Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра «Електронних обчислювальних машин»

Звіт

з лабораторної роботи № 2

з дисципліни: «Кросплатформенні засоби програмування»

на тему: «Класи та пакети»

Виконав:

студент групи КІ-306

Ширий Б. І.

Прийняв:

доцент кафедри ЕОМ

Іванов Ю. С.

Львів – 2023

МЕТОДИЧНІ ВІДОМОСТІ РОБОТИ

МЕТА

Ознайомитися з процесом розробки класів та пакетів мовою Java.

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, а саме для мого варіанту – це **«Шлюпка на веслах»**;
- 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

Для розробленої програми згенерувати документацію.

№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.

ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ

ВИХІДНИЙ КОД

Написав програму, що симулює шлюпку на веслах та навів її у таких файлах

- Моделі програми - лістинг 2.1,
- BoatOnOarsDriver.java - лістинг 2.2,
- BoatMovingFunctions.java - лістинг 2.3,
- FileUtil.java - лістинг 2.4.

Лістинг 2.1. Код основної програми.

```
/**
 * My package I've created because of the task
 */
package CI_306.Shyryi.Lab2;

import java.util.List;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

import java.util.ArrayList;
import java.awt.FlowLayout;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

/**
 * This class represents a rowing boat with the ability to move in different directions.
 * The boat can have a name, be attached to a larger boat, have pairs of oars, sailors, and passengers.
 */
public class BoatOnOars{

            private String name;
            private Boat boat;
            private PairOfOars[] pairOfOars;
            private Sailer[] sailer;
            private List<Person> passengers = new ArrayList<>();
            public enum MoveDirection {
                LEFT,
                RIGHT,
                FORWARD,
                BACKWARD
            }

    /**
     * Creates an empty rowing boat.
     */
    public BoatOnOars() {
            System.out.print("\nПуста шлюпка на веслах була створенна");

            FileUtil.appendToFile("Логи.txt", "Був створений " + this.name);
    }

    /**
     * Creates a rowing boat with specified parameters.
     *
     * @param name       The name of the rowing boat
     * @param boat       The larger boat to which the rowing boat is attached
     * @param pairOfOars An array of oar pairs
     * @param sailer     An array of sailors
     * @param passengers A list of passengers
     */
    public BoatOnOars(String name, Boat boat, PairOfOars[] pairOfOars, Sailer[] sailer, List<Person> passengers) {
        this.name = name;
        this.boat = boat;
        this.pairOfOars = pairOfOars;
        this.sailer = sailer;
        this.passengers = passengers;

        FileUtil.appendToFile("Логи.txt", "Був створений " + this.name);
    }

    /**
     * Copies another rowing boat.
     *
     * @param other Another rowing boat to be copied
     */
    public BoatOnOars(BoatOnOars other) {
        this.name = other.name;
        this.boat = new Boat(other.boat);
```

```
        this.pairOfOars = other.pairOfOars.clone();
        this.sailer = other.sailer.clone();
        this.passengers = new ArrayList<>(other.passengers);

        FileUtil.appendToFile("Логи.txt", "Був створений " + this.name);
    }

    /**
     * This initialization block sets initial values for the rowing boat.
     */
    {
        this.name = "Невідомо";
        this.boat = null;
        this.pairOfOars = null;
        this.sailer = null;
    }

    /**
     * Initiates rowing when interacting with keys. Displays the movement state in a window.
     */
    public void Rowing()
    {
           JFrame frame = new JFrame();
                       frame.setVisible(true);
                       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                       frame.setSize(300, 650);
                       frame.setFocusable(true);

                       JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());

                       JLabel page = new JLabel();
                       panel.add(page);
                       page.setText("<html>" + getBoatInHtml() + "<br/>" +
                                               getSailorsInHtml() + "<br/>" +
                                               getPassengersInHtml() + "<br/>" +
                                               "Руху ще не було спричинено" + "</html>");

                       frame.addKeyListener((KeyListener) new KeyListener() {

                                   public void keyTyped(KeyEvent e) {
                                               // TODO Auto-generated method stub

                                   }

                                   public void keyPressed(KeyEvent e) {
                                               int keyCode = e.getKeyCode();
                                               switch (keyCode) {
                                               case KeyEvent.VK_UP:
                                                           page.setText("<html>" + getBoatInHtml() + "<br/>" +
                                                                                   getSailorsInHtml() + "<br/>" +
                                                                                   getPassengersInHtml() + "<br/>" +
                                                                                   (isRowingAbleTo(MoveDirection.FORWARD) ?
                                                           "Шлюпка провеслувала вперед" : "У шлюпки проблеми") + "</html>");
                                               FileUtil.appendToFile("Логи.txt", "Шлюпка " + name + " провеслувала вперед");
                                                           break;
                                               case KeyEvent.VK_DOWN:
                                                           page.setText("<html>" + getBoatInHtml() + "<br/>" +
                                                                                   getSailorsInHtml() + "<br/>" +
                                                                                   getPassengersInHtml() + "<br/>" +
                                                                                   (isRowingAbleTo(MoveDirection.BACKWARD) ?
                                                           "Шлюпка провеслувала назад": "У шлюпки проблеми") + "</html>");
                                               FileUtil.appendToFile("Логи.txt", "Шлюпка " + name + " провеслувала назад");
                                                           break;
                                               case KeyEvent.VK_RIGHT:
                                                           page.setText("<html>" + getBoatInHtml() + "<br/>" +
                                                                                   getSailorsInHtml() + "<br/>" +
                                                                                   getPassengersInHtml() + "<br/>" +
                                                                                   (isRowingAbleTo(MoveDirection.RIGHT) ?
                                                           "Шлюпка провеслувала вправо" : "У шлюпки проблеми") + "</html>");
                                               FileUtil.appendToFile("Логи.txt", "Шлюпка " + name + " провеслувала вправо");
                                                           break;
                                               case KeyEvent.VK_LEFT:
                                                           page.setText("<html>" + getBoatInHtml() + "<br/>" +
                                                                                   getSailorsInHtml() + "<br/>" +
                                                                                   getPassengersInHtml() + "<br/>" +
                                                                                   (isRowingAbleTo(MoveDirection.LEFT) ?
                                                           "Шлюпка провеслувала вліво" : "У шлюпки проблеми") + "</html>");
                                               FileUtil.appendToFile("Логи.txt", "Шлюпка " + name + " провеслувала вліво");
                                                           break;
                                               case KeyEvent.VK_R:
                                               for (int i = 0; i < getBoat().getDeck().getSailorsCapacity(); i++)
                                               {
                                                           sailer[i].Rest();
                                                           pairOfOars[i].getLeftOar().Repair();
                                                           pairOfOars[i].getRightOar().Repair();
                                               }
                                               boat.getBody().Repair();
                                                           page.setText("<html>" + getBoatInHtml() + "<br/>" +
                                                                                   getSailorsInHtml() + "<br/>" + getPassengersInHtml()
                                                                                   + "<br/>Ремонт та відпочинок здійснено</html>");
                                                           FileUtil.appendToFile("Логи.txt", "Ремонт та відпочинок здійснено");
                                                           break;
                                               }
                                   }
```

```
                                public void keyReleased(KeyEvent e) {
                                        // TODO Auto-generated method stub

                                }

                        });

                        frame.add(panel);
        }

        /**
         * Checks if the rowing boat is able to move in the specified direction.
         *
         * @param moveDirection The direction in which the boat should move (FORWARD, BACKWARD, RIGHT, or LEFT)
         * @return True if the boat is able to move in the specified direction; otherwise, false
         */
        public Boolean isRowingAbleTo(MoveDirection moveDirection)
        {
                for (int i = 0; i < getBoat().getDeck().getSailorsCapacity(); i++)
                {
                 switch (moveDirection)
                 {
                 case FORWARD:
