Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ

СПАДКУВАННЯ ТА ІНТЕРФЕЙСИ

Методичні вказівки
до лабораторної роботи № 3 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

Спадкування та інтерфейси: Методичні вказівки до лабораторної роботи № 3 з курсу “Кросплатформні засоби програмування” для студентів базового напряму 6.123 - “Комп’ютерна інженерія” / Укладач: Олексів М.В. – Львів: Національний університет “Львівська політехніка”, 2023, 13 с.

Укладач Олексів М. В., к.т.н.

Рецензент

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

СПАДКУВАННЯ ТА ІНТЕРФЕЙСИ

Мета: ознайомитися з спадкуванням та інтерфейсами у мові Java.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Спадкування

Спадкування в ООП призначене для розширення функціональності існуючих класів шляхом утворення нових класів на базі вже існуючих. У Java реалізована однокоренева архітектура класів згідно якої всі класи мають єдиного спільного предка (кореневий клас в ієрархії класів) – клас `Object`. Решта класів мови Java утворюються шляхом успадковування даного класу. Будь-яке спадкування у мові Java є відкритим, при цьому аналогів захищеному і приватному спадкуванню мови C++ не існує. На відміну від C++ у Java можливе спадкування лише одного базового класу (множинне спадкування відсутнє). Спадкування реалізується шляхом вказування ключового слова `class` після якого вказується назва *підкласу*, ключове слово `extends` та назва *суперкласу*, що розширюється у новому підкласі. Синтаксис реалізації спадкування:

```
class Підклас extends Суперклас
{
     Додаткові поля і методи
}
```

В термінах мови Java базовий клас найчастіше називається *суперкласом*, а похідний клас – *підкласом*. Дана термінологія запозичена з теорії множин, де підмножина міститься у супермножині.

При наслідувані у Java дозволяється перевизначення (перевантаження) методів та полів. При цьому область видимості методу, що перевизначається, має бути не меншою, ніж область видимості цього методу у суперкласі, інакше компілятор видасть повідомлення, про обмеження привілеїв доступу до даних. Перевизначення методу полягає у визначені у підкласі методу з сигнатурою методу суперкласу. При виклику такого методу з-під об’єкта підкласу викличеться метод цього підкласу. Якщо ж у підкласі немає визначеного методу, що викликається, то викличеться метод суперкласу. Якщо ж у суперкласі даний метод також відсутній, то згенерується повідомлення про помилку.

Перевизначення у підкласах елементів суперкласів (полів або методів) призводить до їх приховування новими елементами. Бувають ситуації, коли у методах підкласу необхідно звернутися до цих прихованих елементів суперкласів. У цій ситуації слід використати ключове слово **`super`**, яке вказує, що елемент до якого йде звернення, розташовується у суперкласі, а не у підкласі. Синтаксис звертання до елементів суперкласу:

```
super.назваМетоду([параметри]); // виклик методу суперкласу
super.назваПоля                 // звертання до поля суперкласу
```

Використання ключового слова `super` у конструкторах підкласів має дещо інший сенс, ніж у методах. Тут воно застосовується для виклику конструктора суперкласу. Виклик конструктора суперкласу має бути першим оператором конструктора підкласу. Конкретний конструктор, який необхідно викликати, вибирається по переданим параметрам. Явний виклик конструктора суперкласу часто є необхідним, оскільки підкласи не мають доступу до приватних полів суперкласів. Тож ініціалізація їх полів значеннями відмінними від значень за замовчуванням без явного виклику відповідного конструктора суперкласу є неможливою. Якщо виклик конструктора суперкласу не вказаний явно у підкласі або суперклас не має конструкторів, тоді автоматично викликається конструктор за замовчуванням суперкласу. Синтаксис виклику конструктора суперкласу з конструктора підкласу:

```
public НазваПідкласу([параметри])
{
      super([параметри]);
      оператори конструктора підкласу
}
```

Поліморфізм

Механізм поліморфізму забезпечує можливість присвоєння об'єктним змінним суперкласу об'єктів похідних класів та звертання з-під цих змінних до перевизначених у підкласі членів суперкласу. У Java всі об'єктні змінні є поліморфними. Поліморфізм реалізується за допомогою механізму динамічного (пізнього) зв'язування, який полягає у тому, що вибір методу, який необхідно викликати, відбувається не на етапі компіляції, а під час виконання програми. Для глибшого розуміння поліморфізму розглянемо покроково виклик методу класу:

1. Компілятор визначає об'явлений тип об'єкту і ім'я методу та нумерує всі методи з однаковою назвою у класі та всі загальнодоступні методи з такою ж назвою у його суперкласах.

