Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра «Електронних обчислювальних машин»

Звіт

з лабораторної роботи № 4

з дисципліни: «Кросплатформенні засоби програмування»

на тему: «Спадкування та інтерфейси»

Виконав:

студент групи КІ-306

Ширий Б. І.

Прийняв:

доцент кафедри ЕОМ

Іванов Ю. С.

Львів – 2022

МЕТОДИЧНІ ВІДОМОСТІ РОБОТИ

МЕТА

Ознайомитися з спадкуванням та інтерфейсами у мові Java.

ЗАВДАННЯ

№1

Написати та налагодити програму на мові Java, що розширює клас, що реалізований у лабораторній роботі №3, для реалізації предметної області заданої варіантом, в мене це. Суперклас, що реалізований у лабораторній роботі №3, зробити абстрактним. Розроблений підклас має забезпечувати механізми свого коректного функціонування та реалізовувати мінімум один інтерфейс. Програма має розміщуватися в пакеті Група.Прізвище.Lab4 та 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.

ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ

ВИХІДНИЙ КОД

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

- Основні моделі - лістинг 2.1,
- Motorboat Driver.java - лістинг 2.2,
- BoatMovingFunctions.java - лістинг 2.3,
- FileUtil.java - лістинг 2.4.

Лістинг 2.1. Код основної програми.

```
package CI_306.Shyryi.Lab4;

import java.awt.FlowLayout;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

/**
 * A class representing a motorboat with various attributes and functionalities.
 */
public class Motorboat {
	private String name;
	private Boat boat;
	private Motor motor;
	private Helmsman helmsman;
	private List<Person> passengers = new ArrayList<>();

	/**
     * Enum representing possible movement directions for the motorboat.
     */
	public enum MoveDirection {
	    LEFT,
	    RIGHT,
	    FORWARD,
	    BACKWARD
	}

    /**
     * Creates an empty rowing boat.
     */
    public Motorboat() {
	System.out.print("\nПустий моторний човен був створений");

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

    /**
     * Creates a Motorboat with specified parameters.
     *
     * @param name       The name of the motorboat.
     * @param boat       The boat component of the motorboat.
     * @param motor      The motor component of the motorboat.
     * @param helmsman   The helmsman steering the motorboat.
     * @param passengers The list of passengers on board.
     */
    public Motorboat(String name, Boat boat, Motor motor, Helmsman helmsman, List<Person> passengers) {
        this.name = name;
        this.boat = boat;
        this.motor = motor;
        this.helmsman = helmsman;
        this.passengers.addAll(passengers);
    }

    /**
     * Creates a copy of a Motorboat object.
     *
     * @param other The Motorboat object to copy.
     */
    public Motorboat(Motorboat other) {
        this.name = other.name;
        this.boat = new Boat(other.boat);
        this.motor = new Motor(other.motor);
        this.helmsman = new Helmsman(other.helmsman);
        this.passengers = new ArrayList<>(other.passengers);
    }

    /**
     * This initialization block sets initial values for the rowing boat.
     */
    {
        this.name = "";
        this.boat = new Boat();
        this.motor = new Motor();
        this.helmsman = new Helmsman();
    }

    /**
     * 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, 400);
		frame.setFocusable(true);

		JPanel panel = new JPanel();
```

```
        panel.setLayout(new FlowLayout());

                        JLabel page = new JLabel();
                        panel.add(page);
                        page.setText("<html>" + getBoatInHtml() + "<br/>" +
                                        getHelmsmanInHtml() + "<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/>" +
                                                                        getHelmsmanInHtml() + "<br/>" +
                                                                        getPassengersInHtml() + "<br/>" +
                                                                        (isRowingAbleTo(MoveDirection.FORWARD) ?
                                                "Моторний човен проплив вперед" : "У човна проблеми") + "</html>");
                                        FileUtil.appendToFile("Логи.txt", "Моторний човен " + name + " проплив вперед");
                                                break;
                                        case KeyEvent.VK_DOWN:
                                                page.setText("<html>" + getBoatInHtml() + "<br/>" +
                                                                        getHelmsmanInHtml() + "<br/>" +
                                                                        getPassengersInHtml() + "<br/>" +
                                                                        (isRowingAbleTo(MoveDirection.BACKWARD) ?
                                                "Моторний човен проплив назад": "У човна проблеми") + "</html>");
                                        FileUtil.appendToFile("Логи.txt", "Моторний човен " + name + " проплив назад");
                                                break;
                                        case KeyEvent.VK_RIGHT:
