Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ

ПАРАМЕТРИЗОВАНЕ ПРОГРАМУВАННЯ

Методичні вказівки
до лабораторної роботи № 6 з курсу “ Кросплатформенні 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

Параметризоване програмування: Методичні вказівки до лабораторної роботи № 6 з курсу “Кросплатформні засоби програмування” для студентів базового напряму 6.123 - “Комп’ютерна інженерія” / Укладач: Олексів М.В. – Львів: Національний університет “Львівська політехніка”, 2023, 16 с.

Укладач Олексів М. В., к.т.н.

Рецензент

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

ПАРАМЕТРИЗОВАНЕ ПРОГРАМУВАННЯ

Мета: оволодіти навиками параметризованого програмування мовою Java.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Параметризоване програмування є аналогом шаблонів у C++. Воно полягає у написанні коду, що можна багаторазово застосовувати з об'єктами різних класів. Користувачів параметризованого програмування можна поділити на 3 рівні кваліфікації:

1. ті, що користуються готовими класами;
2. ті, що користуються готовими класами і вміють виправляти помилки, що виникають при їх використанні;
3. ті, що пишуть власні параметризовані класи.

Для успішного застосування параметризованого програмування слід навчитися розуміти помилки, що генерує середовище при компіляції програми, що можуть стосуватися, наприклад, неоднозначності визначення спільного суперкласу для всіх переданих об'єктів. З іншої сторони необхідно передбачити захист від присвоєння об'єктів параметризованого класу, що містять об'єкти підкласу об'єктам параметризованого класу, що містять об'єкти суперкласу і дозволити зворотні присвоєння. Для вирішення цієї проблеми у мові Java введено так звані *підстановочні типи*. Це далеко не всі «підводні камені», що виникають при застосуванні параметризованого програмування.

Визначення простого параметризованого класу

Параметризований клас – це клас з однією або більше змінними типу.

Синтаксис оголошення параметризованого класу:

```
[public] class НазваКласу <параметризованийТип{,параметризованийТип}>
{…}
```

Іменувати змінні параметризованих типів прийнято великими літерами. У бібліотеці Java використовується літера `E` для позначення типу колекції, `K` і `V` – для типів ключа і значення таблиці, `T`, `U`, `S` та сусідні літери – для позначення довільних типів.

Приклад оголошення параметризованого класу:

```
class GenericClass<T, U>
{
      public GenericClass(T first, U second)
      {
            this.first = first;
            this.second = second;
      }
      public void setFirst(T first)
      {
            this.first = first;
      }
```

```
        public T getFirst()
        {
              return first;
        }
        ...
        private T first;
        private U second;
  }
```

Тут T і U – це змінні параметризованих типів, що використовуються по всьому тілу класу для специфікації типу повернення методів, типів полів і локальних змінних. При створенні об'єкту параметризованого класу замість них підставляються реальні типи, що визначаються в трикутних дужках у місці створення об'єкту параметризованого класу.

Синтаксис створення об'єкту параметризованого класу:

```
НазваКласу < перелікТипів > = new НазваКласу < перелікТипів > (параметри);
```

Приклад створення об'єкту параметризованого класу:

```
GenericClass<String, Integer> obj = new GenericClass<String, Integer> ();
```

Параметризовані методи

Параметризовані методи визначаються в середині як звичайних класів так і параметризованих. На відміну від звичайних методів параметризовані методи мають параметризований тип, що дозволяє за їх допомогою опрацьовувати різнотипні набори даних. Реальні типи для методів, як і для класів, визначаються у місці виклику методу шляхом передачі реального типу у трикутних дужках.

Синтаксис оголошення параметризованого методу:

```
Модифікатори <параметризованийТип{,параметризованийТип}> типПовернення
назваМетоду(параметри);
```

Приклад оголошення параметризованого методу:

```
class ArrayAlg
{
      public static<T> T getMiddle(T[] a)
      {
            return a[a.length / 2];
      }
}
```

Синтаксис виклику параметризованого методу:

```
(НазваКласу|НазваОб'єкту).[<перелікТипів>] НазваМетоду(параметри);
```

У мові Java компілятор здатний самостійно визначати типи, що підставляються замість параметризованих типів, тому у трикутних дужках вказувати реальні типи не

обов’язково. Проте це може призвести до помилок, якщо компілятор не зможе однозначно визначити єдиний супертип для всіх параметрів.