2. Компілятор визначає типи параметрів, що вказані при виклику методу. Якщо серед усіх методів з вказаним іменем є лише один метод, типи параметрів якого співпадають з вказаним, то відбувається його виклик. Цей процес називається *дозволом перевантаження*. Якщо компілятор не знаходить жодного методу з підходящим набором параметрів або в результаті перетворення типів виявлено кілька методів, що відповідають даному виклику, то видається повідомлення про помилку.

3. Якщо метод є приватним, статичним, фінальним або конструктором, то для нього застосовується механізм *статичного зв'язування*. Механізм статичного зв'язування передбачає визначення методу, який необхідно викликати, на етапі компіляції. В протилежному випадку метод, що необхідно викликати, визначається по фактичному типу неявного параметру (*динамічне зв'язування*).

4. Якщо для виклику методу використовується динамічне зв'язування, то віртуальна машина повинна викликати версію методу, що відповідає фактичному типу об'єкту на який посилається об'єктна змінна.

Оскільки на пошук необхідного методу потрібно багато часу, то віртуальна машина заздалегідь створює для кожного класу таблицю методів, в якій перелічуються сигнатури

всіх методів і фактичні методи, що підлягають виклику. При виклику методу віртуальна машина просто переглядає таблицю методів. Якщо відбувається виклик методу з суперкласу за допомогою ключового слова `super`, то при виклику методу переглядається таблиця методів суперкласу неявного параметру.

Приведення об'єктних типів

Приведення типів у Java відбувається вказуванням у круглих дужках перед змінною, яку необхідно привести до іншого типу, типу до якого її необхідно привести. Синтаксис приведення змінної до іншого типу:

```
(новийТип) змінна
```

У Java усі об'єктні змінні є типізовані. Механізми наслідування і поліморфізму дозволяють створювати нові типи (класи та інтерфейси) на базі вже існуючих та присвоювати об'єкти цих типів посиланням на об'єкти супертипу. В цьому випадку об'єкти підтипів мають ті самі елементи, що й об'єкти супертипу, тож таке висхідне приведення типів є безпечним і здійснюється компілятором автоматично. Проте присвоєння посиланню на об'єкт підтипу об'єкту супертипу не завжди є коректним, тому таке приведення вимагає явного приведення типів. При такому приведенні типів можливі дві ситуації:

- якщо посилання на об'єкт супертипу реально посилається на об'єкт підтипу, то приведення посилання на об'єкт супертипу до типу підтипу є коректним;
- якщо посилання на об'єкт супертипу посилається на об'єкт супертипу, то приведення посилання на об'єкт супертипу до типу підтипу викличе виключну ситуацію `ClassCastExeption`.

Наявність бодай одної такої виключної ситуації призводить до аварійного завершення програми. Щоб уникнути цього слід перед приведенням типів використати оператор `instanceof`, який повертає `true`, якщо посилання посилається на об'єкт фактичний тип якого є не вищим в ієрархії типів, ніж вказаний у операторі `instanceof`, і `false` у протилежному випадку. Синтаксис оператора `instanceof`:

```
посилання instanceof Ім'яТипу
```

Таким чином, основні правила приведення типів є наступними:

1. Приведення типів можна виконувати лише в ієрархії спадкування.
2. Щоби перевірити коректність приведення супертипу до підтипу слід використовувати оператор `instanceof`.

Абстрактні класи

Абстрактні класи призначені бути основою для розробки ієрархій класів та не дозволяють створювати об'єкти свого класу. Вони реалізуються за допомогою ключового слова `abstract`. На відміну від звичайних класів абстрактні класи можуть містити абстрактні методи (а можуть і не містити). *Абстрактні методи* – це методи, що

оголошені з використанням ключового слова `abstract` і не місять тіла. Розширюючи абстрактний клас можна залишити деякі або всі методи невизначеними. При цьому підклас автоматично стане абстрактним. Перевизначення у підкласі усіх абстрактних методів призведе до того, що підклас не буде абстрактним, що дозволить створювати на його основі об'єкти класу. Синтаксис оголошення абстрактного класу наведено в пункті «Класи та об'єкти». Синтаксис оголошення абстрактного методу:

```
[СпецифікаторДоступу] abstract Тип назваМетоду([параметри]);
```

Абстрактні класи корисні тоді, коли в ієрархії класів необхідно реалізувати методи з однаковими назвами, проте різною функціональністю і можливістю поліморфного виклику методів підкласів з-під посилання на абстрактний суперклас. Також використовуються при створенні інтерфейсів.

