

**Міністерство освіти і науки України**  
**Національний університет “Львівська політехніка”**



Кафедра ЕОМ

**ДОСЛІДЖЕННЯ БАЗОВИХ КОНСТРУКЦІЙ МОВИ JAVA**

**Методичні вказівки**  
до лабораторної роботи № 1 з курсу “Кросплатформні засоби  
програмування”  
для студентів базового напрямку 6.123 - “Комп’ютерна інженерія”

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

Львів – 2023

**Дослідження базових конструкцій мови Java:** Методичні вказівки до лабораторної роботи № 1 з курсу “ Кросплатформні засоби програмування ” для студентів базового напрямку 6.123 - “Комп’ютерна інженерія” / Укладач: Олексів М.В. – Львів: Національний університет “Львівська політехніка”, 2023, 18 с.

**Укладач**

Олексів М. В., к.т.н

**Рецензент**

**Відповідальний за випуск:**

Мельник А. О., професор, завідувач  
кафедри

# ДОСЛІДЖЕННЯ БАЗОВИХ КОНСТРУКЦІЙ МОВИ JAVA

**Мета:** ознайомитися з базовими конструкціями мови Java та оволодіти навиками написання й автоматичного документування простих консольних програм мовою Java.

## ТЕОРЕТИЧНІ ВІДОМОСТІ

### Автоматичне документування

При автоматичній генерації документації використовується утиліта `javadoc`, яка аналізує вміст між `/**` і `*/` та на його базі генерує документацію у форматі `*.html`. Коментарі між `/**` і `*/` прийнято починати з описового тексту, за яким слідує дескриптори. Використання дескрипторів полегшує як автоматичну генерацію документації, так і розуміння коду, до якого відноситься коментар. Дескриптор, на відміну від решти коментарів, починається з символу `@` за яким слідує ім'я дескриптора. Оскільки документація генерується у форматі `*.html`, то між `/**` і `*/` допускається розташування `html`-тегів, включаючи рисунки.

Для автоматичної генерації документації між `/**` і `*/` можна розмістити:

- коментарі до класу;
- коментарі до методів;
- коментарі до полів;
- загальні коментарі.

**Коментарі до класу** мають бути розміщені після директив `import` безпосередньо перед визначенням класу. Найчастіше цей коментар має вигляд одного або кількох коротких речень:

```
/**
    Об'єкт класу <code>Person</code> описує особу.
    Особа має властивості: ім'я, прізвище та стать.
*/
```

**Коментарі до методів** розташовуються безпосередньо перед методами, які вони описують. Крім дескрипторів загального призначення для коментування методів використовуються дескриптори:

- `@param` змінна опис

Цей дескриптор додає в опис методу розділ `"parameters"`. Опис цього елементу може складатися з кількох рядків та містити `html`-теги. Всі дескриптори `@param`, що відносяться до одного методу слід групувати разом.

- `@return` опис

Цей дескриптор додає в опис методу розділ `"returns"`. Опис цього елементу може складатися з кількох рядків та містити `html`-теги.

- `@throws` опис\_класу

Цей дескриптор додає в опис методу інформацію про класи об'єкти яких можуть генеруватися при виключних ситуаціях. Відомості про кожен клас слід описувати в окремому дескрипторі `@throws`.

**Коментарі до полів (властивостей)** застосовуються, якщо поля є загальнодоступними:

```
/**
    Чоловіча стать
 */
public static final int SEX=1;
```

**Загальні коментарі** відображають інформацію загального характеру за допомогою дескрипторів:

- `@author ім'я`

Цей дескриптор створює розділ “author” у якому відображаються відомості про авторів. Відомості про кожного з авторів записуються в окремих дескрипторах.

- `@version текст`

Цей дескриптор створює розділ “version” у якому відображається інформація про версію (переважно її номер).

- `@since текст`

Цей дескриптор створює розділ “since” у якому відображається інформація про версію у якій вперше появилася дана властивість. Наприклад,

```
@since version 1.4.4
```

- `@deprecated текст`

Цей дескриптор інформує про те, що дана властивість чи метод застарілі, та часто містить інформацію про те, що слід застосовувати замість них. Наприклад,

```
@deprecated Use <code>SetVisible(true)</code> instead.
```

- `@see посилання`

Цей дескриптор створює розділ “See also”. Використовується як для класів так і для методів. Посилання записується у одному з трьох можливих виглядів:

```
пакет.клас#елемент мітка
<a href="..."> мітка </a>
"текст"
```

В першому варіанті елемент класу, на який відбувається посилання, відділяється від класу, в якому він міститься знаком #. Мітка використовується як якор для посилання і є необов'язковою. При її відсутності якорем для посилання буде ім'я коду або URL. У третьому варіанті текст, який є між подвійними лапками відобразиться в розділі “See also”.