Приклад виклику параметризованого методу:

```
String[] names = {“Ivan”, “Ivanovych”, “Ivanov”};

String middle = ArrayAlg.<String>getMiddle(names);
```

або

```
String middle = ArrayAlg.getMiddle(names);
```

Приклад виклику параметризованого методу, що може призвести до помилок, оскільки параметри методу можна привести як до класу `Double` так і до інтерфейсу `Comparable`:

```
ArrayAlg.<String>getMiddle(3.75, 123, 0);
```

Встановлення обмежень для змінних типів

Бувають ситуації, коли клас або метод потребують накладення обмежень на змінні типів. Наприклад, може бути ситуація, коли метод у процесі роботи викликає з-під об’єкта параметризованого типу метод, що визначається у деякому інтерфейсі. У такому випадку немає ніякої гарантії, що цей метод буде реалізований у кожному класі, що передається через змінну типу. Щоб вирішити цю проблему у мові Java можна задати обмеження на множину можливих типів, що можуть бути підставлені замість параметризованого типу. Для цього після змінної типу слід використати ключове слово `extends` і вказати один суперклас, або довільну кількість інтерфейсів (через знак `&`), від яких має походити реальний тип, що підставляється замість параметризованого типу. Якщо одночасно вказуються інтерфейси і суперклас, то суперклас має стояти першим у списку типів після ключового слова `extends`.

Синтаксис оголошення параметризованого методу з обмеженнями типів:

```
Модифікатори <параметризований тип extends обмежуючийТип {& обмежуючий тип}
{, параметризований тип extends обмежуючийТип {& обмежуючий тип} }>
типПовернення назваМетоду(параметри);
```

Приклад оголошення параметризованого методу з обмеженнями типів:

```
class ArrayCmp
{
      public static <T extends Comparable & Serializable> T CopmlComp(T[] a)
      {
            ...
      }
}
```

Правила спадкування параметризованих типів

1. *Всі класи, що утворені з одного і того ж параметризованого класу з використанням різних значень змінних типів є незалежними* навіть якщо між цими типами є залежність спадкування. Тобто наступний код є некоректним:

```
class SupClass {…}
class SubClass extends SupClass {}
Pair<SubClass> subObj = new Pair<SubClass>(…);
// помилка. subObj і supObj посилаються на незалежні об'єкти
Pair<SupClass> supObj = subObj;
```

2. *Завжди можна перетворити параметризований клас у «сирий» клас*, при роботі з яким захист від некоректного коду є значно слабшим, що дозволяє здійснювати небезпечні присвоєння об'єктів параметризованого класу об'єктам «сирого» класу. Проте у цьому випадку можна зробити помилки, які генеруватимуть виключення на етапі виконання програми:

```
Pair<SubClass> subObj = new Pair<SubClass>(…);
Pair rawObj = subObj;               // OK
rawObj.setFirst(new File(…));       // лише попередження при компіляції
```

3. *Параметризовані класи можуть розширювати або реалізовувати інші параметризовані класи*. В цьому відношенні вони не відрізняються від звичайних класів. Наприклад, `ArrayList<T>` реалізує інтерфейс `List<T>`. Це значить, що `ArrayList<SubClass>` можна перетворити у `List<SubClass>`. Але `ArrayList<SubClass>` це не `ArrayList<SupClass>` і не `List<SupClass>`, де `SubClass` – підклас суперкласу `SupClass`.