                                                page.setText("<html>" + getBoatInHtml() + "<br/>" +
                                                                        getHelmsmanInHtml() + "<br/>" +
                                                                        getPassengersInHtml() + "<br/>" +
                                                                        (isRowingAbleTo(MoveDirection.RIGHT) ?
                                                "Моторний човен проплив вправо" : "У човна проблеми") + "</html>");
                                        FileUtil.appendToFile("Логи.txt", "Моторний човен " + name + " проплив вправо");
                                                break;
                                        case KeyEvent.VK_LEFT:
                                                page.setText("<html>" + getBoatInHtml() + "<br/>" +
                                                                        getHelmsmanInHtml() + "<br/>" +
                                                                        getPassengersInHtml() + "<br/>" +
                                                                        (isRowingAbleTo(MoveDirection.LEFT) ?
                                                "Моторний човен проплив вліво" : "У човна проблеми") + "</html>");
                                        FileUtil.appendToFile("Логи.txt", "Моторний човен " + name + " проплив вліво");
                                                break;
                                        case KeyEvent.VK_R:
                                                motor.Repair();
                                                boat.getBody().Repair();
                                                helmsman.Rest();
                                                page.setText("<html>" + getBoatInHtml() + "<br/>" +
                                                                getHelmsmanInHtml() + "<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)
        {
            switch (moveDirection)
            {
            case FORWARD:
               if(!motor.MoveForward(helmsman, boat)){ return false; }
                break;
            case BACKWARD:
               if(!motor.MoveBackward(helmsman, boat)){ return false; }
                break;
            case RIGHT:
               if(!motor.MoveRight(helmsman, boat)){ return false; }
                break;
            case LEFT:
                        if(!motor.MoveLeft(helmsman, 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 getHelmsmanInHtml()
    {
            return "<p> Керманич: </p>"
                                            + helmsman.toHtml()
                                            + "<p> Мотор: </p>"
                                            + motor.toHtml();
    }

    /**
     * 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().getPassengerCapacity(); 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 Motor getMotor() {
        return motor;
    }

    public void setMotor(Motor motor) {
        this.motor = motor;
    }

    public Helmsman getHelmsman() {
        return helmsman;
    }

    public void setHelmsman(Helmsman helmsman) {
        this.helmsman = helmsman;
    }

    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", "Човен був створений");
    }

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

        FileUtil.appendToFile("Логи.txt", "Човен був створений");
    }

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

        FileUtil.appendToFile("Логи.txt", "Човен був створений");
    }

    {
        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 passengerCapacity;

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

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

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

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

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

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

    {
        this.passengerCapacity = 0;
    }

    public Integer getPassengerCapacity() {
        return passengerCapacity;
    }

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

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

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

/**
```

```java
 * 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>";
    }

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

/**
 * A class representing a motor for a boat, implementing BoatMovingFunctions.
 */
class Motor extends Object implements BoatMovingFunctions
{
           public String manufacturer;
           public Motor() {
                      System.out.print("\nПусте весло було створене");

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

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

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

    public Motor(Motor other) {
           super(other);
           this.manufacturer = other.manufacturer;

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

    {
           this.manufacturer = "Greenworks";
    }

    public String getManufacturer() {