- Спеціальний дескриптор `{link пакет.клас#елемент мітка}` дозволяє включати гіпертекстові посилання в будь-якому місці коментарю. Його застосування підпорядковується таким же самим правилам, що й `@see`.

Для генерування документації по пакету слід ввести в консолі ОС Windows:

```
javadoc -d каталог_doc ім'я_пакету
```

Опція `-d каталог_doc` задає каталог, де слід розмістити згенеровану документація до пакету.

### **Основні типи мови Java**

Мова Java є строго типізованою. Це означає, що тип кожної змінної має бути оголошеним. Мова має 8 основних (простих) типів, які не є класами та однаково представляються на будь-якій машині, де виконується програма.

Тип	Розмір, байти	Діапазон значень	Приклад запису
boolean	1	true, false	true
char	2	\u0000...\uFFFF	\u0041 або 'A'
byte	1	-128...127	15
short	2	-32768...32767	15
int	4	$-2^{31} \dots 2^{31}-1$	15
long	8	$-2^{63} \dots 2^{63}-1$	15L
float	4	$\pm 3.4E+38$	15.0F
double	8	$\pm 1.79E+308$	15.0 або 15.0D

### **Змінні**

Синтаксис оголошення змінних:

```
тип назваЗмінної [=значення] {, назваЗмінної [= значення]};
```

Наприклад,

```
int i;  
double x, y;  
boolean isZero = false;
```

Перед використанням змінну слід обов'язково ініціалізувати.

### **Масиви**

Масив – структура даних, що зберігає набір значень однакового типу. Пам'ять під масив виділяється у *керованій кучі*. При завершенні життєвого циклу масиву пам'ять, яку він займав, вивільняється збирачем сміття. Доступ до елементів масиву здійснюється за допомогою індексів. Індксація масивів у Java починається з 0. Для створення масиву у Java необхідно оголосити змінну-масив та ініціалізувати її. При створенні за допомогою оператора `new` масиву чисел всі його елементи ініціалізуються нулями (масиви типу `boolean` ініціалізуються значеннями `false`, масиви об'єктів ініціалізуються значеннями `null`). Після створення масиву змінити його розмір неможливо.

#### **Одновимірні масиви**

Синтаксиси оголошення неініціалізованого одновимірного масиву:

```
тип[] змінна;  
тип змінна[];
```

Приклади оголошення неініціалізованого одновимірного масиву типу `int`:

```
int[] arr;  
int arr[];
```

Синтаксиси оголошення та ініціалізації одновимірного масиву:

```
тип[] змінна = new тип[кількість_елементів_масиву];  
тип[] змінна = {значення1, значення2, ..., значенняN};  
тип змінна[] = new тип[кількість_елементів_масиву];  
тип змінна[] = {значення1, значення2, ..., значенняN};
```

Приклади оголошення та ініціалізації одновимірного масиву типу `int`:

```
int[] arr = new int[5];  
int[] arr = {1,2,3,4,5};  
int arr[] = new int[5];  
int arr[] = {1,2,3,4,5};
```

Java дозволяє створювати і *анонімні масиви* (без іменні). При створенні анонімного масиву відбувається виділення необхідної кількості пам'яті для збереження елементів масиву та ініціалізація масиву значеннями зі списку ініціалізації. Синтаксис створення анонімного масиву:

```
new тип [] { значення1, значення2, ..., значенняN };
```

Анонімні масиви корисні тоді, коли необхідно ініціалізувати існуючий масив новими значеннями без створення нової змінної, коли необхідно повернути з методу масив нульової довжини (пустий масив), або коли в метод необхідно передати масив наперед відомих значень. Анонімні масиви дозволяють записати код:

```
int[] arr = {1, 2, 3};  
...  
int[] anonym = {4, 5, 6}  
arr = anonym;
```

у скороченому вигляді:

```
int[] arr = {1, 2, 3};  
...  
arr = new int[] {4, 5, 6};
```

```
int[] arr = {1, 2, 3};  
obj.met(arr);
```

у скороченому вигляді:

```
obj.met(new int[] {1, 2, 3});
```

### Багатовимірні масиви

Багатовимірний масив – це масив, який складається з множини масивів. У Java нема багатовимірних масивів в принципі, а багатовимірні масиви реалізуються як множина одновимірних. Кількість вимірів масиву задається парами закриваючих і відкриваючих прямокутних дужок. Як і одновимірні масиви багатовимірні масиви перед використанням необхідно оголосити і ініціалізувати.