                                if(!pairOfOars[i].MoveForward(sailer[i], boat)){ return false; }
                            break;
                 case BACKWARD:
                                if(!pairOfOars[i].MoveBackward(sailer[i], boat)){ return false; }
                            break;
                 case RIGHT:
                                if(!pairOfOars[i].MoveRight(sailer[i], boat)){ return false; }
                            break;
                 case LEFT:
                                if(!pairOfOars[i].MoveLeft(sailer[i], boat)){ return false; }
                            break;
                 }
                }
                return true;
        }

        /**
         * Generates an HTML representation of the boat and its characteristics.
         *
         * @return An HTML string containing information about the boat's name and characteristics
         */
        public String getBoatInHtml()
        {
                return "<p> Ви пливете на " + this.name + " </p> "
                                        + "<p> Характеристики: </p> "
                                        + this.getBoat().getBody().toHtml()
                                        + this.getBoat().getDeck().toHtml();
        }

        /**
         * Generates an HTML representation of the sailors and their associated oars.
         *
         * @return An HTML string containing information about sailors and their oars
         */
        public String getSailorsInHtml()
        {
                String htmlRaw = "";
                for (int i = 0; i < getBoat().getDeck().getSailorsCapacity(); i++)
                {
                        htmlRaw += "<p> Весляр " + i + ": </p>"
                                                + this.sailer[i].toHtml() + this.pairOfOars[i].toHtml();
                }
                return htmlRaw;
        }

        /**
         * Generates an HTML representation of the passengers.
         *
         * @return An HTML string containing information about passengers
         */
        public String getPassengersInHtml()
        {
                String htmlRaw = "";
                for (int i = 0; i < getBoat().getDeck().getPassengersCapacity(); i++)
                {
                        htmlRaw += "<p> Пасажир " + i + ": </p>" + this.passengers.get(i).toHtml();
                }
                return htmlRaw;
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        public Boat getBoat() {
            return boat;
        }
```

```
    public void setBoat(Boat boat) {
        this.boat = boat;
    }

    public PairOfOars[] getPairOfOars() {
        return pairOfOars;
    }

    public void setPairOfOars(PairOfOars[] pairOfOars) {
        this.pairOfOars = pairOfOars;
    }

    public Sailer[] getSailer() {
        return sailer;
    }

    public void setSailer(Sailer[] sailer) {
        this.sailer = sailer;
    }

    public List<Person> getPassengers() {
        return passengers;
    }

    public void setPassengers(List<Person> passengers) {
        this.passengers = passengers;
    }
}

/**
 * This class represents a boat with a deck and body.
 * The boat can have various configurations for its deck and body.
 */
class Boat
{
           private Deck deck;
           private Body body;

    public Boat() {
           System.out.print("\nПуста шлюпка була створенна");

           FileUtil.appendToFile("Логи.txt", "Шлюпка була створенна");
    }

    public Boat(Deck deck, Body body) {
        this.deck = deck;
        this.body = body;

        FileUtil.appendToFile("Логи.txt", "Шлюпка була створенна");
    }

    public Boat(Boat other) {
        this.deck = new Deck(other.deck);
        this.body = new Body(other.body);

        FileUtil.appendToFile("Логи.txt", "Шлюпка була створенна");
    }

    {
        this.deck = new Deck();
        this.body = new Body();
    }

    public Deck getDeck() {
        return deck;
    }

    public void setDeck(Deck deck) {
        this.deck = deck;
    }

    public Body getBody() {
        return body;
    }

    public void setBody(Body body) {
        this.body = body;
    }

}

/**
 * This class represents a boat's deck configuration, including the capacity for sailors and passengers.
 * The deck can have specific capacities for sailors and passengers.
 */
class Deck
{
           private Integer sailorsCapacity;
           private Integer passengerCapacity;

            public Deck() {
                       System.out.print("\nПуста палуба була створенна");

                       FileUtil.appendToFile("Логи.txt", "Було створене " + this.toString());
```

```
    }

    public Deck(Integer sailorsCapacity, Integer passengerCapacity) {
        this.sailorsCapacity = sailorsCapacity;
        this.passengerCapacity = passengerCapacity;

        FileUtil.appendToFile("Логи.txt", "Було створене " + this.toString());
    }

    public Deck(Deck other) {
        this.sailorsCapacity = other.sailorsCapacity;
        this.passengerCapacity = other.passengerCapacity;

        FileUtil.appendToFile("Логи.txt", "Було створене " + this.toString());
    }
    {
        this.sailorsCapacity = 0;
        this.passengerCapacity = 0;
    }

    public Integer getSailorsCapacity() {
        return sailorsCapacity;
    }

    public void setSailorsCapacity(Integer sailorsCapacity) {
        this.sailorsCapacity = sailorsCapacity;
    }

    public Integer getPassengerCapacity() {
        return passengerCapacity;
    }

    public void setPassengerCapacity(Integer passengerCapacity) {
        this.passengerCapacity = passengerCapacity;
    }

    public String toHtml() {
        return "<p>Палуба: </p>" +
                "<p>кількість веслярів = " + sailorsCapacity + "</p>" +
                "<p>кількість пасажирів=" + passengerCapacity + "</p>";
    }

    @Override
    public String toString() {
        return "Палуба{" +
                "кількість веслярів='" + sailorsCapacity + '\'' +
                ", кількість пасажирів='" + passengerCapacity + '\'' +
                '}';
    }
}

/**
 * This class represents the body of a boat, including its material, durability, quality, and purpose.
 * The body can be configured with specific material, durability, quality, and purpose.
 */
class Body extends Object {
    private String purpose;

    public Body() {
        super();
        System.out.print("\nПустий корпус був створенний");

        FileUtil.appendToFile("Логи.txt", "Був створений " + this.toString());
    }

    public Body(String material, Integer durability, Integer quality, Integer maxDurability, String purpose) {
        super(material, durability, quality, maxDurability);
        this.purpose = purpose;

        FileUtil.appendToFile("Логи.txt", "Був створений " + this.toString());
    }

    public Body(Body other) {
        super(other);
        this.purpose = other.purpose;

        FileUtil.appendToFile("Логи.txt", "Був створений " + this.toString());
    }
    {
        this.purpose = "За замовчуванням";
    }

    public String getPurpose() {
        return purpose;
    }

    public void setPurpose(String purpose) {
        this.purpose = purpose;
    }

    public String toHtml() {
        return "<p>Корпус:</p>" +
                "<p>призначення = " + purpose + ", матеріал = " + this.getMaterial() + "</p>" +
                "<p> справність = " + this.getDurability() + ", якість = " + this.getQuality() + "</p>";
    }
```

```java
    @Override
    public String toString() {
        return "Корпус{" +
                "призначення='" + purpose + '\'' +
                ", матеріал='" + super.getMaterial() + '\'' +
                ", справність=" + super.getDurability() +
                ", maxСправність=" + super.getMaxDurability() +
                ", якість=" + super.getQuality() +
                '}';
    }
}

/**
 * Moves the boat forward by rowing with both oars.
 *
 * @param sailer The sailor controlling the oars
 * @return True if the boat successfully moved forward; otherwise, false
 */
class PairOfOars implements BoatMovingFunctions
{
           private Oar rightOar;
           private Oar leftOar;

    public PairOfOars() {
           System.out.print("\nПуста пара весел була створенна");

           FileUtil.appendToFile("Логи.txt", "Пара весел була створенна");
    }

    public PairOfOars(Oar rightOar, Oar leftOar) {
        this.rightOar = rightOar;
        this.leftOar = leftOar;

        FileUtil.appendToFile("Логи.txt", "Пара весел була створенна");
    }

    public PairOfOars(PairOfOars other) {
        this.rightOar = new Oar(other.rightOar);
        this.leftOar = new Oar(other.leftOar);

        FileUtil.appendToFile("Логи.txt", "Пара весел була створенна");
    }

    {
        this.rightOar = new Oar();
        this.leftOar = new Oar();
    }

    /**
     * Moves the boat forward by rowing with both oars.
     *
     * @param sailer The sailer controlling the boat.
     * @param boat   The boat to be moved.
     * @return True if the boat moves forward successfully; otherwise, false.
     */
    public Boolean MoveForward(Sailer sailer, Boat boat) {
        return leftOar.Rowing(sailer, boat) & rightOar.Rowing(sailer, boat);
    }

    /**
     * Moves the boat backward by rowing with both oars.
     *
     * @param sailer The sailer controlling the boat.
     * @param boat   The boat to be moved.
     * @return True if the boat moves backward successfully; otherwise, false.
     */
    public Boolean MoveBackward(Sailer sailer, Boat boat) {
        return leftOar.Rowing(sailer, boat) & rightOar.Rowing(sailer, boat);
    }

    /**
     * Moves the boat to the left by rowing with the right oar.
     *
     * @param sailer The sailer controlling the boat.
     * @param boat   The boat to be moved.
     * @return True if the boat moves left successfully; otherwise, false.
     */
    public Boolean MoveLeft(Sailer sailer, Boat boat) {
        return rightOar.Rowing(sailer, boat);
    }

    /**
     * Moves the boat to the right by rowing with the left oar.
     *
     * @param sailer The sailer controlling the boat.
     * @param boat   The boat to be moved.
     * @return True if the boat moves right successfully; otherwise, false.
     */
    public Boolean MoveRight(Sailer sailer, Boat boat) {
        return leftOar.Rowing(sailer, boat);
    }

    public Oar getRightOar() {
        return rightOar;
```

```
    }

    public void setRightOar(Oar rightOar) {