        return manufacturer;
    }

    public void setManufacturer(String manufacturer) {
        this.manufacturer = manufacturer;
    }
```

```java
    /**
     * Moves the boat forward by rowing with the motor.
     *
     * @param helmsman The helmsman controlling the motorboat.
     * @param boat     The boat to be moved.
     * @return True if the boat moves forward successfully; otherwise, false.
     */
    public Boolean MoveForward(Helmsman helmsman, Boat boat) {
        return Rowing(helmsman, boat);
    }

    /**
     * Moves the boat backward by rowing with the motor.
     *
     * @param helmsman The helmsman controlling the motorboat.
     * @param boat     The boat to be moved.
     * @return True if the boat moves backward successfully; otherwise, false.
     */
    public Boolean MoveBackward(Helmsman helmsman, Boat boat) {
        return Rowing(helmsman, boat);
    }

    /**
     * Moves the boat to the left by rowing with the motor.
     *
     * @param helmsman The helmsman controlling the motorboat.
     * @param boat     The boat to be moved.
     * @return True if the boat moves left successfully; otherwise, false.
     */
    public Boolean MoveLeft(Helmsman helmsman, Boat boat) {
        return Rowing(helmsman, boat);
    }

    /**
     * Moves the boat to the right by rowing with the motor.
     *
     * @param helmsman The helmsman controlling the motorboat.
     * @param boat     The boat to be moved.
     * @return True if the boat moves right successfully; otherwise, false.
     */
    public Boolean MoveRight(Helmsman helmsman, Boat boat) {
        return Rowing(helmsman, boat);
    }

    /**
     * Simulates rowing action, reducing the durability of the oar, stamina of the helmsman, and boat's body durability.
     *
     * @param helmsman The helmsman 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(Helmsman helmsman, Boat boat) {
        if (super.durability > 0 && helmsman.getStamina() > 0 && boat.getBody().getDurability() > 0) {
            // Reduce oar's durability
            super.durability -= ((int) (Math.random() * 11) * 100) / super.quality;

            // Reduce sailer's stamina
            helmsman.setStamina(helmsman.getStamina() - (((int) (Math.random() * 11) * 100) / helmsman.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> виробник = " + manufacturer + ", матеріал = " + this.getMaterial() + "</p>" +
               "<p> справність = " + this.getDurability() + ", якість = " + this.getQuality() + "</p>";
    }

    @Override
    public String toString() {
        return "Мотор{" +
                "виробник='" + manufacturer + '\'' +
                ", матеріал='" + super.getMaterial() + '\'' +
                ", справність=" + super.getDurability() +
                ", maxСправність=" + super.getMaxDurability() +
                ", якість=" + super.getQuality() +
                '}';
    }
}

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 +
                ", махСправність=" + maxDurability +
                ", якість=" + quality +
                '}';
    }
}
/**
 * A class representing a helmsman who operates the boat, extending the Person class.
 */
class Helmsman extends Person
```

```
{
            private Integer stamina;
            private Integer maxStamina;
            private Integer power;
            private Integer experience;

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

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

    public Helmsman(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 Helmsman(Helmsman 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;
    }

    /**
     * Resets the helmsman'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 "Керманич{" +
                "ім'я='" + getFirstname() + '\'' +
                ", прізвище='" + getLastname() + '\'' +
                ", вік=" + getYears() +
                ", енергія=" + stamina +
                ", maxЕнергія=" + maxStamina +
                ", потужність=" + power +
                ", досвід=" + experience +
                '}';
    }
}

/**
```

```java
 * A class representing a person 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.Lab4;

import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
/**
 * A class representing a driver program for the Motorboat simulation.
 */
public class MotorboatDriver {
        /**
     * The main entry point of the program.
     *
     * @param args Command line arguments (not used in this program).
     * @throws FileNotFoundException If an error occurs while accessing files (not used in this
program).
     */
         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));
                   Motorboat boat= new Motorboat(
                  "Сталевий дракон", new Boat(new Deck(2),
                  new Body("Легка сталь", 1000, 80, 1000, "Воєнний")),
                                       new Motor("Залізо", 1000, 50, 1050, " Greenworks"),
                                       new Helmsman("Данило", "Хмілевський", 25, 1000, 45, 2,
1000),
                                       people);
                   boat.Rowing();
          }
}
```