Синтаксиси оголошення неініціалізованого двовимірного масиву:

```
тип[][] змінна;  
тип змінна[][];
```

Приклади оголошення неініціалізованого двовимірного масиву типу `int`:

```
int[][] arr;  
int arr[][];
```

Синтаксиси оголошення та ініціалізації двовимірного масиву:

```
тип[][] змінна = new тип[розмір_виміру_1][розмір_виміру_2];  
тип[][] змінна = {{значення11, значення12, ..., значення1N},  
                  {значення21, значення22, ..., значення2N}  
                  ...  
                  {значенняM1, значенняM2, ..., значенняMN}};  
тип змінна[][] = new тип[розмір_виміру_1][розмір_виміру_2];  
тип змінна[][] = {{значення11, значення12, ..., значення1N},  
                  {значення21, значення22, ..., значення2N}  
                  ...  
                  {значенняM1, значенняM2, ..., значенняMN}};
```

Приклади оголошення та ініціалізації двовимірного масиву типу `int`:

```
int[][] arr = new int[2][5];  
int[][] arr = {{1,2,3,4,5},{11,12,13,14,15}};  
int arr[][] = new int[2][5];  
int arr[][] = {{1,2,3,4,5},{11,12,13,14,15}};
```

### Зубчаті масиви

Завдяки тому, що багатовимірні масиви у Java реалізуються як множина одновимірних масивів, стає можливим реалізувати багатовимірні масиви з різною кількістю елементів у межах виміру. Синтаксис оголошення зубчатого масиву нічим не відрізняється від синтаксису оголошення звичайного багатовимірного масиву. Різниця є лише у способі ініціалізації, де використовується виділення пам'яті під різну кількість елементів у межах виміру.

Синтаксис оголошення та ініціалізації зубчатого масиву:

```
тип[][] змінна = new тип[N][];  
змінна[0] = new тип[розмір_виміру_20];  
змінна[1] = new тип[розмір_виміру_21];  
...  
змінна[N-1] = new тип[розмір_виміру_2N-1];
```

Приклад оголошення та ініціалізації зубчатого масиву:

```
int[][] arr = new int[3][];  
arr[0] = new int[3];  
arr[1] = new int[0];  
arr[2] = new int[2];
```

### Особливості використання масивів

Розмір масиву зберігається у властивості `length`.

Копіювання масивів не можна здійснити звичайним присвоюванням однієї змінної-масиву іншій. У цьому випадку обидві змінні-масиви посилатимуться на одну і ту саму область пам'яті, тобто фізично на один і той самий масив. Для коректного копіювання

масивів слід скористатися методом `copyOf` класу `Arrays`. Цей метод створює копію масиву, що переданий через перший параметр методу, у пам'яті та повертає посилання на нього. Кількість елементів масиву, що підлягають копіюванню, передається через другий параметр методу. Приклад копіювання масиву:

```
int [] copiedArray = Arrays.copyOf(originalArray, originalArray.length);
```

Для перевірки масивів на ідентичність використовується метод `equals` класу `Arrays`. Цей метод приймає через параметри 2 масиви простих типів, та повертає `true`, якщо вони ідентичні, та `false`, якщо ні. Приклад порівняння масивів на ідентичність:

```
boolean compared = Arrays.equals(array1, array2);
```

### Основні конструкції мови Java

Основні конструкції мови Java багато в чому співпадають з аналогічними конструкціями мов C/C++. Такі оператори як `switch`, `if-else`, `while`, `do-while` – ідентичні аналогічним конструкціям у мовах C/C++. Оператор циклу `for` має деякі особливості. У Java цей оператор має 2 різновиди:

- конструкція в стилі C/C++ з полем ініціалізації, логічною умовою та кроком;
- конструкція з *синтаксисом* `foreach`.

**Синтаксис оператора `for` в стилі C/C++** має такий вигляд:

```
for (ініціалізація лічильника; логічна умова; модифікація лічильника)  
    оператори
```

Робота оператора циклу `for` в стилі C/C++ починається з виконання операторів поля ініціалізації лічильника, після чого відбувається перевірка логічної умови, виконання операторів тіла циклу та модифікація лічильника. Після першої ітерації, поки логічний вираз є істинним, циклічно послідовно виконуються лише операції перевірки умови, тіла циклу та модифікації лічильника. Область видимості змінних, що оголошені в полі ініціалізації лічильника та час їх життя обмежені тілом циклу `for`.