Підстановочні типи

Підстановочні типи були введені у мову Java для збільшення гнучкості жорсткої існуючої системи параметризованих типів. На відміну від неї підстановочні типи дозволяють враховувати залежності між типами, що виступають параметрами для параметризованих типів. Це в свою чергу дозволяє застосовувати обмеження для параметрів, що підставляються замість параметризованих типів. Завдяки цьому підвищується надійність параметризованого коду, полегшується робота з ним та розділяється використання безпечних методів доступу і небезпечних модифікуючих методів. Підстановочні типи застосовуються у вигляді параметру типу, що передається у трикутних дужках при утворені реального типу з параметризованого типу, наприклад, у методі `main`.

Підстановочні типи дозволяють реалізувати:

1. обмеження підтипу;
2. обмеження супертипу;
3. необмежені підстановки.

Обмеження підтипу – дозволяє позначити будь-який параметризований тип, чий параметр типу є типом або підтипом вказаного у параметрі типу, що дозволяє одержувати результати роботи методів параметризованого типу, але не передавати параметри методам, що приймають параметри параметризованого типу. Це відбувається тому, що компілятор не здатний вивести з "?" конкретний тип параметру, але гарантує, що цей тип буде типом або підтипом вказаного у параметрі типу.

Обмеження супертипу – дозволяє позначити будь-який параметризований тип, чий параметр типу є класом або суперкласом вказаного у параметрі типу, що дозволяє передавати параметри методам, що приймають параметри параметризованого типу. Це відбувається тому, що компілятор хоч і не здатний вивести з "?" конкретний тип параметру, що передається у метод, але він може безпечно привести передане значення до будь-якого з супертипів. При одержанні результатів роботи методів параметризованого типу нема ніякої гарантії стосовно типу результату, тому результат роботи можна присвоїти лише типу `Object`.

Синтаксис встановлення обмеження підтипу для підстановок має такий вигляд:

```
НазваПараметризованогоТипу <? extends НазваТипуОбмеження>
```

Синтаксис встановлення обмеження супертипу для підстановок має такий вигляд:

```
НазваПараметризованогоТипу <? super НазваТипуОбмеження >
```

Приклад використання обмеження підтипу для підстановок:

```
public static void main ()
{     …
      Pair <? extends List> lst = new Pair<ArrayList<Integer>>();
}
```

Синтаксис *необмежених підстановок* має такий вигляд:

```
НазваПараметризованогоТипу <?>
```

Цей запис означає, що параметризований тип `НазваПараметризованогоТипу` можне приймати параметр будь-якого типу. Але **сама по собі підстановка "?" не є змінною типу, тому ми не можемо використати "?" як тип**.

Таким чином ми не можемо написати:

```
public static void someMethod(Pair<?> p)
{
      ? t = p.otherMethod();
      …
}
```

Щоб вийти з цієї ситуації застосовують техніку, що називається *захоплення підстановок*. Вона полягає у оголошенні допоміжного параметризованого методу (метод

someMethod не є узагальненим методом, оскільки він має фіксований параметр типу Pair<?>) , який буде викликатися з методу someMethod:

```
public static void someMethod(Pair<?> p)
{
      HelperMethod(p);
}

public static <T> void HelperMethod (Pair<T> p)
{
      T t = p.otherMethod();
      …
}
```

В цьому випадку кажуть, що параметр T методу HelperMethod захоплює підстановку.

Захоплення підстановки не є необхідним для реалізації всіх методів, що використовують необмежену підстановку. Воно не потрібне при реалізації методів, яким для роботи не потрібно знати реальний тип підстановки, наприклад:

```
public static boolean hasNulls (Pair<?> p)
{
      return p.getFirst() == null || p.getSecond() == null;
}
```

КОНТРОЛЬНІ ПИТАННЯ

1. Дайте визначення терміну «параметризоване програмування».
2. Розкрийте синтаксис визначення простого параметризованого класу.
3. Розкрийте синтаксис створення об’єкту параметризованого класу.
4. Розкрийте синтаксис визначення параметризованого методу.
5. Розкрийте синтаксис виклику параметризованого методу.
6. Яку роль відіграє встановлення обмежень для змінних типів?
7. Як встановити обмеження для змінних типів?
8. Розкрийте правила спадкування параметризованих типів.
9. Яке призначення підстановочних типів?
10. Застосування підстановочних типів.