        this.rightOar = rightOar;
    }

    public Oar getLeftOar() {
        return leftOar;
    }

    public void setLeftOar(Oar leftOar) {
        this.leftOar = leftOar;
    }

    public String toHtml() {
        return "<p>Праве весло:</p>" + this.rightOar.toHtml()
                    + "<p>Ліве весло:</p>" + this.leftOar.toHtml() ;
    }
}

/**
 * This class represents an oar used for rowing a boat. It includes information about the oar's material,
 * durability, quality, and blade type.
 */
class Oar extends Object
{
            public String blade;
            public Oar() {
                        System.out.print("\nПусте весло було створене");

                        FileUtil.appendToFile("Логи.txt", "Було створене " + this.toString());
    }

    public Oar(String material, Integer durability, Integer quality, Integer maxDurability, String blade) {
            super(material, durability, quality, maxDurability);
            this.blade  = blade;

            FileUtil.appendToFile("Логи.txt", "Було створене " + this.toString());
    }

    public Oar(Oar other) {
            super(other);
            this.blade = other.blade;

            FileUtil.appendToFile("Логи.txt", "Було створене " + this.toString());
    }

    {
            this.blade = "Тесак";
    }

    public String getBlade() {
        return blade;
    }

    public void seBlade(String blade) {
        this.blade = blade;
    }

    /**
     * Simulates rowing action, reducing the durability of the oar, stamina of the sailer, and boat's body durability.
     *
     * @param sailer The sailer performing the rowing.
     * @param boat   The boat in which rowing is taking place.
     * @return True if rowing is successful and all conditions are met; otherwise, false.
     */
    public Boolean Rowing(Sailer sailer, Boat boat) {
        if (super.durability > 0 && sailer.getStamina() > 0 && boat.getBody().getDurability() > 0) {
            // Reduce oar's durability
            super.durability -= ((int) (Math.random() * 11) * 100) / super.quality;

            // Reduce sailer's stamina
            sailer.setStamina(sailer.getStamina() - (((int) (Math.random() * 11) * 100) / sailer.getPower()));

            // Reduce boat's body durability
            boat.getBody().setDurability(boat.getBody().getDurability() -
                    ((int) (Math.random() * 11) * 100) / boat.getBody().getQuality());

            return true;
        }
        return false;
    }


    public String toHtml() {
        return "<p> лопать = " + blade + ", матеріал = " + this.getMaterial() + "</p>" +
               "<p> справність = " + this.getDurability() + ", якість = " + this.getQuality() + "</p>";
    }

    @Override
    public String toString() {
        return "Весло{" +
                "лопать='" + blade + '\'' +
                ", матеріал='" + super.getMaterial() + '\'' +
                ", справність=" + super.getDurability() +
```

```
                ", maxСправність=" + super.getMaxDurability() +
                ", якість=" + super.getQuality() +
                '}';
    }
}

/**
 * This abstract class represents a generic object with material, durability, quality, and maximum durability attributes.
 * It serves as a base class for various objects in the application.
 */
abstract class Object {
    private String material;
    protected Integer durability;
    private Integer maxDurability;
    protected Integer quality;

    public Object() {
        System.out.print("\nПустий об'єкт був створений");

        FileUtil.appendToFile("Логи.txt", "Був створений " + this.toString());
    }

    public Object(String material, Integer durability, Integer quality, Integer maxDurability) {
        this.material = material;
        this.durability = durability;
        this.quality = quality;
        this.maxDurability = maxDurability;

        FileUtil.appendToFile("Логи.txt", "Був створений " + this.toString());
    }

    public Object(Object other) {
        this.material = other.material;
        this.durability = other.durability;
        this.quality = other.quality;
        this.maxDurability = other.maxDurability;

        FileUtil.appendToFile("Логи.txt", "Був створений " + this.toString());
    }

    {
        this.material = "Невідомо";
        this.durability = 1000;
        this.quality = 30;
    }

    /**
     * Repairs the object's durability if it is below the desired level.
     *
     * @return True if the object was repaired; otherwise, false
     */
    public Boolean Repair() {
        if (((this.maxDurability * this.quality) / 100) > this.durability) {
            this.durability = (this.maxDurability * 100) / this.quality;
            return true;
        }
        return false;
    }

    public String getMaterial() {
        return material;
    }

    public void setMaterial(String material) {
        this.material = material;
    }

    public Integer getDurability() {
        return durability;
    }

    public void setDurability(Integer durability) {
        this.durability = durability;
    }

    public Integer getMaxDurability() {
        return maxDurability;
    }

    public void setMaxDurability(Integer maxDurability) {
        this.maxDurability = maxDurability;
    }

    public Integer getQuality() {
        return quality;
    }

    public void setQuality(Integer quality) {
        this.quality = quality;
    }


    public String toHtml() {
        return "Об'єкт{" +
                "матеріал='" + material + '\'' +
```

```
                ", справність='" + durability + '\'' +
                ", якість=" + quality +
                '}';
    }

    @Override
    public String toString() {
        return "Об'єкт{" +
                "матеріал='" + material + '\'' +
                ", справність=" + durability +
                ", maxСправність=" + maxDurability +
                ", якість=" + quality +
                '}';
    }

}

/**
 * The `Sailer` class represents a sailor who can row a boat. It extends the `Person` class.
 */
class Sailer extends Person
{
           private Integer stamina;
           private Integer maxStamina;
           private Integer power;
           private Integer experience;

    public Sailer() {
        super();
        System.out.print("\nПустий весляр був створений");

        FileUtil.appendToFile("Логи.txt", "Був створений " + this.toString());
    }

    public Sailer(String firstname, String lastname, Integer years, Integer stamina,
                  Integer power, Integer experience, Integer maxStamina) {
        super(firstname, lastname, years);
        this.stamina = stamina;
        this.power = power;
        this.experience = experience;
        this.maxStamina = maxStamina;

        FileUtil.appendToFile("Логи.txt", "Був створений " + this.toString());
    }

    public Sailer(Sailer other) {
        super(other);
        this.stamina = other.stamina;
        this.power = other.power;
        this.experience = other.experience;
        this.maxStamina = other.maxStamina;

        FileUtil.appendToFile("Логи.txt", "Був створений " + this.toString());
    }

    {
        this.stamina = 0;
        this.power = 0;
        this.experience = 0;
        this.maxStamina = 0;
    }

    /**
     * Restores the sailor's stamina to its maximum value.
     */
    public void Rest()
    {
           this.stamina = this.maxStamina;
    }

    public Integer getStamina() {
        return stamina;
    }

    public void setStamina(Integer stamina) {
        this.stamina = stamina;
    }

    public Integer getPower() {
        return power;
    }

    public void setPower(Integer power) {
        this.power = power;
    }

    public Integer getExperience() {
        return experience;
    }

    public void setExperience(Integer experience) {
        this.experience = experience;
    }

    public String toHtml()
```

```
    {
            return "<p>" + " ім'я = " + getFirstname() + ", прізвище = " + getLastname() + ",</p>" +
                       "<p>" + "вік = " + getYears() +", енергія = " + stamina + ",</p>" +
                       "<p>" + "потужність = " + power + ", досвід = " + experience + "</p>";
    }

    @Override
    public String toString() {
        return "Весляр{" +
                "ім'я='" + getFirstname() + '\'' +
                ", прізвище='" + getLastname() + '\'' +
                ", вік=" + getYears() +
                ", енергія=" + stamina +
                ", maxЕнергія=" + maxStamina +
                ", потужність=" + power +
                ", досвід=" + experience +
                '}';
    }
}

/**
 * The `Person` class represents an individual with a first name, last name, and age.
 */
class Person
{
            private String firstname;
            private String lastname;
            private Integer years;

    public Person() {
            System.out.print("\nПуста людина була створена");

            FileUtil.appendToFile("Логи.txt", "Була створена " + this.toString());
    }

    public Person(String firstname, String lastname, Integer years) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.years = years;

        FileUtil.appendToFile("Логи.txt", "Була створена " + this.toString());
    }

    public Person(Person other) {
        this.firstname = other.firstname;
        this.lastname = other.lastname;
        this.years = other.years;

        FileUtil.appendToFile("Логи.txt", "Була створена " + this.toString());
    }

    {
        this.firstname = "Невідомо";
        this.lastname = "Невідомо";
        this.years = 0;
    }

    public String getFirstname() {
        return firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getLastname() {
        return lastname;
    }

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }

    public Integer getYears() {
        return years;
    }

    public void setYears(Integer years) {
        this.years = years;
    }

    public String toHtml()
    {
            return "ім'я=" + firstname +", прізвище=" + lastname + ", вік=" + years;
    }

            @Override
            public String toString() {
                    return "Людина{" +
                                                "ім'я='" + firstname + '\'' +
                                                ", прізвище='" + lastname + '\'' +
                                                ", вік=" + years +
                                                '}';
            }
}
```