Приклад оператора оператора `for` в стилі C/C++:

```
for (int i = 0; i < 100; i++)  
    System.out.println(i);
```

**Оператор циклу `for` з синтаксисом `foreach`** дозволяє послідовно перебирати всі елементи набору даних без застосування лічильника. Таким набором даних може бути будь-який клас, що реалізує інтерфейс `Iterable`, або масив. Оператор циклу `for` з синтаксисом `foreach` має наступний вигляд:



```
for (змінна : набір даних)
    оператори
```

При опрацюванні циклу змінній послідовно присвоюється кожен елемент набору даних (наприклад, елемент масиву) після чого виконується оператор.

Приклад використання оператора `for` з синтаксисом `foreach`:

```
for (int elem: arr)
    System.out.println(elem);
```

Цей фрагмент коду є аналогічним наступному:

```
for (int i = 0; i < arr.length; i++)
    System.out.println(arr[i]);
```

**Оператори переривання потоку виконання.** До операторів переривання потоку виконання відносяться оператори `break` і `continue`. Вони призначені для переривання послідовності виконання операцій в циклах. У циклах дані оператори можуть використовуватися з мітками і без них.

Оператор `break` без мітки перериває виконання циклу та передає керування на першу інструкцію, що стоїть після циклу. Крім цього `break` застосовується для переривання послідовності виконання операцій у операторі `switch` та передачі керування на наступну інструкцію після `switch`. Оператор `continue` без мітки перериває виконання поточної ітерації циклу та передає керування наступній ітерації циклу.

Оператори `break` і `continue` з міткою застосовуються у вкладених циклах для переривання роботи вкладеного і зовнішнього циклів. Мітка у цих операторах призначена для того, щоб визначити зовнішній цикл, на роботу якого впливатиме оператор `break` або `continue`. Для цього вона ставиться перед оператором зовнішнього циклу, на який має впливати `break` або `continue`. Міткою може бути будь-який ідентифікатор, що відповідає правилам оголошення ідентифікаторів (змінних) та закінчується двокрапкою.

Оператор `break` з міткою перериває виконання вкладеного циклу та передає керування на першу інструкцію, що стоїть після циклу, який маркований міткою. Оператор `continue` з міткою перериває виконання поточної ітерації циклу та передає керування наступній ітерації зовнішнього циклу, що маркований міткою.

Приклад використання операторів `break` і `continue`:

```
label1:
зовнішній цикл
{
    внутрішній цикл
    {
        ...
        break;                // (1)
        ...
        continue;            // (2)
    }
}
```

```

        ...
        break label1;           // (3)
        ...
        continue label1;       // (4)
    }
}
оператор1

```

При виконанні (1) відбудеться вихід з внутрішнього циклу. При виконанні (2) відбудеться перехід на наступну ітерацію внутрішнього циклу. При виконанні (3) відбудеться вихід з внутрішнього і зовнішнього циклів та перехід на мітку label1 з передачею керування на оператор1. Тобто, наступним після виконання break label1 виконається оператор1. При виконанні (4) відбудеться вихід з внутрішнього і зовнішнього циклів та перехід на мітку label1 з передачею керування наступній ітерації зовнішнього циклу. Тобто, після виконання continue label1 виконається наступна ітерація зовнішнього циклу.

### **Ввід з консолі**

Для введення інформації з консолі необхідно створити об'єкт класу Scanner і зв'язати його з стандартним потоком вводу System.in, наприклад:

```
Scanner in = new Scanner(System.in);
```

Зробивши це ми отримаємо доступ до методів класу Scanner, які призначені для введення даних простих типів і рядків:

- hasNext – перевіряє чи є доступні дані для введення;
- nextBoolean – вводить дані типу boolean;
- nextByte – вводить дані типу byte;
- nextShort – вводить дані типу short;
- nextInt – вводить дані типу int;
- nextLong – вводить дані типу long;
- nextFloat – вводить дані типу float;
- nextDouble – вводить дані типу double;
- nextLine – вводить весь рядок і представляє його як об'єкт класу String;
- next – вводить одне слово (токен) і представляє його як об'єкт класу String.

### **Вивід на консоль**

Популярним механізмом виводу на консоль є використання методу print об'єкту out з пакету System, який виводить переданий через параметр текстовий рядок на екран:

```
System.out.print("Hello!!!");
```

Недоліком цього методу є неможливість здійснити форматований вивід на консоль. Для здійснення форматowanego виводу на консоль використовується метод printf

синтаксис якого повністю співпадає з синтаксисом функції `printf` у мові C/C++ за винятком кількох доданих функціональних можливостей по виводу дати, часу, логічних значень і хеш-кодів, наприклад:

```
System.out.printf("Hello %s!!! ", str);
```

