи з власного пакету і всі *загальнодоступні класи* з інших пакетів. Доступ до класів з інших пакетів можна отримати двома шляхами:

1. вказуючи повне ім'я пакету перед іменем кожного класу, наприклад,

```
java.util.Date today = new java.util.Date();
```

2. використовуючи оператор `import`, що дозволяє підключати як один клас так і всі загальнодоступні класи пакету, позбавляючи необхідності записувати імена класів з вказуванням повної назви пакету перед ними.

Оператор `import` слід розміщувати в коді програми після оператора `package` та перед оголошенням класів.

Для *підключення одного загальнодоступного класу пакету* необхідно за допомогою оператора `import` через крапку вказати повну ієрархію пакету та назву класу, який має бути імпортовано, наприклад,

```
import java.util.Date
Date today = new Date();
```

Для *підключення всех загальнодоступних класів пакету* необхідно за допомогою оператора `import` через крапку вказати повну ієрархію пакету та символ зірочка (*), наприклад,

```
import java.util.*
Date today = new Date();
```

Який би з шляхів ви не обрали на розмір скомпільованої програми це не вплине. Однак слід зауважити, що оператор `import` з зірочкою можна застосовувати для імпортування лише одного пакету. Не можна використовувати, наприклад: "`import java.*`, `import java.*.*`", - щоб імпортувати всі пакети, що містять префікс `java`. В більшості випадків імпортується весь пакет не залежно від його розміру. Єдина ситуація, коли не можна імпортувати весь пакет – це конфлікт імен. Конфлікт імен виникає тоді, коли в пакетах, що підключаються, є класи з однаковими іменами. В цій ситуації, при одночасному використанні конфліктуючих класів, слід явно вказувати якому з пакетів він належить:

```
import java.util.*;
import java.sql.*;

java.util.Date today = new java.util.Date();
java.sql.Date dbToday = new java.sql.Date();
```

Якщо ж в програмі використовується клас, що належить тільки одному з кількох пакетів, то можна обмежитися явним вказуванням в операторі `import` приналежності класу потрібному пакету:

```
import java.util.*;
import java.sql.*;
import java.util.Date;

Date today = new Date();
```

Статичний імпорт пакетів

Починаючи з Java SE 5.0 у мову додано можливість імпортувати статичні методи і поля класів. Для цього при підключені пакету слід вжити ключове слово `static` та вказати назву пакету, або назву пакету класу та статичного методу чи поля, які ви хочете підключити:

```
import static НазваПакету{.НазваПідпакету}.НазваКласу.
    НазваСтатичногоМетодуАбоПоля;
import static НазваПакету{.НазваПідпакету}.*;
```

Наприклад,

```
// підключити тільки java.lang.Math.sqrt
import static java.lang.Math.sqrt;
// підключити всі статичні члени пакету java.lang
import static java.lang.*;
```

Статичний імпорт дозволяє не вживати явно назву класу при звертанні до статичного поля або методу класу, наприклад:

```
sqrt(pow(x, 3));        //  замість Math.sqrt(Math.pow(x, 3));

dow = d.get(DAY_OF_WEEK); //  замість dow = d.get(Calendar.DAY_OF_WEEK);
```

КОНТРОЛЬНІ ПИТАННЯ

1. Синтаксис визначення класу.
2. Синтаксис визначення методу.
3. Синтаксис оголошення поля.
4. Як оголосити та ініціалізувати константне поле?
5. Які є способи ініціалізації полів?
6. Синтаксис визначення конструктора.
7. Синтаксис оголошення пакету.
8. Як підключити до програми класи, що визначені в зовнішніх пакетах?
9. В чому суть статичного імпорту пакетів?
10. Які вимоги ставляться до файлів і каталогів при використанні пакетів?