Лістинг 2.2 Клас-драйвер програми.

```
package CI_306.Shyryi.Lab2;

import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;

/**
 * The `BoatOnOarsDriver` class is the main entry point for the BoatOnOars application.
 * It demonstrates the creation of a boat on oars and simulates rowing actions.
 */
public class BoatOnOarsDriver
{
    /**
     * The main method of the application. It creates a boat on oars and simulates rowing actions.
     *
     * @param args The command-line arguments (not used in this application)
     * @throws FileNotFoundException If a file is not found while reading
     */
            public static void main(String[] args) throws FileNotFoundException
            {
                        List<Person> people = new ArrayList<>();
                        people.add(new Person("Святослав", "Куйзалізо", 30));
                        people.add(new Person("Іван", "Розлука", 25));
                       BoatOnOars boat= new BoatOnOars(
                       "Повітряний змій", new Boat(new Deck(2,2), new Body("Дерево", 1000, 80, 1000, "Sport")),
                       new PairOfOars[] {
new PairOfOars(new Oar("Дерево", 1000, 50, 1050, "Тесак"), new Oar("Дерево", 1000, 50, 1050, "Тесак")),
new PairOfOars(new Oar("Дерево", 2000, 40, 2050, "Тесак"), new Oar("Дерево", 2000, 60, 2050, "Тесак"))},
                       new Sailer[] {
                       new Sailer("Роман", "Поворозник", 30, 1000, 50, 3, 1000),
                       new Sailer("Данило", "Хмілевський", 25, 1000, 45, 2, 1000),}, people);

                       boat.Rowing();


            }
}
```