### **Ввід з текстового файлу**

Для введення інформації з файлу необхідно підключити пакет `java.io` та створити об'єкт класу `Scanner` з об'єкту `File`:

```
Scanner fin = new Scanner(File("MyFile.txt"));
```

При компіляції цього рядка коду може виникнути виключна ситуація неіснування файлу з якого має проходити ввід даних (`FileNotFoundException`), яка розглядається компілятором серйозніше, ніж, скажімо, ділення на нуль. Тому, компілятор не дасть нам просто так скомпілювати цей рядок коду. Для коректної компіляції цього рядку коду необхідно або вказати, що наш метод може генерувати це виключення, або реалізувати код для перехоплення і обробки виключення. Для того, щоб вказати, що наш метод може генерувати це виключення, у оголошенні методу, де відбувається створення об'єкту класу `Scanner` з файлу, необхідно додати наступний код:

```
throws FileNotFoundException
```

наприклад,

```
public static void main(String[] args) throws FileNotFoundException
```

Пошук файлу відбувається у директорії з якої була запущена на виконання програма. Після відкриття файлу інформацію з нього можна читати використовуючи методи класу `Scanner`.

### **Вивід у текстовий файл**

Для виведення інформації у текстовому вигляді у файл треба підключити пакет `java.io` та створити об'єкт класу `PrintWriter` в конструкторі якого необхідно вказати назву файлу, що відкривається на запис, наприклад:

```
PrintWriter fout = new PrintWriter ("MyFile.txt");
```

Зробивши це ми отримаємо доступ до методів класу `PrintWriter`, які призначені для виведення даних простих типів і рядків:

`print` – виводить значення простих типів і рядків у текстовому вигляді;

`write` – призначений для виводу даних типу `char` і `String` у текстовий файл.

При створенні об'єкту класу `PrintWriter` з іменем файлу, який не може бути створений буде згенероване виключення `FileNotFoundException`. Для його обробки необхідно виконати такі самі дії, що і в розглянутому вище способі введення даних з файлу.

## **КОНТРОЛЬНІ ПИТАННЯ**

1. Які дескриптори використовуються при коментуванні класів?
2. Які дескриптори використовуються при коментуванні методів?
3. Як автоматично згенерувати документацію?
4. Які прості типи даних підтримує Java?
5. Як оголосити змінну-масив?
6. Які керуючі конструкції підтримує Java?
7. В чому різниця між різними варіантами оператора `for`?
8. Як здійснити ввід з консолі?
9. Як здійснити ввід з текстового файлу?
10. Як здійснити запис у текстовий файл?

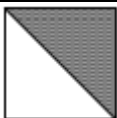

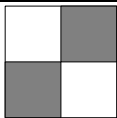
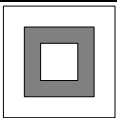


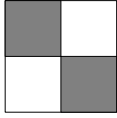
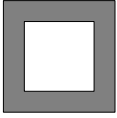



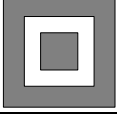
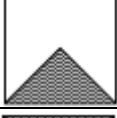
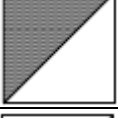
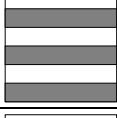
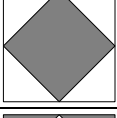


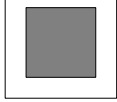
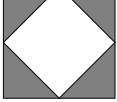
## **ЛІТЕРАТУРА**

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 згідно варіанту. Програма має задовольняти наступним вимогам:
  - програма має розміщуватися в загальнодоступному класі `Lab1ПрізвищеГрупа`;
  - програма має генерувати зубчатий масив, який міститиме лише заштриховані області квадратної матриці згідно варіанту;
  - розмір квадратної матриці і символ-заповнювач масиву вводяться з клавіатури;
  - при не введенні або введенні кількох символів-заповнювачів відбувається коректне переривання роботи програми;
  - сформований масив вивести на екран і у текстовий файл;
  - програма має володіти коментарями, які дозволять автоматично згенерувати документацію до розробленої програми.
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 ЗАВДАНЬ

№		№		№		№	
1		6		11		16	
2		7		12		17	
3		8		13		18	
4		9		14		19	
5		10		15		20	

## 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 підпункт меню File->New->Project... У вікні, що відкриється слід вибрати Java Project (рис. 1) та натиснути кнопку "Next>".