Інтерфейси

Інтерфейси вказують що повинен робити клас не вказуючи як саме він це повинен робити. Інтерфейси покликані компенсувати відсутність множинного спадкування у мові Java та гарантують визначення у класах оголошених у собі прототипів методів. Синтаксис оголошення інтерфейсів:

```
[public] interface НазваІнтерфейсу
{
      Прототипи методів та оголошення констант інтерфейсу
}
```

Інтерфейси можуть містити прототипи методів, які мають визначатися у класі, що відповідає цьому інтерфейсу, і константи, які автоматично успадковуються класом, що реалізує цей інтерфейс. Всі методи інтерфейсу вважаються загальнодоступними, тому оголошувати методи як `public` у інтерфейсі нема необхідності. Всі поля, що оголошені у інтерфейсі автоматично вважаються такими, що оголошені як `public static final`, тому додавати ці модифікатори самостійно необхідності також нема. Інтерфейси можуть успадковувати інші інтерфейси, утворюючи таким чином ієрархії інтерфейсів. Синтаксис реалізації спадкування у інтерфейсів співпадає з синтаксисом реалізації спадкування у класах.

Оскільки інтерфейс не є класом, то створити його об'єкт за допомогою оператора `new` неможливо. Проте можна оголосити посилання на інтерфейсний тип та присвоїти цьому посиланню об'єкт, що реалізує цей інтерфейс.

На відміну від спадкування клас може реалізувати кілька інтерфейсів. Ця особливість і компенсує відсутність множинного спадкування у мові Java.

Щоб клас реалізував інтерфейс необхідно:

1. Оголосити за допомогою ключового слова `implements`, що клас реалізує інтерфейс. Якщо клас реалізує кілька інтерфейсів, то вони перелічуються через кому після ключового слова `implements`.
2. Визначити у класі усі методи, що вказані у інтерфейсі.

КОНТРОЛЬНІ ПИТАННЯ

1. Синтаксис реалізації спадкування.
2. Що таке суперклас та підклас?
3. Як звернутися до членів суперкласу з підкласу?
4. Коли використовується статичне зв'язування при виклику методу?
5. Як відбувається динамічне зв'язування при виклику методу?
6. Що таке абстрактний клас та як його реалізувати?
7. Для чого використовується ключове слово `instanceof?`
8. Як перевірити чи клас є підкласом іншого класу?
9. Що таке інтерфейс?
10. Як оголосити та застосувати інтерфейс?

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 розширює клас, що реалізований у лабораторній роботі №2, для реалізації предметної області заданої варіантом. Суперклас, що реалізований у лабораторній роботі №2, зробити абстрактним. Розроблений підклас має забезпечувати механізми свого коректного функціонування та реалізовувати мінімум один інтерфейс. Програма має розміщуватися в пакеті `Група.Прізвище.Lab3` та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.
2. Автоматично згенерувати документацію до розробленого пакету.
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. Спортсмен
2. Багаторазовий космічний корабель
3. Піддослідний пес
4. Піддослідний кіт
5. Вантажна машина
6. Бомбардувальник
7. Ноутбук
8. Цифрова відеокамера
9. Дерево
10. Офісний центр
11. Сенсорний екран
12. Море
13. Мобільний телефон
14. Телевізор з тюнером
15. Фрегат
16. Диктофон
17. Відеомагнітофон
18. Копіювальний апарат
19. Багатофункціональний пристрій
20. Чоботи
21. Водяний Пістолет
22. Штурмова гвинтівка
23. Газова плитка
24. Спорядження військового альпініста
25. Пристрій кліматконтролю
26. Моторний човен
27. Інтелектуальний патрон
28. Енергозберігаюча лампочка
29. Протигаз командира
30. Електричка

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>”.