Лістинг 2.3 Інтерфейс імплементації руху.

```
package CI_306.Shyryi.Lab2;

/**
 * An interface for boat moving functions.
 */
public interface BoatMovingFunctions {

    /**
     * Move forward.
     *
     * @param sailer The sailer object.
     * @param boat   The boat object.
     * @return True if successful, false otherwise.
     */
    Boolean MoveForward(Sailer sailer, Boat boat);

    /**
     * Move backward.
     *
     * @param sailer The sailer object.
     * @param boat   The boat object.
     * @return True if successful, false otherwise.
     */
    Boolean MoveBackward(Sailer sailer, Boat boat);

    /**
     * Move left.
     *
     * @param sailer The sailer object.
```

```
     * @param boat   The boat object.
     * @return True if successful, false otherwise.
     */
    Boolean MoveLeft(Sailer sailer, Boat boat);

    /**
     * Move right.
     *
     * @param sailer The sailer object.
     * @param boat   The boat object.
     * @return True if successful, false otherwise.
     */
    Boolean MoveRight(Sailer sailer, Boat boat);
}
```

Лістинг 2.4. Клас організації коректної роботи з файлами.

```
package CI_306.Shyryi.Lab2;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

/**
 * The `FileUtil` class provides utility methods for reading and appending to files.
 */
public class FileUtil {

    /**
     * Reads the content of a file and prints it to the console.
     *
     * @param fileName The name of the file to be read
     * @param logText  A string to store each line of the file content
     */