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. Створити параметризований клас, що реалізує предметну область задану варіантом. Клас має містити мінімум 4 методи опрацювання даних включаючи розміщення та виймання елементів. Парні варіанти реалізують пошук мінімального елементу, непарні – максимального. Написати на мові Java та налагодити програму-драйвер для розробленого класу, яка мстить мінімум 2 різні класи екземпляри яких розмішуються у

екземплярі розробленого класу-контейнеру. Програма має розміщуватися в пакеті `Група.Прізвище.Lab6` 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. Масив
2. Однозв'язний список
3. Побутовий пакет
4. Конвеєр
5. Двозв'язний список
6. Шафа
7. Трюм корабля
8. Коробка
9. Споруда
10. Пенал
11. Торговий тентр
12. Грузова машина
13. Словник (тип даних)
14. Поличка
15. Вагон
16. Земельна ділянка
17. Кошик
18. Відсік для зброї
19. Сейф
20. Стек
21. Бак для сміття
22. Валіза
23. Хлів
24. Коробка для інструментів
25. Сховище товарів
26. Банка
27. Шухляда
28. Антресолі
29. Тумбочка
30. Множина

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>".

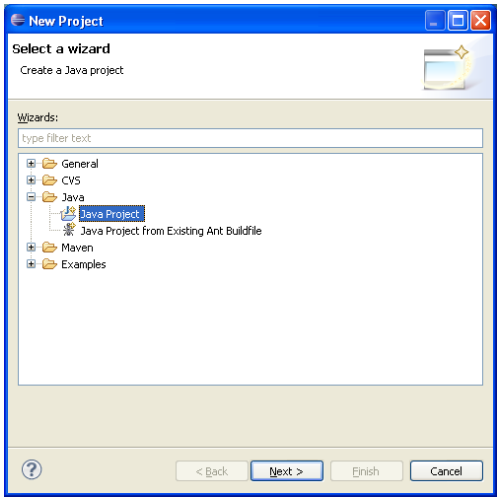

Рис. 1. Діалогове вікно вибору типу проекту в Eclipse IDE.

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).

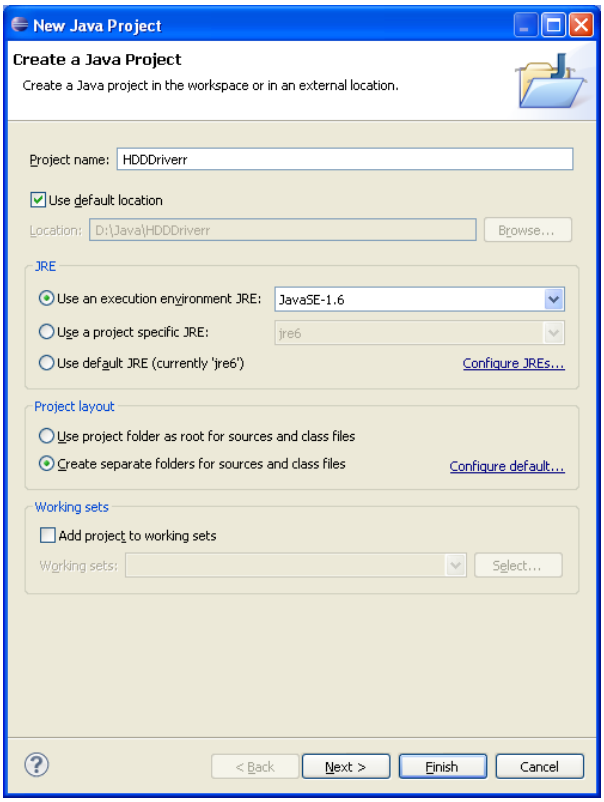

Рис. 2. Діалогове вікно створення нового проекту в Eclipse IDE.