ЛІТЕРАТУРА

1. Schildt H. Java: The Complete Reference, 12th Edition. / Herbert Schildt. – McGraw Hill, 2021. – 1280 p.
2. Java SE Documentation at a Glance [електронний ресурс]. – Режим доступу до документації: http://www.oracle.com/technetwork/java/javase/documentation/index.html

ЗАВДАННЯ

1. Написати та налагодити програму на мові Java, що реалізує у вигляді класу предметну область згідно варіанту. Програма має задовольняти наступним вимогам:
   - програма має розміщуватися в пакеті `Група.Прізвище.Lab2`;
   - клас має містити мінімум 3 поля, що є об'єктами класів, які описують складові частини предметної області;
   - клас має містити кілька конструкторів та мінімум 10 методів;
   - для тестування і демонстрації роботи розробленого класу розробити клас-драйвер;
   - методи класу мають вести протокол своєї діяльності, що записується у файл;
   - розробити механізм коректного завершення роботи з файлом (не надіятися на метод `finalize()`);
   - програма має володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.
2. Автоматично згенерувати документацію до розробленої програми.
3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.
4. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.
5. Дати відповідь на контрольні запитання.

ВАРІАНТИ ЗАВДАННЬ

1. Людина
2. Космічний корабель
3. Пес
4. Кіт
5. Машина
6. Літак
7. Комп'ютер
8. Фотоапарат
9. Рослина
10. Будинок
11. Монітор
12. Водойма
13. Телефон
14. Телевізор
15. Корабель
16. Аудіоплеєр
17. Відеоплеєр
18. Сканер
19. Принтер
20. Взуття
21. Пістолет
22. Автомат
23. Плитка для приготування їжі
24. Спорядження альпініста
25. Кондиціонер
26. Шлюпка на веслах
27. Патрон
28. Лампочка
29. Протигаз
30. Локомотив

ПОРЯДОК ВИКОНАННЯ

1. Запустити на виконання середовище Eclipse IDE.
2. Дослідити тестову програму, що наведена в методичних вказівках.
3. Почати створення власного проекту. Для цього слід викликати підпункут меню File->New->Project… У вікні, що відкриється слід вибрати Java Project (рис. 1) та натиснути кнопку "Next>".

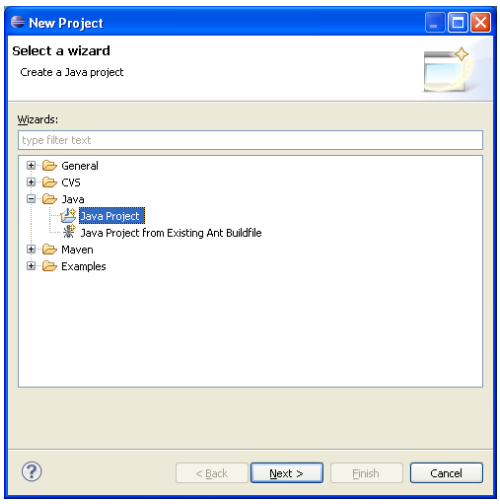

Рис. 1. Діалогове вікно вибору типу проекту в Eclipse IDE.

4. У вікні, що відкрилося, необхідно вказати назву нового проекту, місце розташування проекту, версію JRE на яке орієнтована програма, топологію проекту (Project layout) та натиснути кнопку ”Next>” (рис. 2).

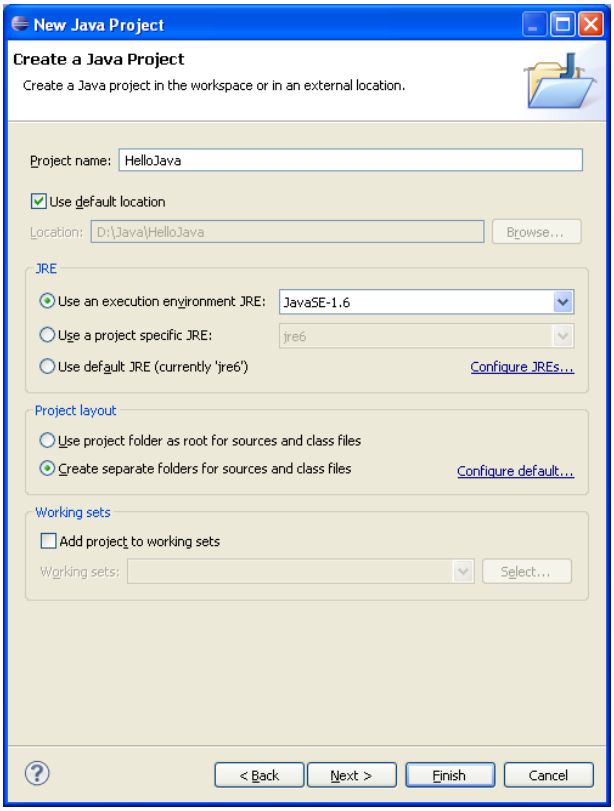

Рис. 2. Діалогове вікно створення нового проекту в Eclipse IDE.