    public static void readFile(String fileName, String logText) {
        try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
            while ((logText = reader.readLine()) != null) {
                System.out.println(logText);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Appends text to a specified file with a timestamp.
     *
     * @param fileName The name of the file to which the text will be appended
     * @param logText  The text to be appended to the file
     */
    public static void appendToFile(String fileName, String logText) {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss");
        LocalDateTime currentDateTime = LocalDateTime.now();
        try (FileWriter writer = new FileWriter(fileName, true)) {
            writer.write("\n" + logText +" [" + currentDateTime.format(formatter)+"]");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

РЕЗУЛЬТАТИ ВИКОНАННЯ

Початковий вигляд програми наведено на рисунку 2.1.

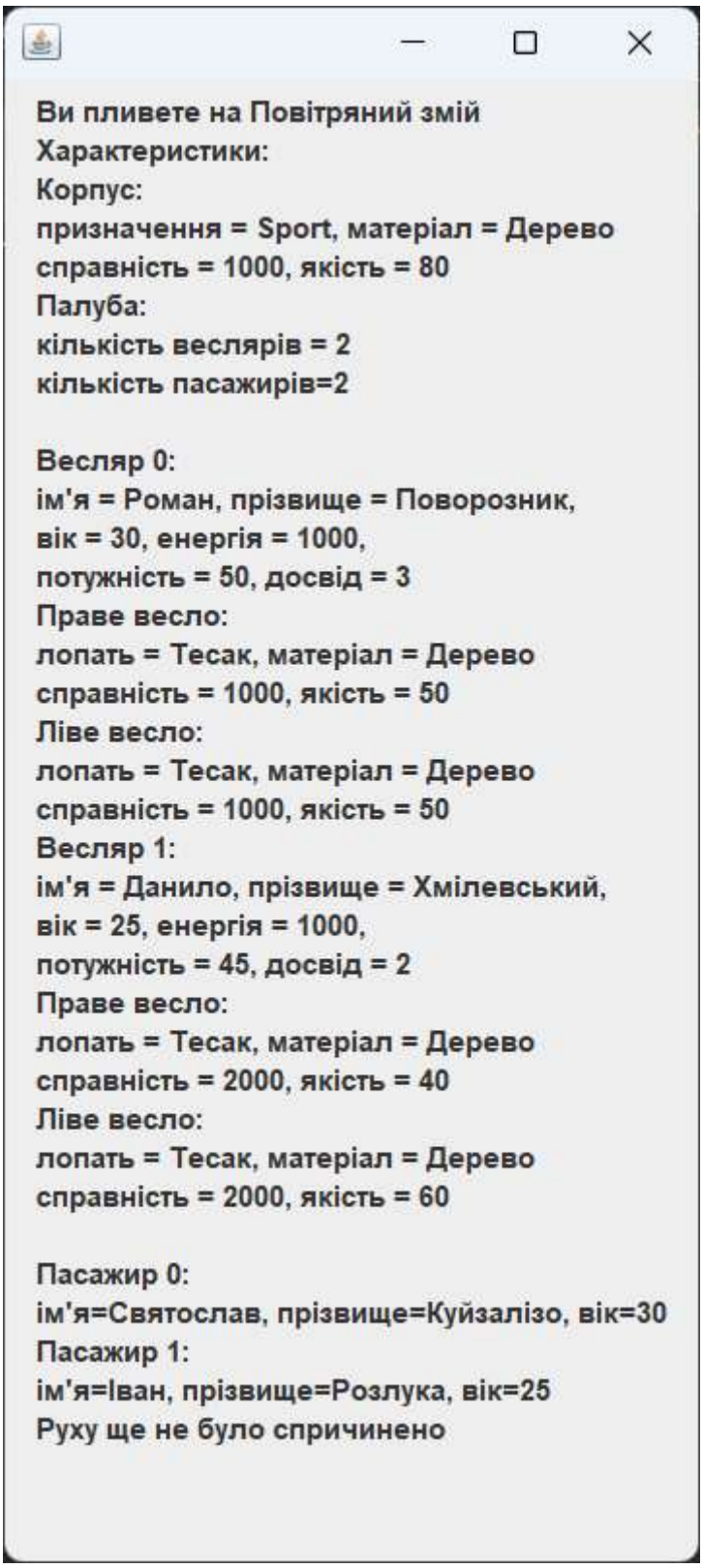

Рисунок 2.1. Початковий вигляд програми.

У веслярів та весел є характеристики, які впливають на плавання, а саме ті що видно на описі у програмі на рисунку 2.1. Основні характеристики це енергія для веслярів та справність для весел, відповідно, якщо вони не будуть більше

нуля, то неможливо буде. З кожним разом енергія та справність падають, як це зображено на рисунку 2.2.

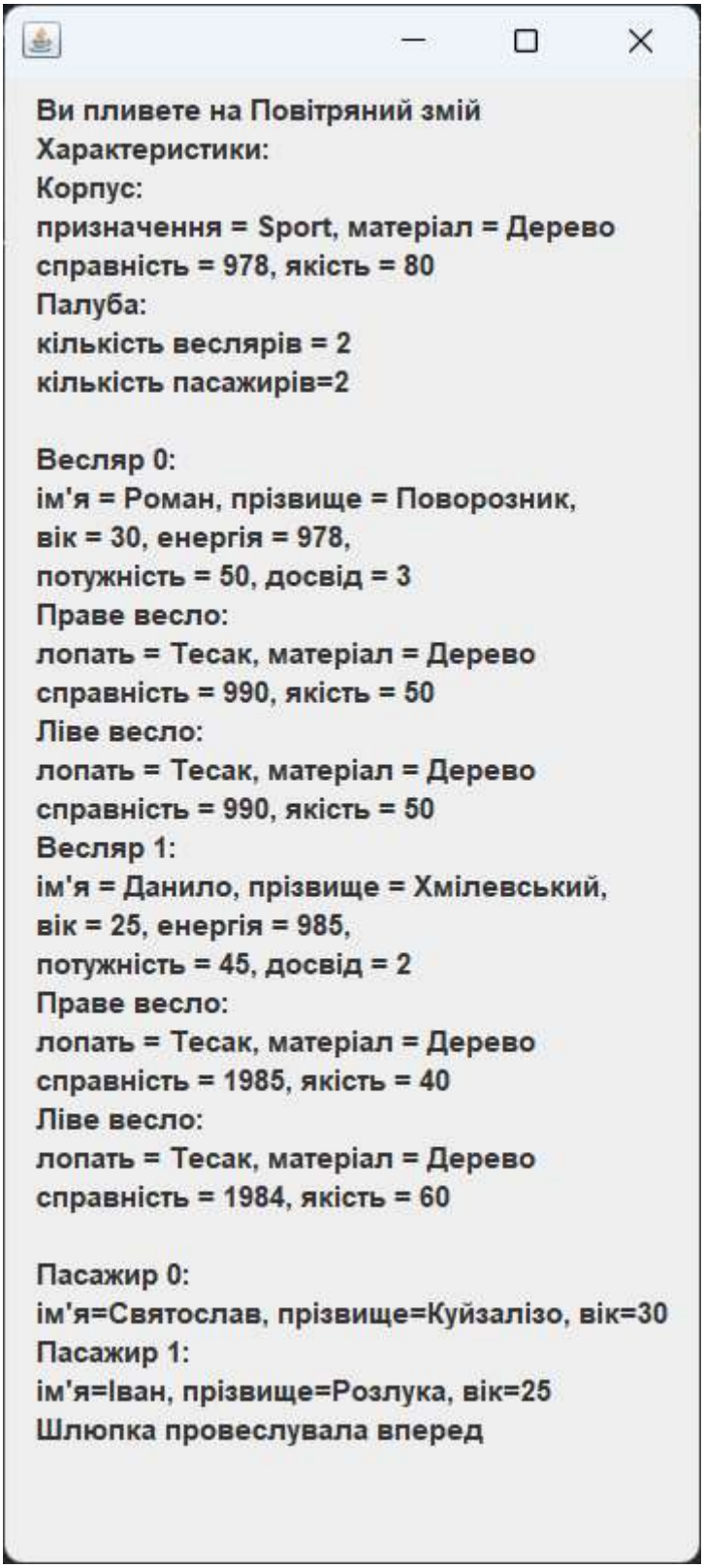

Рисунок 2.2. Характеристики після веслування вперед.