Лістинг 2.3 Інтерфейс імплементації руху.

```
package CI_306.Shyryi.Lab4;
/**
 * An interface for boat moving functions.
 */
public interface BoatMovingFunctions {

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

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

    /**
     * Move left.
     *
     * @param helmsman The helmsman object.
     * @param boat   The boat object.
     * @return True if successful, false otherwise.
     */
    Boolean MoveLeft(Helmsman helmsman, Boat boat);

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

Лістинг 2.4. Клас організації коректної роботи з файлами.

```
package CI_306.Shyryi.Lab4;

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.

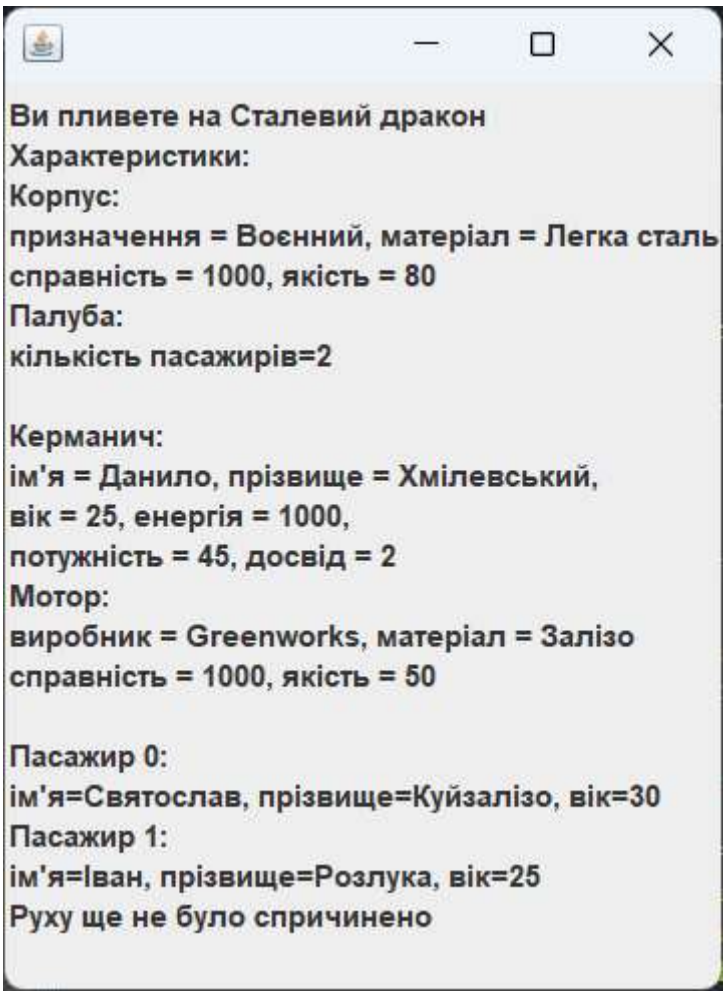


Рисунок 2.1. Початковий вигляд програми.

У керманичів та маторів є характеристики, які впливають на плавання, а саме ті що видно на описі у програмі на рисунку 2.1. Основні характеристики це енергія для керманичів та справність для моторів, відповідно, якщо вони не будуть більше нуля, то неможливо буде. З кожним разом енергія та справність падають, як це зображено на рисунку 2.2. Керманити також можна вліво, вправо та назад. Загалом, втрата справності залежить від інших характеристик мотора, як максимальна можливість ремонту. Схожа, механіка працює з керманичем та корпусом. Відповідно, при поломці мотора, корпусу, або виснаженні керманича необхідно здійснити ремонт або відпочинок як це зображено на рисунку 2.3. Ремонт шлюпки та відпочинок екіпажу здійснюється натиском клавіші “R”, а керування плаванням стрілками клавіатури. Програма після натиску клавіші “R” наведена на рисунку 2.4. 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.

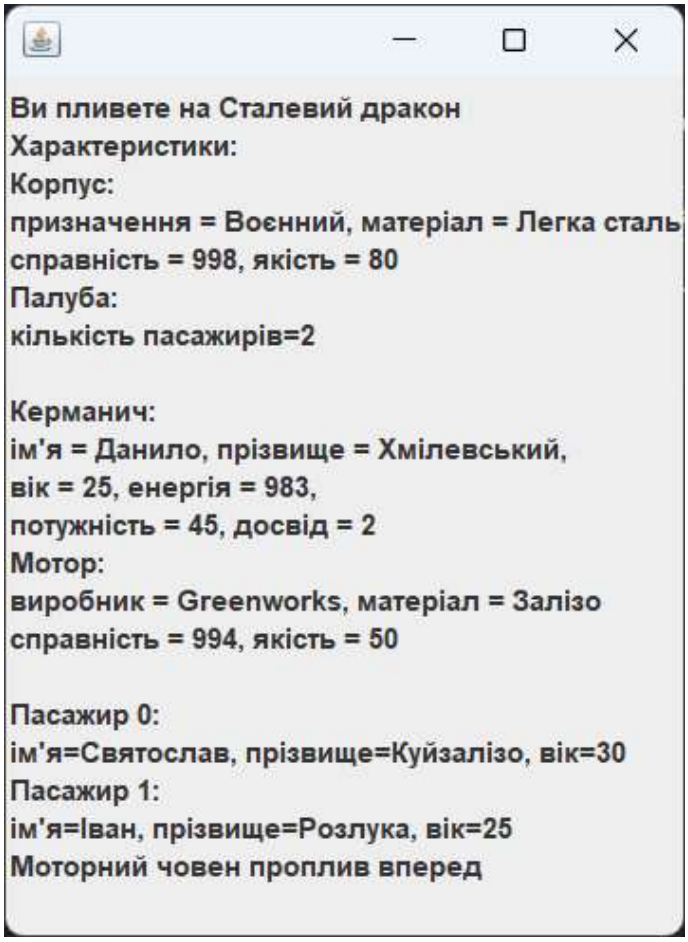


Рисунок 2.2. Характеристики після веслування вперед.