5. У вікні, що відкрилося, натиснути кнопку ”Finish” (рис. 3).

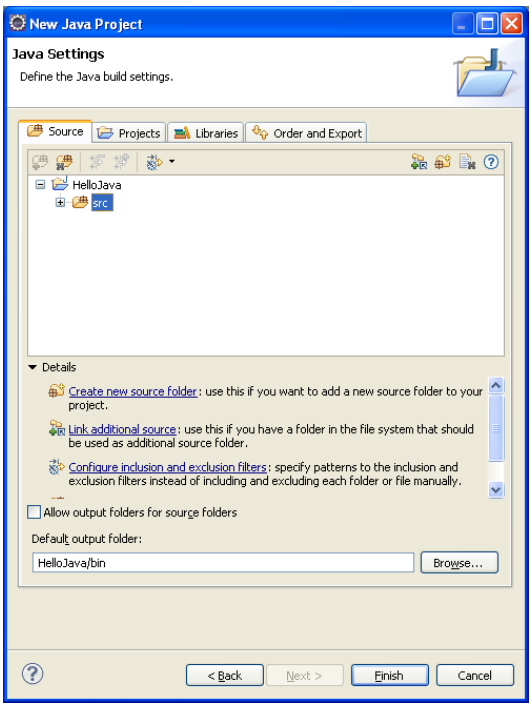

Рис. 3. Діалогове вікно налаштування властивостей побудови нового проекту в Eclipse IDE.

6. У вікні, що з’явилося (рис.4), викликати підпункт меню File->New->Class. У вікні, що відкрилося, слід задати кореневий каталог проекту, назву пакету, назву класу драйвера, що створюється, модифікатор доступу класу, базовий клас, методи, які слід згенерувати автоматично, вказати чи генерувати коментарі автоматично та натиснути кнопку ”Finish” (рис. 5). Аналогічним чином створити клас, що описує предметну область.

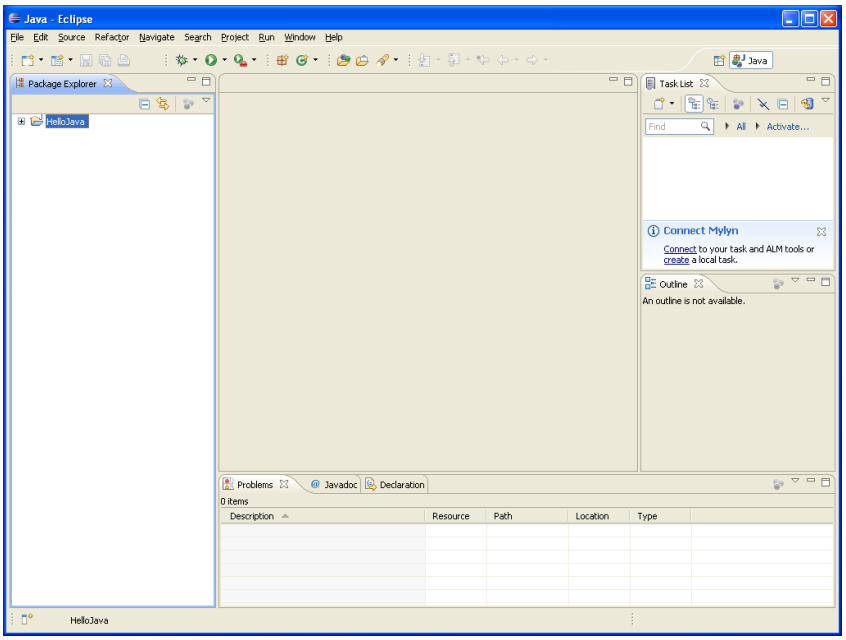

Рис. 4. Середовище Eclipse IDE з відкритим проектом Java.

6. У створеному файлі написати програму згідно завдання.

7. Запустити програму на виконання. Для цього слід вибрати підпункт меню Run-> Run.