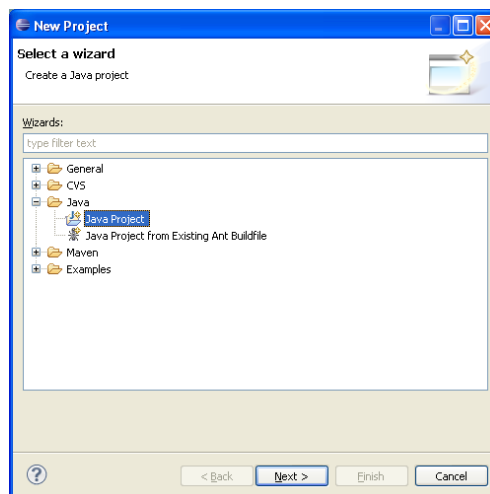


Рис. 1. Діалогове вікно вибору типу проекту в Eclipse IDE.

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).

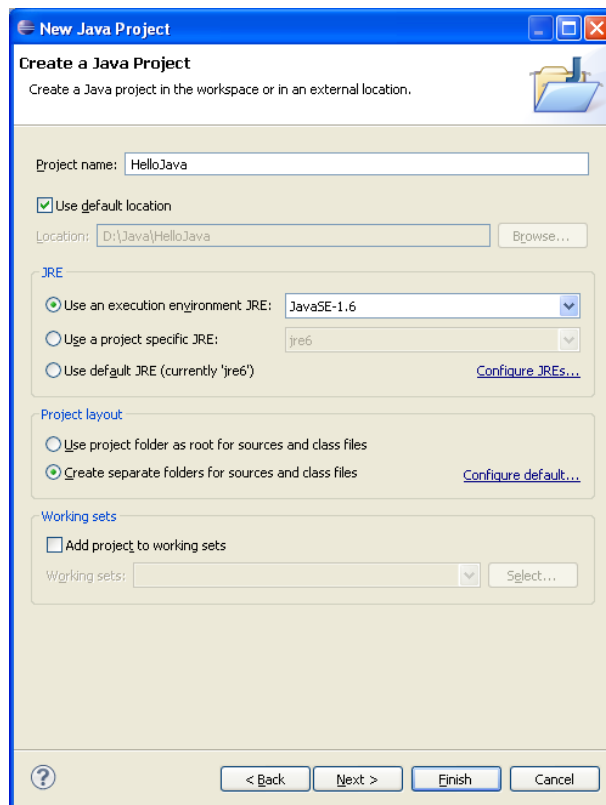


Рис. 2. Діалогове вікно створення нового проекту в Eclipse IDE.

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

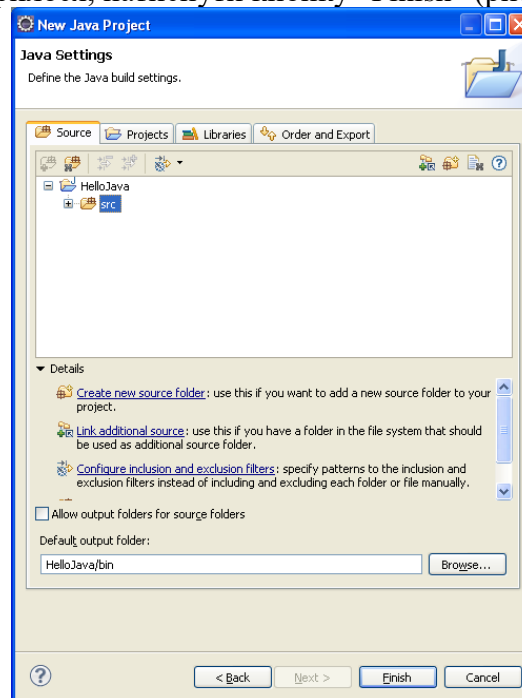


Рис. 3. Діалогове вікно налаштування властивостей побудови нового проекту в Eclipse IDE.

6. У вікні, що з'явилося (рис.4), викликати підпункт меню File->New->File. У вікні, що відкрилося слід задати каталог, де розташовуватиметься новий файл з кодом, назву файлу з розширенням .java та натиснути кнопку "Finish" (рис. 5).

