Міністерствоосвіти і науки

Національний університет „Львівська політехніка”

**Кафедра ЕОМ**

****

**Звіт про виконання лабораторної роботи №2**

З дисципліни: «Кросплатформні засоби програмування»

На тему: «Класи та пакети»

**Виконала:** ст. групи КІ-305

Циніцька Олена

**Прийняв:** доцент каф. ЕОМ

Іванов Ю. С.

**Львів – 2023**

**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.Lab3;

• 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 розробленого пакету.

3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації.

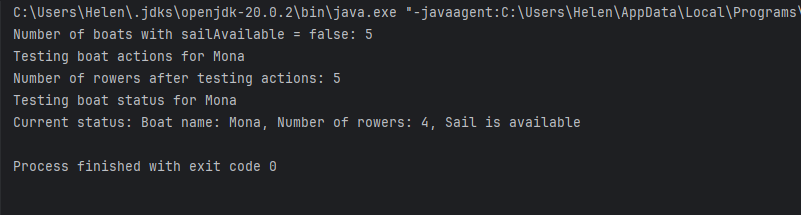
4. Дати відповідь на контрольні запитання.

Варіант: 26. човен на веслах

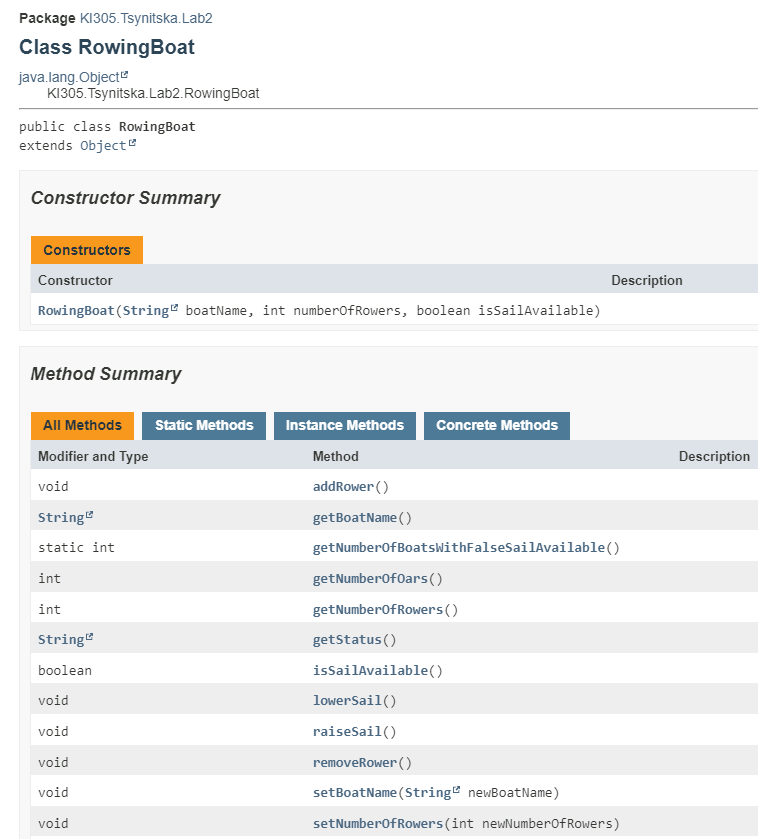
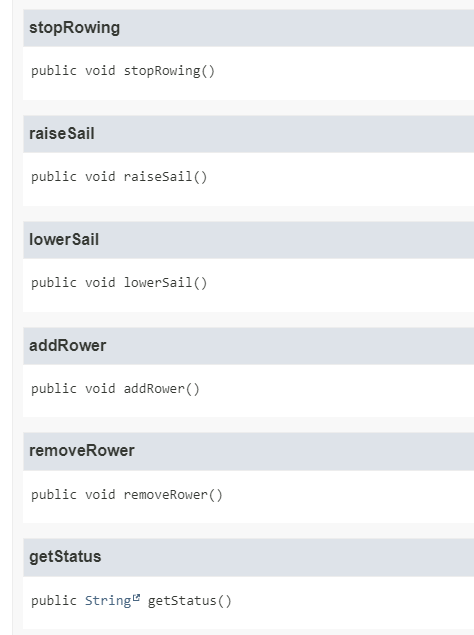
**Код програми:**