8. Встановити точки переривання, запустити налагоджувач (Run->Debug) та покроково дослідити процес виконання програми (рис. 6).

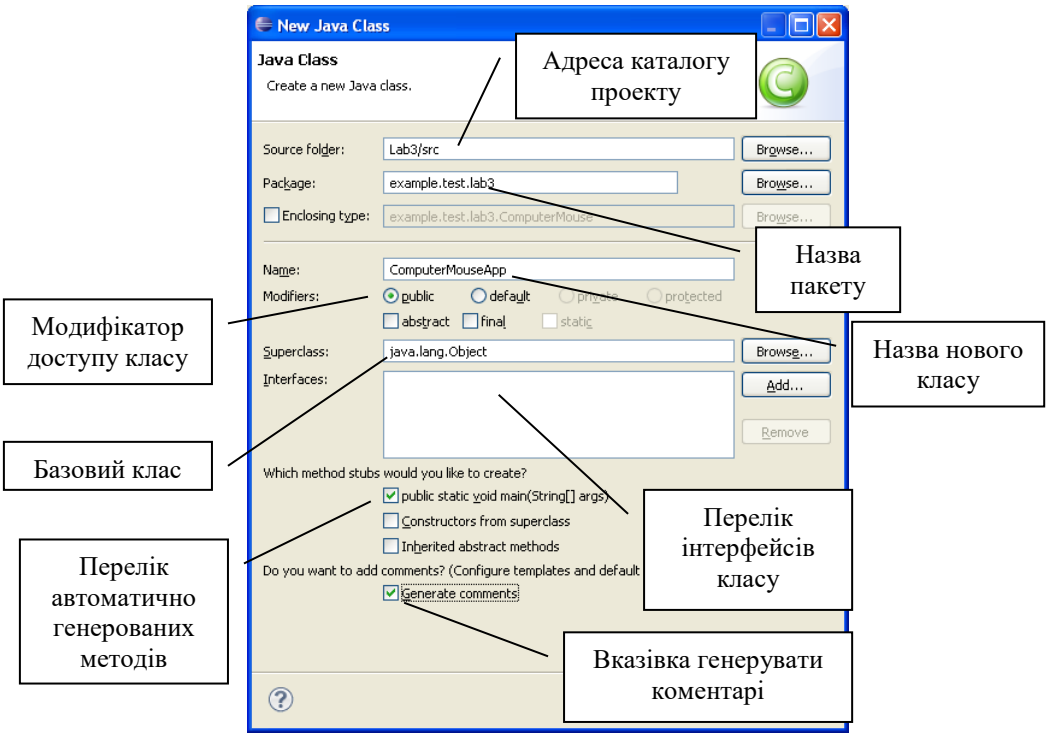

Рис. 5. Створення нового класу з використанням візуального конструктора класів.