Як бачимо, при веслуванні вперед тратиться справність і правого, і лівого весла. Якщо веслування буде відбуватися в конкретну сторону, то буде тратитись справність лише протилежного до напрямку весла, як це зображено на рисунку 2.3.

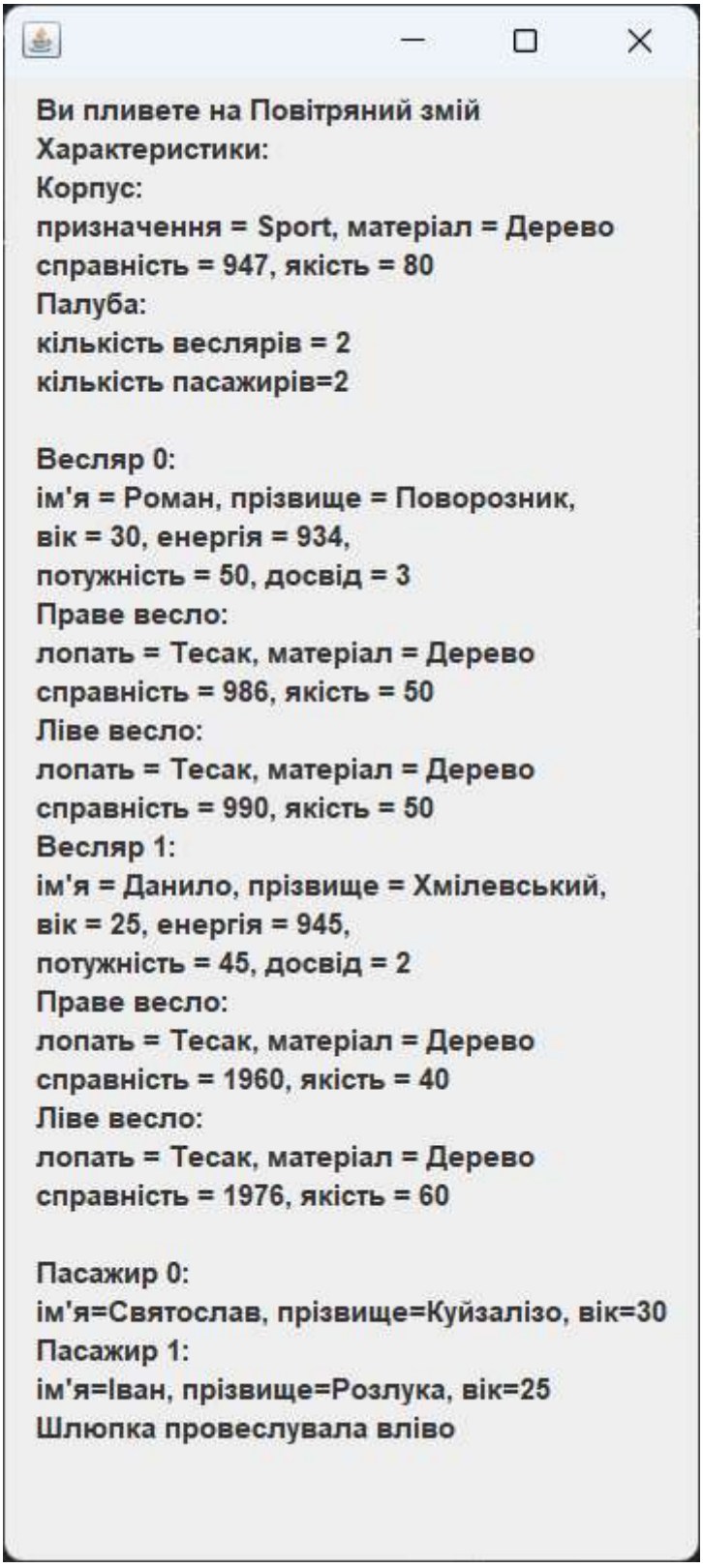

Рисунок 2.3. Характеристики після веслування вліво.

Веслувати також можна вправо та назад. Загалом, втрата справності залежить від інших характеристик весла, як максимальна можливість ремонту. Схожа, механіка працює з веслярем та корпусом. Відповідно, при поломці весла, корпусу, або виснаженні весляра необхідно здійснити ремонт або відпочинок як це зображено на рисунках 2.3 та 2.4.

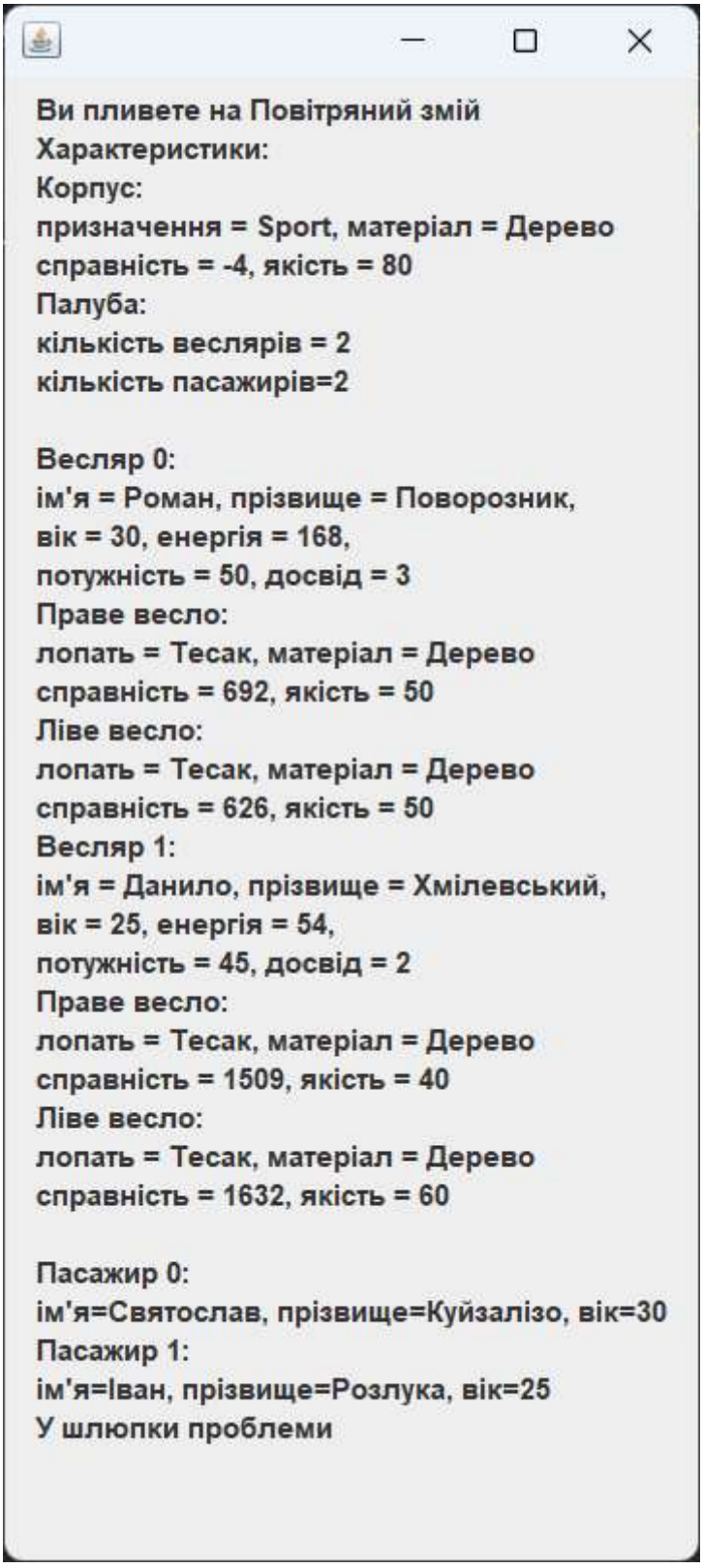

Рисунок 2.4. Програма після поломки корпусу.

Ремонт шлюпки та відпочинок екіпажу здійснюється натиском клавіші “R”, а керування плаванням стрілками клавіатури. Програма після натиску клавіші “R” наведена на рисунку 2.5. Як бачимо корпус мав хорошу якість, отож він відремонтувався на кращу якість, а оскільки весла мають не хорошу якість, відносно корпусу, то ремонт не здійснився, оскільки це би лише зашкодило справності.

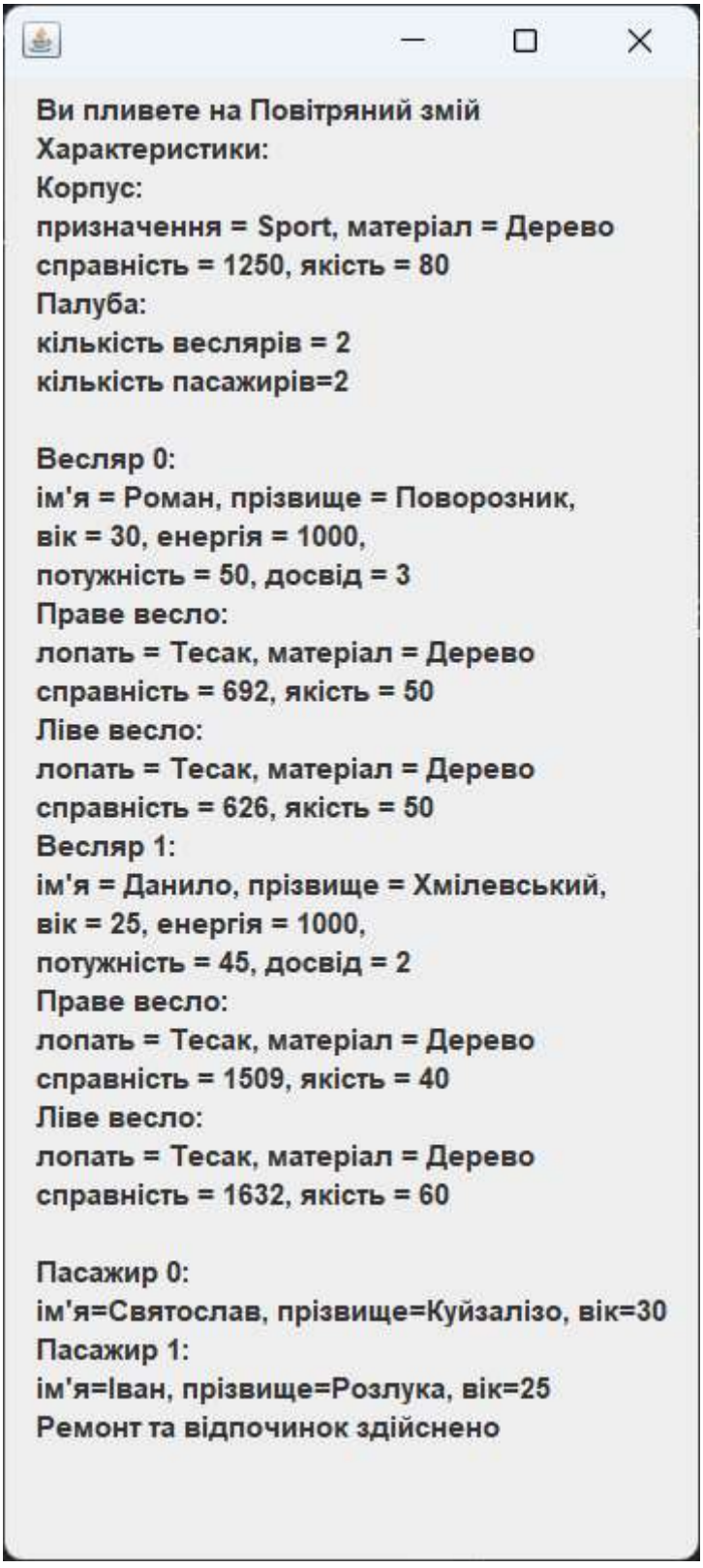

Рисунок 2.5. Програма після натиску клавіші “R”.

Відповідно до завдання, програма зберігає логи своєї роботи, які наведені на рисунку 2.6.

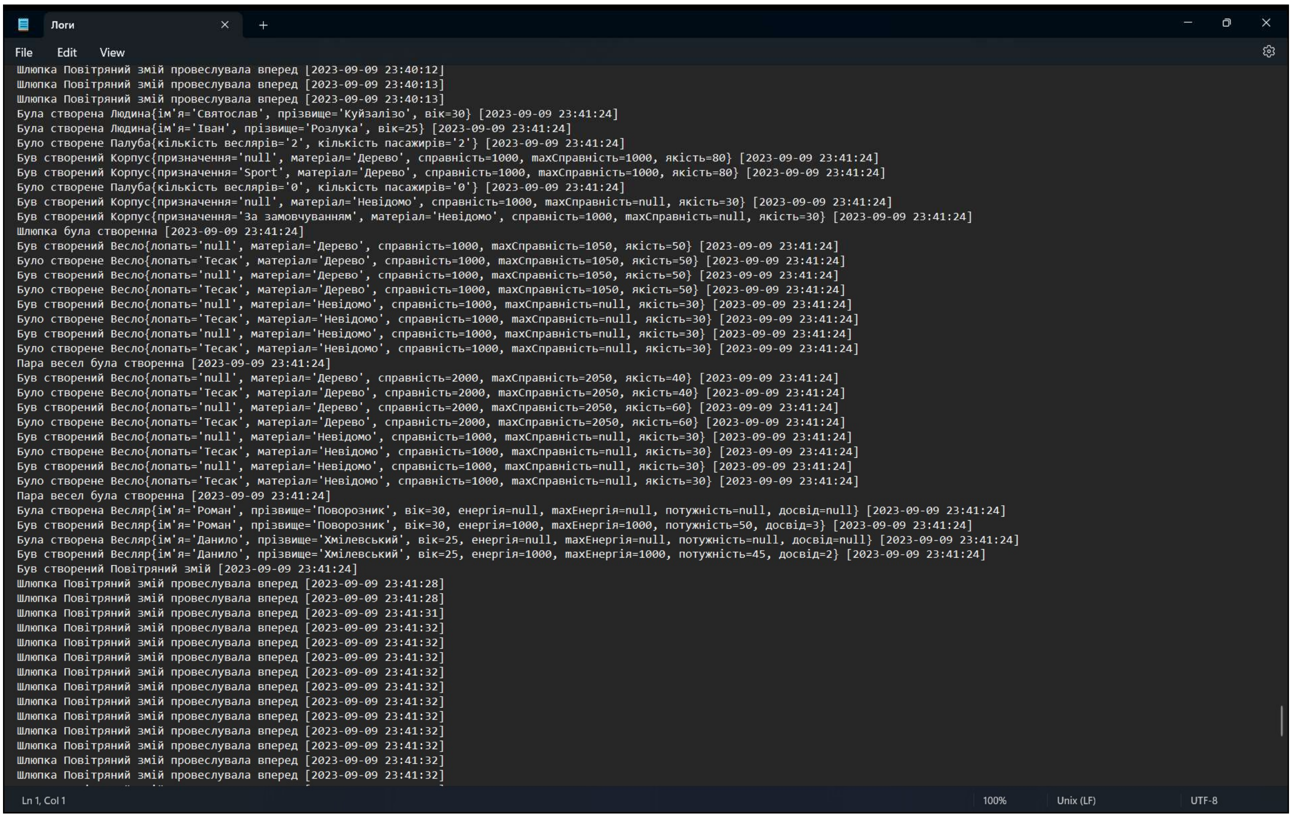

Рисунок 2.6. Логи програми.

ДОКУМЕНТАЦІЯ

Згенерував документацію, фрагмент якої навів на рисунку 2.7.

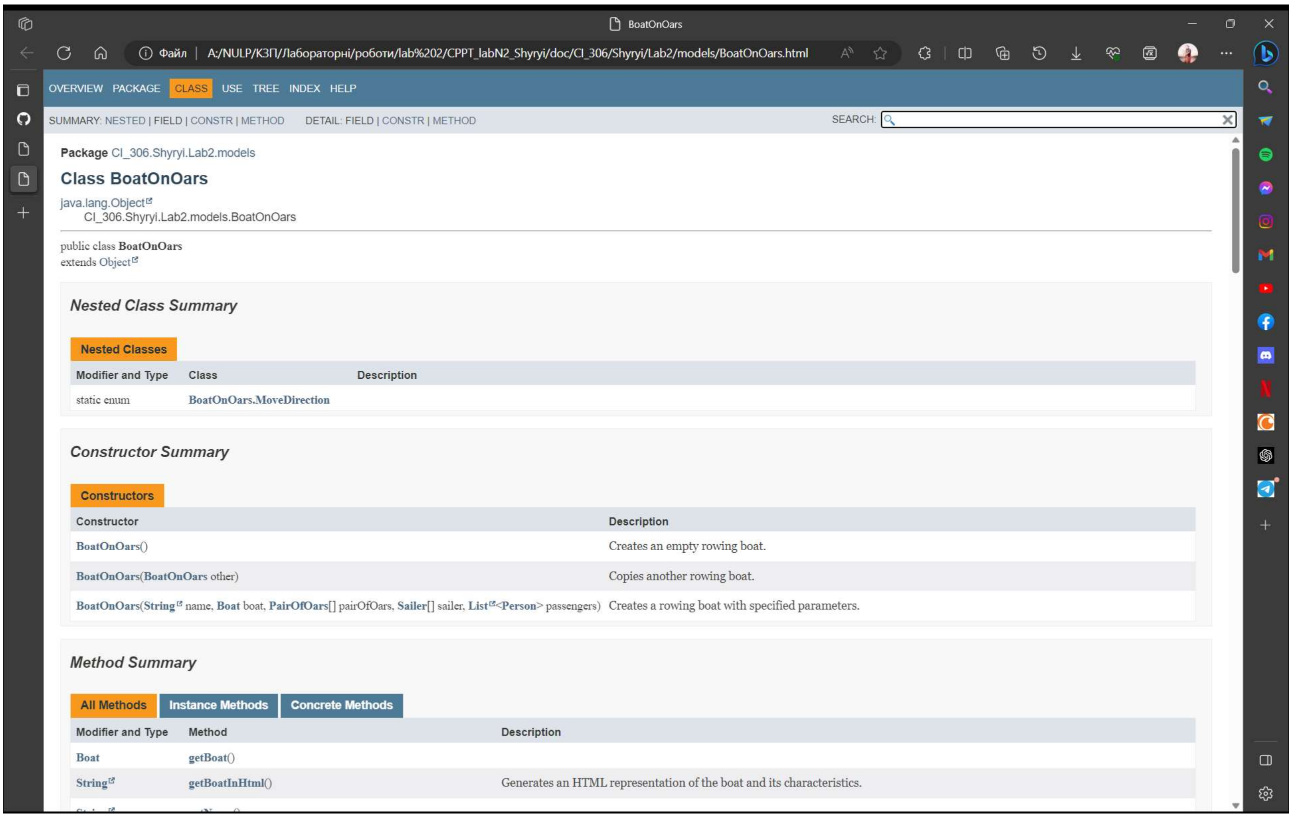

Рисунок 2.7. Фрагмент документації програми.

ВІДПОВІДІ НА КОНТРОЛЬНІ ПИТАННЯ

СИНТАКСИС ВИЗНАЧЕННЯ КЛАСУ.

Синтаксис наведений у лістингу 2.5.

Лістинг 2.5