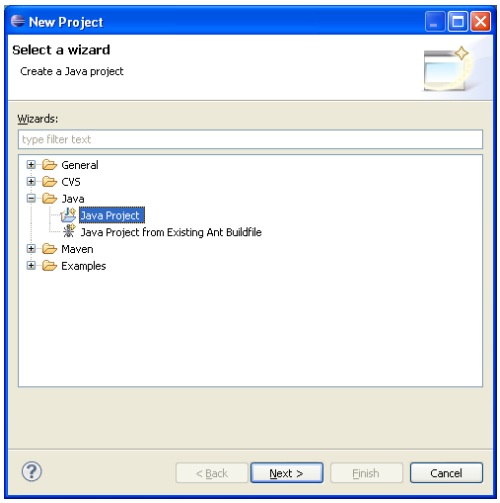

Рис. 1. Діалогове вікно вибору типу проекту в Eclipse IDE.

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).

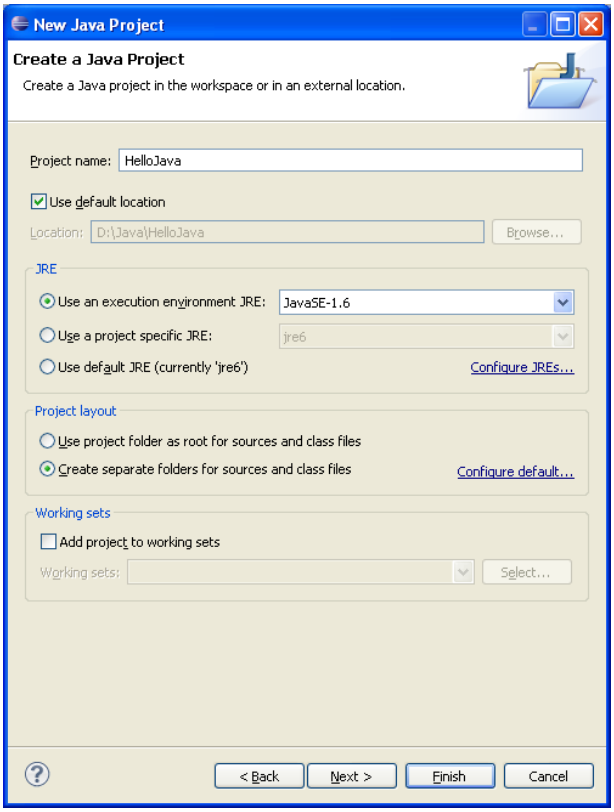

Рис. 2. Діалогове вікно створення нового проекту в Eclipse IDE.

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

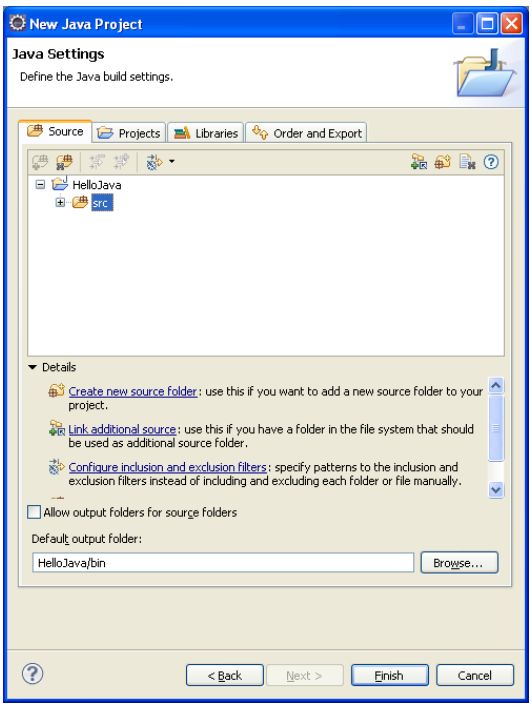

Рис. 3. Діалогове вікно налаштування властивостей побудови нового проекту в Eclipse IDE.

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.