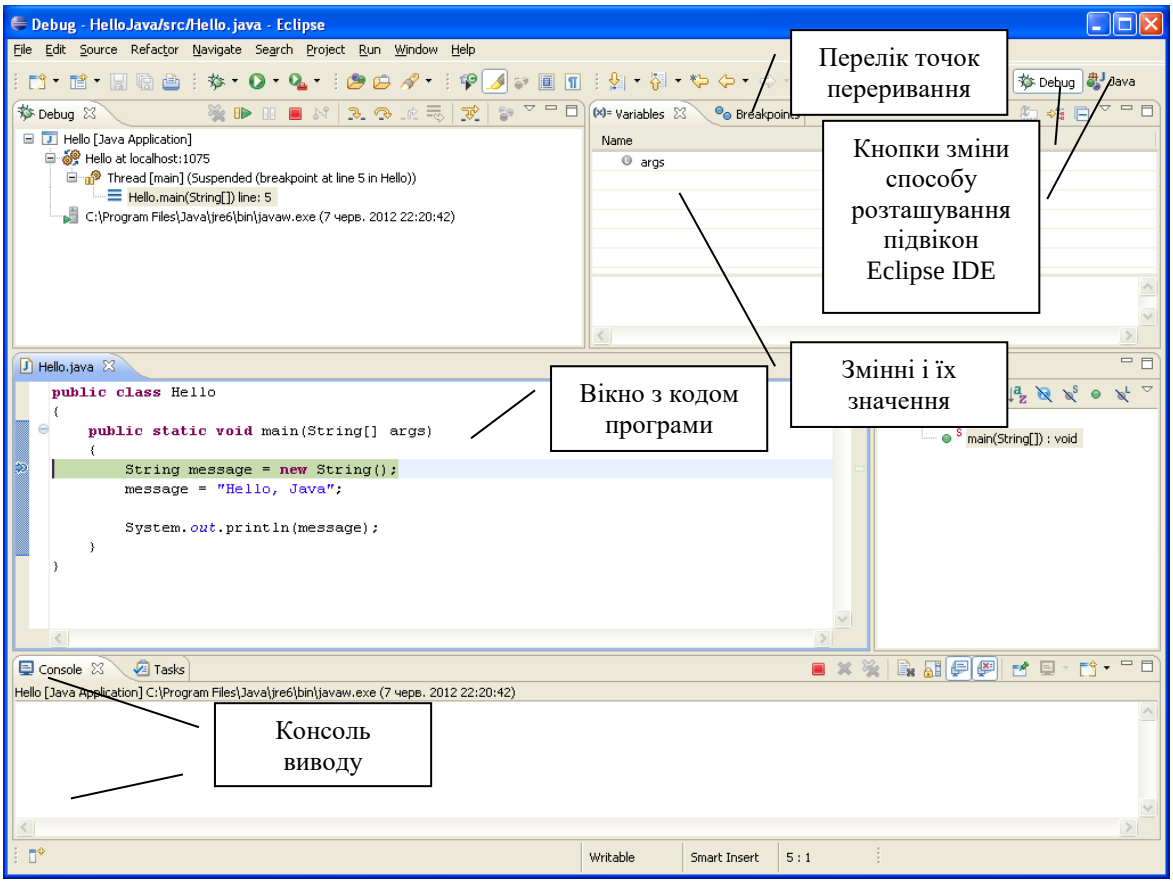

Рис. 6. Приклад вікна налагодження програми.

9. Перебуваючи у підкаталозі \src автоматично згенерувати документацію для всіх загальнодоступних класів у каталог \doc вашого проекту за допомогою команди:

```
javadoc -d ..\doc Група.Прізвище.Lab2
```

Проаналізувати автоматично згенеровану документацію.

ДЕМОНСТРАЦІЙНА ПРОГРАМА

Файл ComputerMouseApp.java

```
/*******************************************************************************
 * Copyright (c) 2013-2023 Lviv Polytechnic National University. All Rights Reserved.
 *
 * This program and the accompanying materials are made available under the terms
 * of the Academic Free License v. 3.0 which accompanies this distribution, and is
 * available at https://opensource.org/license/afl-3-0-php/
 *
 * SPDX-License-Identifier: AFL-3.0
 *******************************************************************************/

/**
 * lab 2 package
 */
package example.test.lab2;

import static java.lang.System.out;
import java.io.*;

/**
 * Computer Mouse Application class implements main method for
 * ComputerMouse class abilities demonstration
*/
public class ComputerMouseApp
{

	/**
	 * @param args
	 * @throws FileNotFoundException
	 */
	public static void main(String[] args) throws FileNotFoundException
	{
		// TODO Auto-generated method stub

		Scroller.Directions dir = null;
		ComputerMouse mouse = new ComputerMouse();

		mouse.clickLeftButton();
		out.print(mouse.getLeftButtonResource() + "\n");
		mouse.clickRightButton();

		out.print(mouse.getRightButtonResource() + "\n");
		mouse.setMousePosition(5, -3);

		mouse.scrollUp();
		dir = mouse.getScrollingDirection();
		if (dir == Scroller.Directions.DOWN)
			out.print ("Down" + "\n");
		else if (dir == Scroller.Directions.UP)
			out.print ("Up" + "\n");
		else
			out.print ("Neutral" + "\n");

		mouse.dispose();
	}

}
```