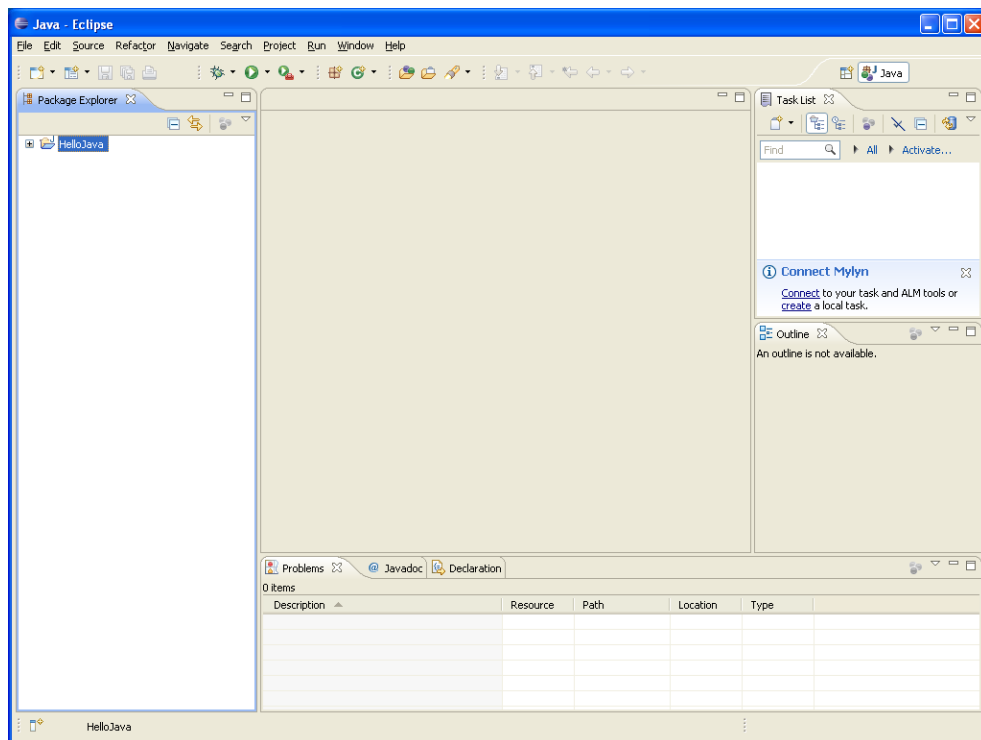


Рис. 4. Середовище Eclipse IDE з відкритим проектом Java.

6. У створеному файлі написати програму згідно завдання. Зверніть увагу, що програма має обов'язково містити public клас, назва якого співпадає з назвою файлу.
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).

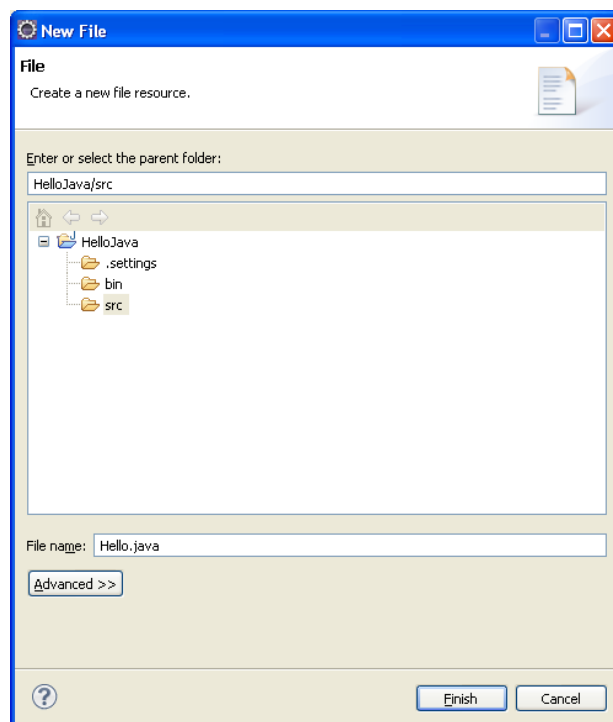


Рис. 5. Створення нового файлу.