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

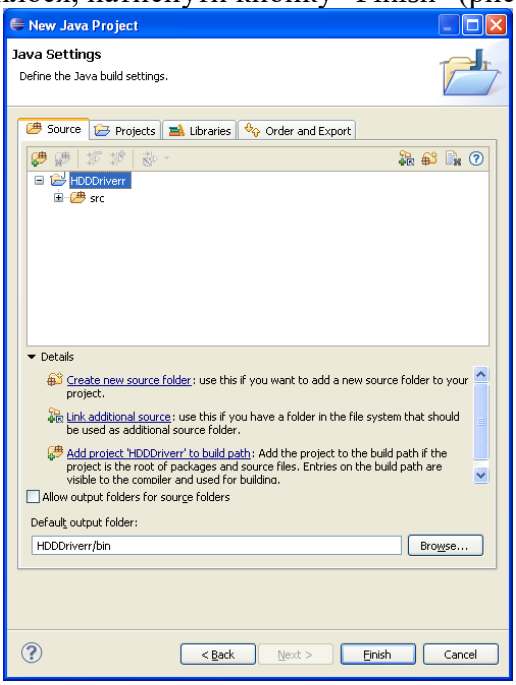

Рис. 3. Діалогове вікно налаштування властивостей побудови нового проекту в Eclipse IDE.

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 реалізує завдання.

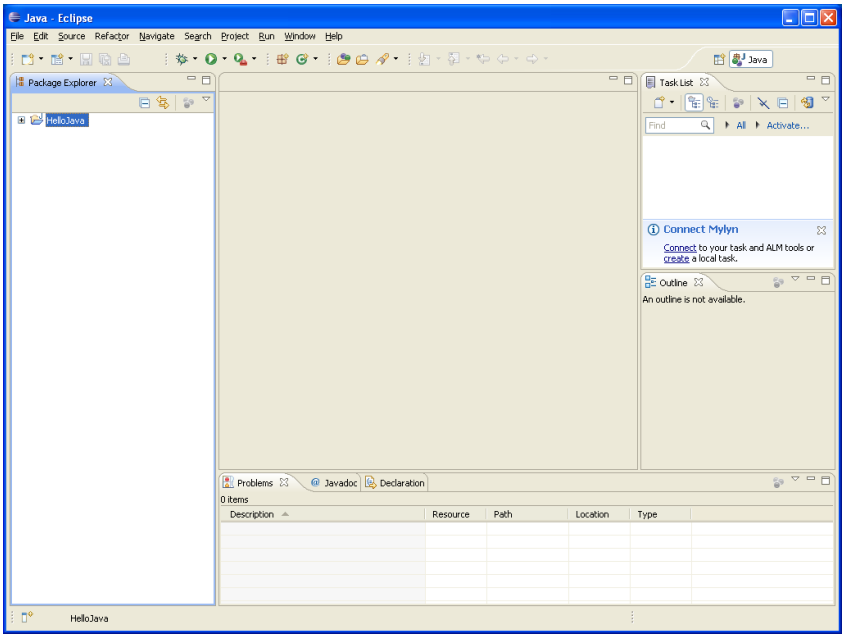

Рис. 4. Середовище Eclipse IDE з відкритим проектом Java.

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).