Файл ComputerMouse.java

```
/*******************************************************************************
 * Copyright (c) 2013-2023 Lviv Polytechnic National University. All Rights Reserved.
 *
 * This program and the accompanying materials are made available under the terms
 * of the Academic Free License v. 3.0 which accompanies this distribution, and is
 * available at https://opensource.org/license/afl-3-0-php/
 *
 * SPDX-License-Identifier: AFL-3.0
 *******************************************************************************/

/**
 * lab 2 package
 */
package example.test.lab2;

import java.io.*;

/**
 * Class <code>ComputerMouse</code> implements computer mouse
 */
public class ComputerMouse {

	private Scroller scrollerDevice;
	private RelativePosition pos;
	private Button rightButton;
	private Button leftButton;
	private PrintWriter fout;

	/**
	 * Constructor
	 * @throws FileNotFoundException
	 */
	public ComputerMouse() throws FileNotFoundException
	{
		scrollerDevice = new Scroller();
		pos = new RelativePosition();
		rightButton = new Button();
		leftButton = new Button();

		fout = new PrintWriter(new File("Log.txt"));
	}

	/**
	 * Constructor
	 * @param <code>resource</code> Button clicks resource
	 * @throws FileNotFoundException
	 */
	public ComputerMouse(int resource) throws FileNotFoundException
	{
		scrollerDevice = new Scroller();
		pos = new RelativePosition();
		rightButton = new Button(resource);
		leftButton = new Button(resource);

		fout = new PrintWriter(new File("Log.txt"));
	}

	/**
	 * Method implements mouse position offset on (xPos, yPos)
	 * @param <code>xPos</code> The X coordinate of the mouse position
	 * @param <code>yPos</code> The Y coordinate of the mouse position
	 */
```

```
	public void moveMouseOnDistance(int xPos, int yPos)
	{
		pos.setXPosition(pos.getXPosition() + xPos);
		pos.setYPosition(pos.getYPosition() + yPos);

		fout.print("New mouse X position: " + pos.getXPosition() + "\n");
		fout.print("New mouse Y position: " + pos.getYPosition() + "\n");
	}

	/**
	 * Method sets the new mouse position
	 * @param <code>xPos</code> The X coordinate of the mouse position
	 * @param <code>yPos</code> The Y coordinate of the mouse position
	 */
	public void setMousePosition(int xPos, int yPos)
	{
		pos.setXPosition(xPos);
		pos.setYPosition(yPos);
	}

	/**
	 * Method returns mouse's current X position
	 * @return Mouse's current X position
	 */
	public int getMouseXPosition()
	{
		return pos.getXPosition();
	}

	/**
	 * Method returns mouse's current Y position
	 * @return Mouse's current Y position
	 */
	public int getMouseYPosition()
	{
		return pos.getYPosition();
	}

	/**
	 * Method simulates mouse's right button clicking
	 */
	public void clickRightButton()
	{
		rightButton.clickButton();

		fout.print("New mouse's right button resource: " +
rightButton.getButtonResourse() + "\n");
		fout.flush();
	}

	/**
	 * Method simulates mouse's left button clicking
	 */
	public void clickLeftButton()
	{
		leftButton.clickButton();

		fout.print("New mouse's left button resource: " +
leftButton.getButtonResourse() + "\n");
		fout.flush();
	}

	/**
	 * Method returns mouse's right button clicking resource
```

```
	 * @return Mouse's right button clicking resource
	 */
	public int getRightButtonResource()
	{
		return rightButton.getButtonResourse();
	}

	/**
	 * Method returns mouse's left button clicking resource
	 * @return Mouse's left button clicking resource
	 */
	public int getLeftButtonResource()
	{
		return leftButton.getButtonResourse();
	}

	/**
	 * Method simulates mouse's scrolling up
	 */
	public void scrollUp()
	{
		scrollerDevice.setUpDirection();

		fout.print("Mouse scrolled up\n");
		fout.flush();
	}

	/**
	 * Method simulates mouse's scrolling down
	 */
	public void scrollDown()
	{
		scrollerDevice.setDownDirection();

		fout.print("Mouse scrolled down\n");
		fout.flush();
	}

	/**
	 * Method simulates mouse's scroller setting in a neutral position
	 */
	public void resetScroller()
	{
		scrollerDevice.resetScrooller();

		fout.print("Mouse scroller in neutral state");
		fout.flush();
	}

	/**
	 * Method returns mouse's scrolling direction