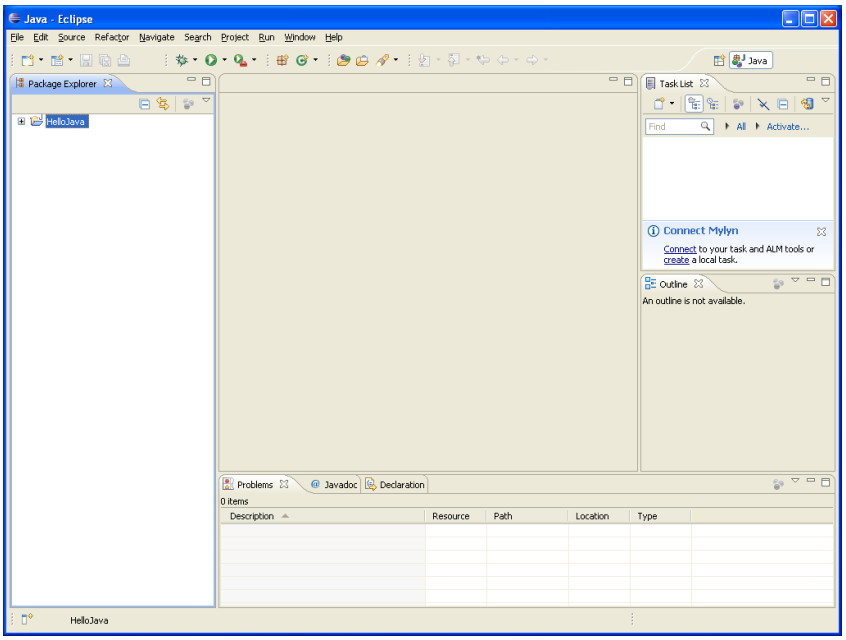

Рис. 4. Середовище Eclipse IDE з відкритим проектом Java.

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).

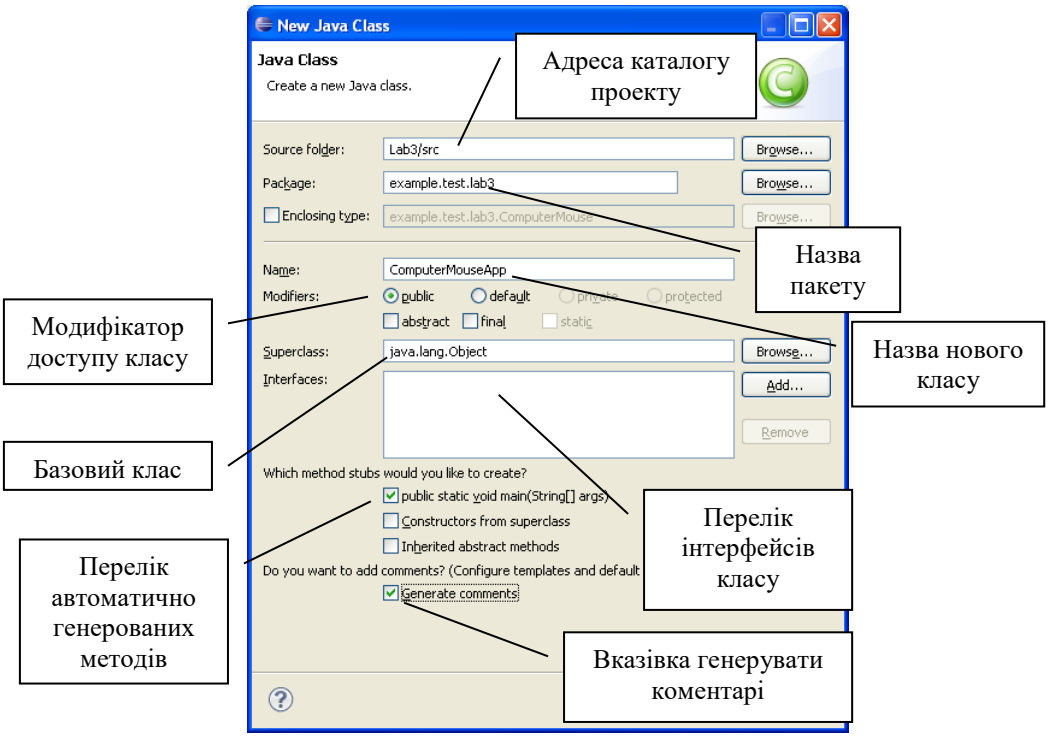

Рис. 5. Створення нового класу з використанням візуального конструктора класів.