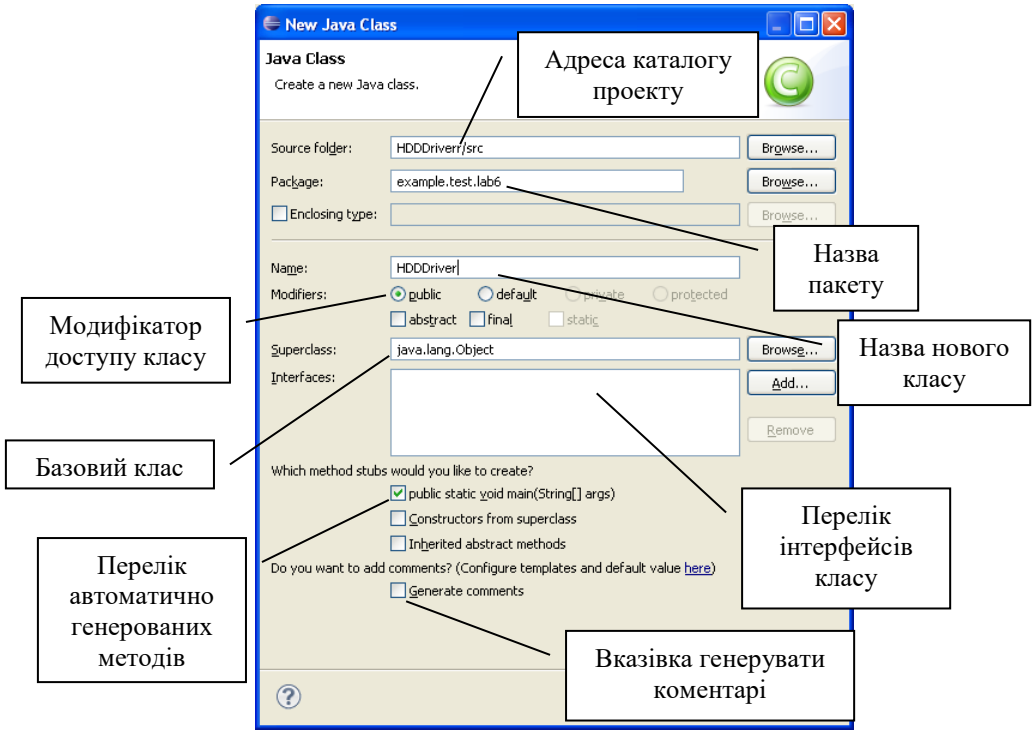

Рис. 5. Створення нового класу з використанням візуального конструктора класів.