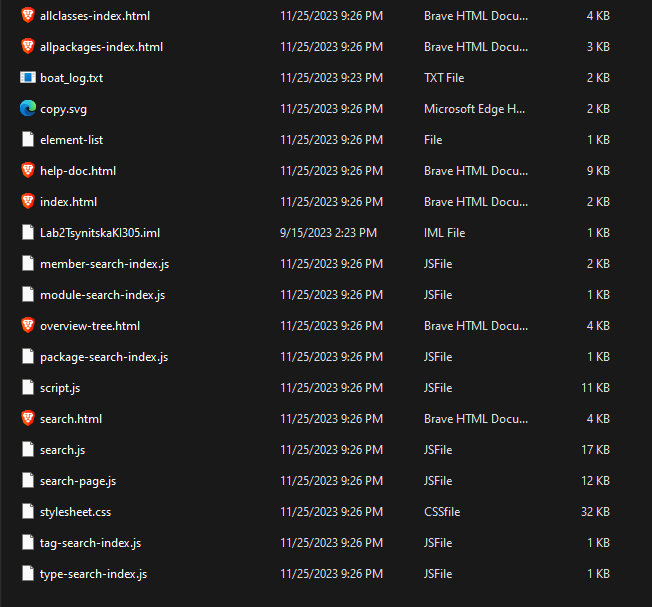
// Package declaration  
package KI305.Tsynitska.Lab2;  
  
// Import statements  
import java.io.FileWriter;  
import java.io.IOException;  
  
// RowingBoat class definition  
public class RowingBoat {  
 // Instance variables  
 private String boatName;  
 private int numberOfRowers;  
 private boolean isSailAvailable;  
 private int numberOfOars;  
  
 // Static variable to track the number of boats with false sail availability  
 private static int *numberOfBoatsWithFalseSailAvailable* = 0;  
  
 // Constructor  
 public RowingBoat(String boatName, int numberOfRowers, boolean isSailAvailable) {  
 this.boatName = boatName;  
 this.numberOfRowers = numberOfRowers;  
 this.isSailAvailable = isSailAvailable;  
 this.numberOfOars = numberOfRowers \* 2;  
  
 // Increment the counter if sail is not available  
 if (!isSailAvailable) {  
 *numberOfBoatsWithFalseSailAvailable*++;  
 }  
 }  
  
 // Getter for the counter  
 public static int getNumberOfBoatsWithFalseSailAvailable() {  
 return *numberOfBoatsWithFalseSailAvailable*;  
 }  
  
 // Getter methods  
 public String getBoatName() {  
 return boatName;  
 }  
  
 public int getNumberOfRowers() {  
 return numberOfRowers;  
 }  
  
 public boolean isSailAvailable() {  
 return isSailAvailable;  
 }  
  
 public int getNumberOfOars() {  
 return numberOfOars;  
 }  
  
 // Setter methods  
 public void setNumberOfRowers(int newNumberOfRowers) {  
 this.numberOfRowers = newNumberOfRowers;  
 this.numberOfOars = newNumberOfRowers \* 2;  
 }  
  
 public void setSailAvailable(boolean isSailAvailable) {  
 this.isSailAvailable = isSailAvailable;  
 // Update the counter based on sail availability  
 if (!isSailAvailable) {  
 *numberOfBoatsWithFalseSailAvailable*++;  
 } else {  
 *numberOfBoatsWithFalseSailAvailable*--;  
 }  
 }  
  
 public void setBoatName(String newBoatName) {  
 this.boatName = newBoatName;  
 }  
  
 // Boat action methods  
 public void startRowing() {  
 log("Rowing the boat.");  
 }  
  
 public void stopRowing() {  
 log("Stopped rowing.");  
 }  
  
 public void raiseSail() {  
 if (isSailAvailable) {  
 log("Raising the sail.");  
 } else {  
 log("No sail available.");  
 }  
 }  
  
 public void lowerSail() {  
 log("Lowering the sail.");  
 }  
  
 public void addRower() {  
 numberOfRowers++;  
 numberOfOars = numberOfRowers \* 2;  
 log("Added a rower. Total rowers: " + numberOfRowers);  
 }  
  
 public void removeRower() {  
 if (numberOfRowers > 0) {  
 numberOfRowers--;  
 numberOfOars = numberOfRowers \* 2;  
 log("Removed a rower. Total rowers: " + numberOfRowers);  
 } else {  
 log("No rowers on the boat.");  
 }  
 }  
  
 // Get the status of the boat as a formatted string  
 public String getStatus() {  
 String sailStatus = isSailAvailable ? "Sail is available" : "Sail is not available";  
 return "Boat name: " + boatName + ", Number of rowers: " + numberOfRowers + ", " + sailStatus;  
 }  
  
 // Private method to log messages to a file  
 private void log(String message) {  
 try {  
 FileWriter writer = new FileWriter("boat\_log.txt", true);  
 writer.write(message + "\n");  
 writer.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