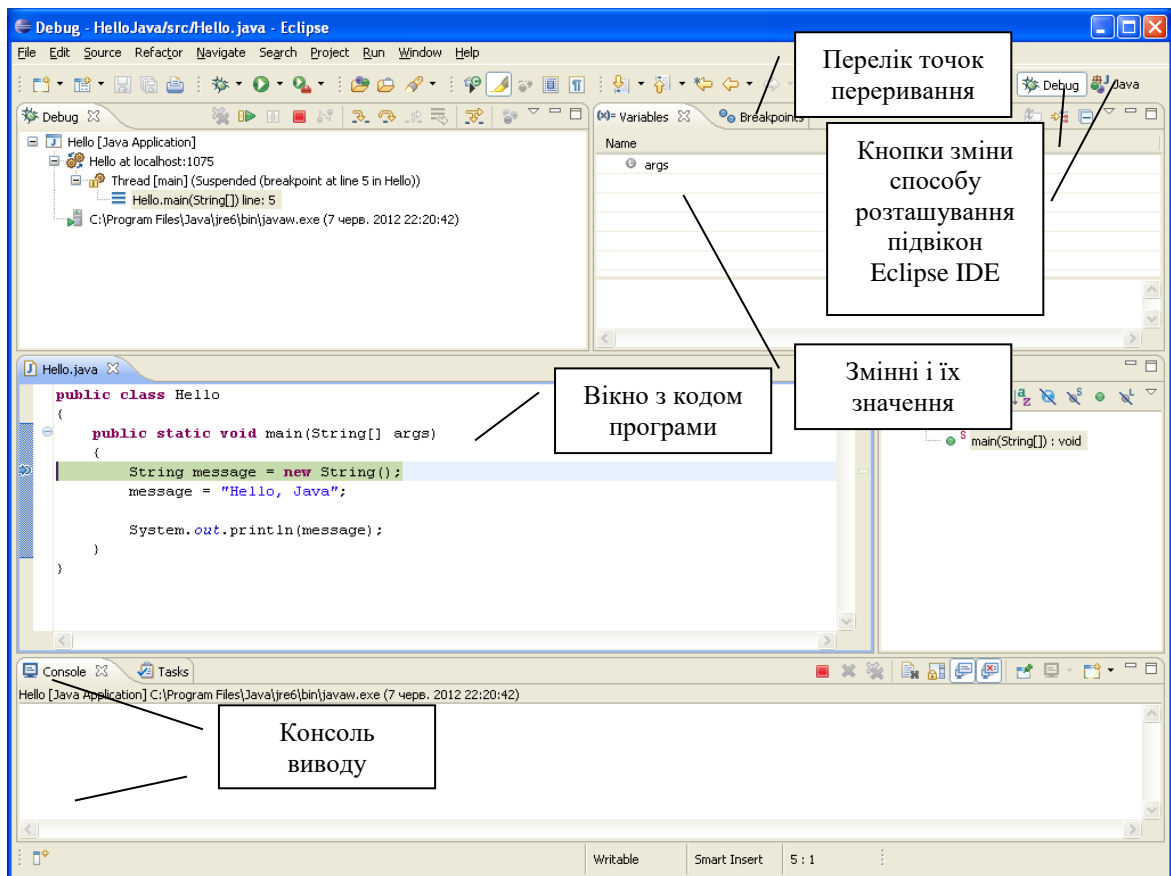


Рис. 6. Процес налагодження програми.

9. Автоматично згенерувати документацію у каталог \doc вашого проекту.  
Проаналізувати автоматично згенеровану документацію.

## ДЕМОНСТРАЦІЙНА ПРОГРАМА

```

/*****
 * 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
 *****/

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

/**
 * Клас Lab1 реалізує приклад програми до лабораторної роботи №1
 */

public class Lab1
{
    /**
     * Статичний метод main є точкою входу в програму
     *
     * @param args
     * @throws FileNotFoundException
     */
    public static void main(String[] args) throws FileNotFoundException

```



```

{
    int nRows;
    char[][] arr;
    String filler;
    Scanner in = new Scanner(System.in);
    File dataFile = new File("MyFile.txt");
    PrintWriter fout = new PrintWriter(dataFile);

    System.out.print("Введіть розмір квадратної матриці: ");
    nRows = in.nextInt();
    in.nextLine();

    arr = new char[nRows][];
    for(int i = 0; i < nRows; i++)
    {
        arr[i] = new char[i+1];
    }

    System.out.print("\nВведіть символ-заповнювач: ");
    filler = in.nextLine();

exit:
    for(int i = 0; i < nRows; i++)
    {
        for(int j = 0; j < i+1; j++)
        {
            if(filler.length() == 1)
            {
                arr[i][j] = (char) filler.codePointAt(0);
                System.out.print(arr[i][j] + " ");
                fout.print(arr[i][j] + " ");
            }
            else if (filler.length() == 0)
            {
                System.out.print("\nНе введено символ
заповнювач");
                break exit;
            }
            else
            {
                System.out.print("\nЗабагато символів
заповнювачів");
                break exit;
            }
        }
        System.out.print("\n");
        fout.print("\n");
    }
    fout.flush();
    fout.close();
}

```

НАВЧАЛЬНЕ ВИДАННЯ

## ДОСЛІДЖЕННЯ БАЗОВИХ КОНСТРУКЦІЙ МОВИ JAVA

### МЕТОДИЧНІ ВКАЗІВКИ

до лабораторної роботи № 1 з дисципліни “ Кросплатформні засоби програмування ”  
для студентів базового напрямку 6.123 - “Комп’ютерна інженерія”

**Укладач**

Олексів М. В., к.т.н.

**Редактор**

**Комп’ютерне верстання**

Здано у видавництво . Підписано до друку  
Формат 70х100/16. Папір офсетний. Друк на різнографі  
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*