	 * @return Mouse's scrolling direction of
<code>Scroller.Directions</code> type
	 */
	public Scroller.Directions getScrollingDirection()
	{
		return scrollerDevice.getDirection();
	}

	/**
	 * Method releases used recourses
	 */
	public void dispose()
	{
```

```
            fout.close();
      }
}

class Scroller
{
      // Type for scrolling directions
      enum Directions {NEUTRAL, UP, DOWN};

      // current scroller's state
      private Directions direction;

      /**
       * Constructor
       */
      public Scroller()
      {
            direction = Directions.NEUTRAL;
      }

      /**
       * Method sets up scrolling direction
       */
      public void setUpDirection()
      {
            direction = Directions.UP;
      }

      /**
       * Method sets neutral scrolling direction
       */
      public void setNeutralDirection()
      {
            direction = Directions.NEUTRAL;
      }

      /**
       * Method sets down scrolling direction
       */
      public void setDownDirection()
      {
            direction = Directions.DOWN;
      }

      /**
       * Method resets scrolling direction to neutral state
       */
      public void resetScrooller()
      {
            setNeutralDirection();
      }

      /**
       * Method returns scrolling direction
       * @return Scrolling direction of <code>Scrooller.Directions</code> type
       */
      public Directions getDirection()
      {
            return direction;
      }
}
```

```
/**
 * Class <code>RelativePosition</code> implements relative positioning
coordinate system
 *
 * @author EOMStuff
 * @version 1.0
 */
class RelativePosition
{
	// coordinates of the mouse position
	private int x, y;

	/**
	 * Constructor
	 */
	public RelativePosition()
	{
		x = 0;
		y = 0;
	}

	/**
	 * Constructor
	 * @param <code>xPos</code> The X coordinate value
	 * @param <code>yPos</code> The Y coordinate value
	 */
	public RelativePosition(int xPos, int yPos)
	{
		x = xPos;
		y = yPos;
	}

	/**
	 * Method returns the X coordinate value
	 * @return The X coordinate value
	 */
	public int getXPosition()
	{
		return x;
	}

	/**
	 * Method returns the Y coordinate value
	 * @return The Y coordinate value
	 */
	public int getYPosition()
	{
		return y;
	}

	/**
	 * Method returns coordinates of the position in the <code>obj</code>,
that is passed into method through method parameter
	 * @param <code>obj</code> The object, where coordinates of the current
position are set
	 */
	public void getPosition(RelativePosition obj)
	{
		obj.x = x;
		obj.y = y;
	}

	/**
	 * Method sets the X coordinate value
```

```
	 * @param <code>xPos</code> The X coordinate value
	 */
	public void setXPosition(int xPos)
	{
		x = xPos;
	}

	/**
	 * Method sets the Y coordinate value
	 * @param <code>yPos</code> The Y coordinate value
	 */
	public void setYPosition(int yPos)
	{
		y = yPos;
	}

}

/**
 * @author EOMStuff
 * Class <code>Button</code> implements mouse button
 */
class Button
{

	// Button clicks resource
	private int btnResource;

	/**
	 * Constructor
	 */
	public Button()
	{
		btnResource = 1000000;
	}

	/**
	 * Constructor
	 * @param <code>res</code> Button clicks resource
	 */
	public Button(int res)
	{
		btnResource = res;
	}

	/**
	 * Method simulates Button clicking
	 */
	public void clickButton()
	{
		btnResource = btnResource - 1;
	}

	/**
	 * Method returns button clicks resource available
	 * @return Button clicks resource available
	 */
	public int getButtonResourse()
	{
		return btnResource;
	}
}
```

НАВЧАЛЬНЕ ВИДАННЯ

КЛАСИ ТА ПАКЕТИ

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторної роботи № 2 з дисципліни “ Кросплатформні засоби програмування ”
для студентів базового напряму 6.123 - “Комп’ютерна інженерія”

Укладач Олексів М. В., к.т.н.

Редактор

Комп’ютерне верстання

Здано у видавництво . Підписано до друку
Формат 70x100/16. Папір офсетний. Друк на різографі
Умовн. друк. арк. Обл..-вид. арк..
Тираж прим. Зам..

Видавництво Національного університету “Львівська політехніка”
Реєстраційне свідоцтво ДК №751 від 27.12.2001 р.

Поліграфічний центр Видавництва
Національного університету “Львівська політехніка”

Вул.. Ф. Колесси, 2. Львів, 79000