// Package declaration  
package KI305.Tsynitska.Lab2;  
  
// RowingBoatDriver class definition  
public class RowingBoatDriver {  
  
 // Main method  
 public static void main(String[] args) {  
 // Create three instances of RowingBoat with different parameters  
 RowingBoat boat1 = new RowingBoat("Mona", 4, false);  
 RowingBoat boat2 = new RowingBoat("Mona", 4, true);  
 RowingBoat boat3 = new RowingBoat("Mona", 4, false);  
  
 // Call main1 method  
 *main1*();  
  
 // Print the number of boats with sailAvailable = false  
 System.*out*.println("Number of boats with sailAvailable = false: " + RowingBoat.*getNumberOfBoatsWithFalseSailAvailable*());  
  
 // Test additional functionality  
 *testBoatActions*(boat1);  
 *testBoatStatus*(boat2);  
 }  
  
 // Additional method for testing  
 public static void main1() {  
 // Create four instances of RowingBoat with different parameters  
 RowingBoat boat1 = new RowingBoat("Stella", 4, true);  
 RowingBoat boat2 = new RowingBoat("Stella", 4, false);  
 RowingBoat boat3 = new RowingBoat("Stella", 4, false);  
 RowingBoat boat4 = new RowingBoat("Stella", 4, false);  
 }  
  
 // Additional method for testing boat actions  
 public static void testBoatActions(RowingBoat boat) {  
 System.*out*.println("Testing boat actions for " + boat.getBoatName());  
 boat.startRowing();  
 boat.raiseSail();  
 boat.addRower();  
 boat.lowerSail();  
 boat.stopRowing();  
 System.*out*.println("Number of rowers after testing actions: " + boat.getNumberOfRowers());  
 }  
  
 // Additional method for testing boat status  
 public static void testBoatStatus(RowingBoat boat) {  
 System.*out*.println("Testing boat status for " + boat.getBoatName());  
 System.*out*.println("Current status: " + boat.getStatus());  
 }  
}

****

**Згенерована документація:**

**   **

****

**Відповіді на контрольні питання**

1. **Синтаксис визначення класу.**

|  |
| --- |
| **public** **class** MyClass {  *// Тіло класу*  } |

1. **Синтаксис визначення методу.**

|  |
| --- |
| [модифікатори] [тип повернення] ім'я\_методу([параметри]) {  // Тіло методу  } |

1. **Синтаксис оголошення поля.**

|  |
| --- |
| [модифікатори] тип\_даних ім'я\_поля; |

1. Як оголосити та ініціалізувати константне поле?

В Java константне поле оголошується за допомогою ключового слова **final**, приклад:

|  |
| --- |
| **public** **final** **int** MY\_CONSTANT = 42; |

1. **Які є способи ініціалізації полів?**

Поля можуть бути ініціалізовані при оголошенні, в конструкторі класу або в блоках ініціалізації. Приклад:

|  |
| --- |
| **public** **class** MyClass {  **public** **int** field1 = 10; *// Ініціалізація при оголошенні*  **public** **int** field2;    **public** MyClass() {  field2 = 20; *// Ініціалізація в конструкторі*  }  } |

1. **Синтаксис визначення конструктора.**

Конструктор має те ж ім'я, що і клас, і не має типу повернення. Ось приклад:

|  |
| --- |
| **public** MyClass() {  *// Тіло конструктора*  } |

1. **Синтаксис оголошення пакету.**

Пакет оголошується на початку файлу програми,

|  |
| --- |
| **package** mypackage; |

1. **Як підключити до програми класи, що визначені в зовнішніх пакетах?**

Використовується імпорт для включення класів з інших пакетів. Наприклад:

|  |
| --- |
| **import** mypackage.MyClass; |

1. **В чому суть статичного імпорту пакетів?**

Статичний імпорт дозволяє вам використовувати статичні члени класу без звернення до самого класу. Наприклад:

|  |
| --- |
| **import** static mypackage.MyClass.myStaticMethod; |

1. **Які вимоги ставляться до файлів і каталогів при використанні пакетів?**

Файли з кодом класів повинні бути розташовані у каталозі, відповідному їхньому пакету, і мати ім'я, яке відображає ієрархію пакетів (наприклад, **mypackage/MyClass.java**). Каталоги, що відповідають пакетам, повинні бути включені в CLASSPATH для компіляції і виконання.

**Висновок:**

У цій лабораторній роботі дослідила процес розробки класів та пакетів в мові програмування Java. Метою роботи було ознайомлення з основними аспектами створення класів, визначення методів, оголошення полів, ініціалізації константних полів, роботи з конструкторами та пакетами.