```
public class MyClass {
    // Тіло класу
}
```

СИНТАКСИС ВИЗНАЧЕННЯ МЕТОДУ.

Синтаксис наведений у лістингу 2.6.

Лістинг 2.6.

```
[модифікатори] [тип повернення] ім'я_методу([параметри])
{
    // Тіло методу
}
```

СИНТАКСИС ОГОЛОШЕННЯ ПОЛЯ.

Синтаксис наведений у лістингу 2.7.

Лістинг 2.7.

```
[модифікатори] тип_даних ім'я_поля;
```

ЯК ОГОЛОСИТИ ТА ІНІЦІАЛІЗУВАТИ КОНСТАНТНЕ ПОЛЕ?

В Java константне поле оголошується за допомогою ключового слова **final**, як зображено у лістингу 2.8.

Лістинг 2.8.

```
public final int MY_CONSTANT = 42;
```

ЯКІ Є СПОСОБИ ІНІЦІАЛІЗАЦІЇ ПОЛІВ?

Поля можуть бути ініціалізовані при оголошенні, в конструкторі класу або в блоках ініціалізації. Приклад наведено у лістингу 2.9.

Лістинг 2.9.

```
public class MyClass {
    public int field1 = 10; // Ініціалізація при оголошенні
    public int field2;

    public MyClass() {
        field2 = 20; // Ініціалізація в конструкторі
    }
}
```

СИНТАКСИС ВИЗНАЧЕННЯ КОНСТРУКТОРА.

Конструктор має те ж ім'я, що і клас, і не має типу повернення. Ось приклад, що наведений у лістингу 2.10.

Лістинг 2.10.

```
public MyClass() {
    // Тіло конструктора
}
```

СИНТАКСИС ОГОЛОШЕННЯ ПАКЕТУ.

Пакет оголошується на початку файлу програми, як наведено у лістингу 2.11.

Лістинг 2.11.

```
package mypackage;
```

ЯК ПІДКЛЮЧИТИ ДО ПРОГРАМИ КЛАСИ, ЩО ВИЗНАЧЕНІ В ЗОВНІШНІХ ПАКЕТАХ?

Використовується імпорт для включення класів з інших пакетів. Наприклад, як у лістингу .

Лістинг 2.12.

```
import mypackage.MyClass;
```

В ЧОМУ СУТЬ СТАТИЧНОГО ІМПОРТУ ПАКЕТІВ?

Статичний імпорт дозволяє вам використовувати статичні члени класу без звернення до самого класу. Наприклад, у лістингу 2.13.

Лістинг 2.13.

```
import static mypackage.MyClass.myStaticMethod;
```

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 дослідив 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.

Вивчив синтаксис визначення класу, методу та поля, а також навчився оголошувати та ініціалізувати константні поля. Для ініціалізації полів було розглянуто різні способи, включаючи використання конструкторів.

Особливу увагу було приділено роботі з пакетами. Вивчив синтаксис оголошення пакету та способи підключення класів з інших пакетів. Також ознайомився з статичним імпортом пакетів і вимогами до файлів і каталогів при використанні пакетів.

У результаті виконаної роботи ми розробив програму, яка відповідає вимогам завдання. Мій клас містить необхідні поля, конструктори і методи, а також має можливість записувати протокол своєї діяльності у файл. Також автоматично згенерував документацію до розробленого пакету, що полегшить розуміння та використання мого коду.

Звіт про виконану роботу містить текст програми, результати її виконання і фрагмент згенерованої документації. У цій лабораторній роботі здобув важливі навички роботи з класами та пакетами в Java, які є фундаментальними для подальшого програмування в цій мові.