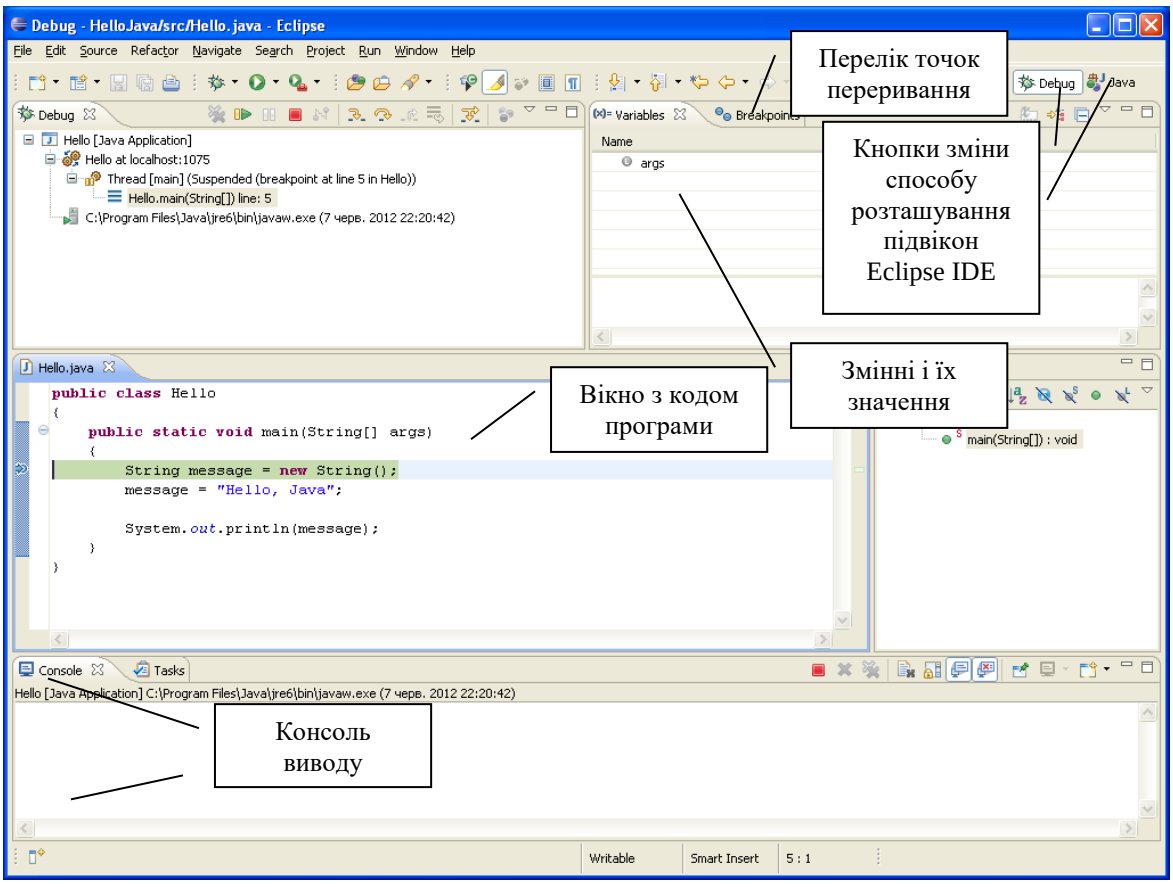

Рис. 6. Приклад вікна налагодження програми.

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.Lab6
```

Проаналізувати автоматично згенеровану документацію.

ПРИКЛАД ПРОГРАМИ

Файл HDDDriver.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
 *******************************************************************************/

package example.test.lab6;

import java.util.*;
import java.io.*;

public class HDDDriver {

	public static void main(String[] args)
	{
		HDD <? super Data> hdd = new HDD <Data>();
```

```
		hdd.AddData(new Program("Windows", 0, 852124325));
		hdd.AddData(new Photo("Weekend"  , 32568741));
		hdd.AddData(new Photo("People"   , 2547814));
		hdd.AddData(new Program("JRE"    , 0, 67894257));

		Data res = hdd.findMax();
		System.out.print("The greatest data on HDD is: \n");
		res.print();
	}
}
class HDD <T extends Data>
{
	private ArrayList<T> arr;

	public HDD()
	{
		arr = new ArrayList<T>();
	}

	public T findMax()
	{
		if (!arr.isEmpty())
		{
			T max = arr.get(0);
			for (int i=1; i< arr.size(); i++)
			{
				if ( arr.get(i).compareTo(max) > 0 )
					max = arr.get(i);
			}
			return max;
		}
		return null;
	}

	public void AddData(T data)
	{
		arr.add(data);
		System.out.print("Element added: ");
		data.print();
	}
	public void DeleteData(int i)
	{
		arr.remove(i);
	}
}

interface Data extends Comparable<Data>
{
	public int getSize();
	public void print();
}

class Program implements Data
{
	private String progName;
	private int usedTimes;
	private int size;

	public Program(String pName, int pUsed, int pSize)
```

```
	{
		progName = pName;
		usedTimes = pUsed;
		size = pSize;
	}

	public String getProgramName()
	{
		return progName;
	}

	public void setProgramName(String name)
	{
		progName = name;
	}

	public int getUsedTimes()
	{
		return usedTimes;
	}

	public void setUsedTimes(int n)
	{
		usedTimes = n;
	}

	public int getSize()
	{
		return size;
	}

	public int compareTo(Data p)
	{
		Integer s = size;
		return s.compareTo(p.getSize());
	}

	public void print()
	{
		System.out.print("Program: " + progName + ", Times used: " + usedTimes +
", Program Size: " + size + ";\n");
	}
}

class Photo implements Data
{
	private String photoFileName;
	private int photoFileSize;

	public Photo(String pName, int pSize)
	{
		photoFileName = pName;
		photoFileSize = pSize;
	}

	public String getName()
	{
		return photoFileName;
	}
```

```
	public void getFileName(String name)
	{
		photoFileName = name;
	}

	public void SetSize(int n)
	{
		photoFileSize = n;
	}

	public int getSize()
	{
		return photoFileSize;
	}

	public int compareTo(Data p)
	{
		Integer s = photoFileSize;
		return s.compareTo(p.getSize());
	}

	public void print()
	{
		System.out.print("Photo File Name: " + photoFileName + ", Photo File
Size: " + photoFileSize + ";\n");
	}
}
```

НАВЧАЛЬНЕ ВИДАННЯ

ПАРАМЕТРИЗОВАНЕ ПРОГРАМУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторної роботи № 6 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