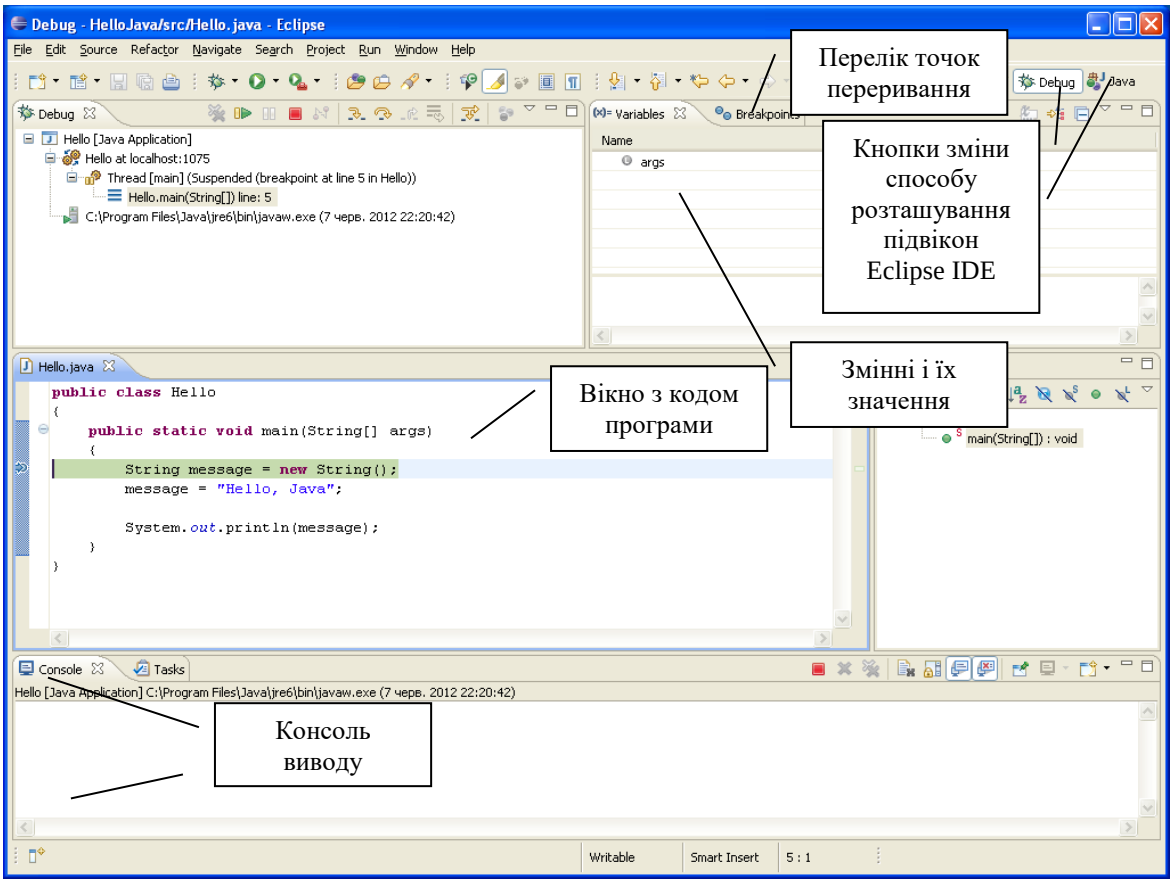

Рис. 6. Приклад вікна налагодження програми.

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.Lab3
```

Проаналізувати автоматично згенеровану документацію.

ПРИКЛАД ПРОГРАМИ

Файл Car.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
 *****************************************************************************/

// Інтерфейс Moveable містить методи для реалізації руху
interface Moveable
{
	void move (double x);	// прототип методу
}

// Інтерфейс Powered розширює інтерфейс Moveable
interface Powered extends Moveable
{
	double milesToFueling (); // прототип методу
	int SPEEDLIMIT = 100;	// константа
}

// Клас Car реалізує інтерфейс Powered
public class Car extends Object implements Powered
{
	public Car (double lFuel, double lMilesPerGalon)
	{
		fuel = lFuel;
		milesPerGalon = lMilesPerGalon;
		distance = 0.0;
	}

	private Car ()
	{
		fuel = 0.0;
		milesPerGalon = 0.0;
		distance = 0.0;
	}

	public void move (double x)
	{
		distance = distance + x;
		fuel = fuel - distance / milesPerGalon;
	}

	public double getDistance()
	{
		return distance;
	}

	public double milesToFueling ()
	{
		return fuel*milesPerGalon;
	}
```

```
        private double distance;
        private final double milesPerGalon;
        private double fuel;
}
```

НАВЧАЛЬНЕ ВИДАННЯ

СПАДКУВАННЯ ТА ІНТЕРФЕЙСИ

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторної роботи № 3 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