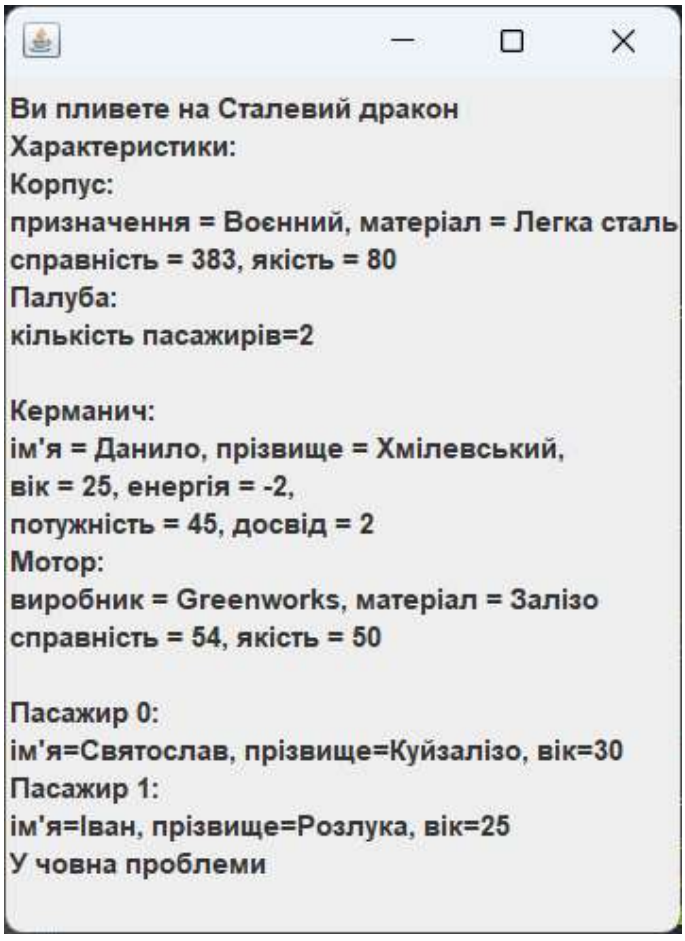


Рисунок 2.3. Програма після виснаження керманича.

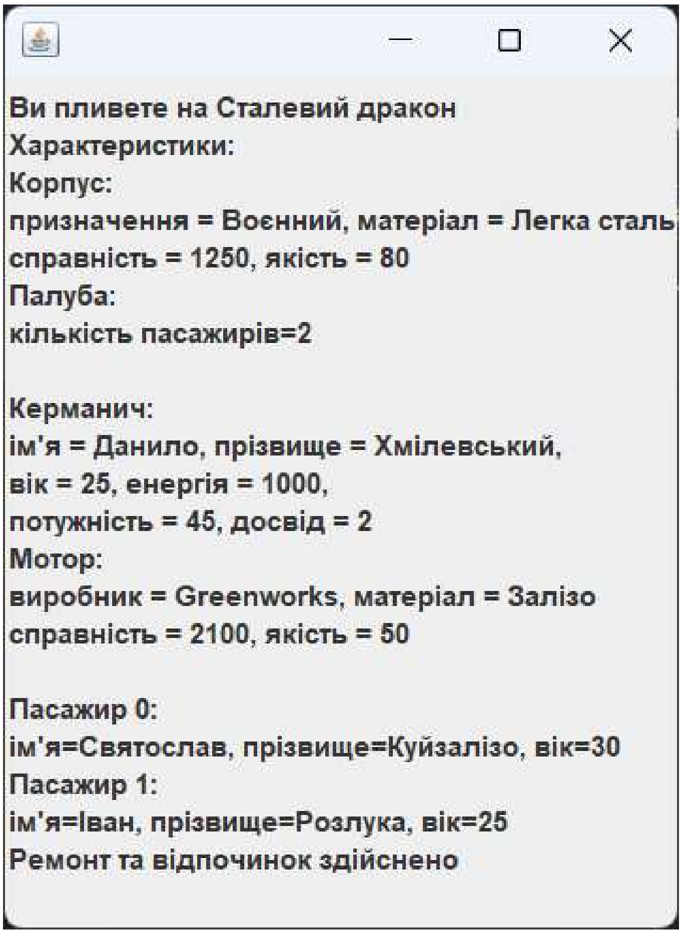
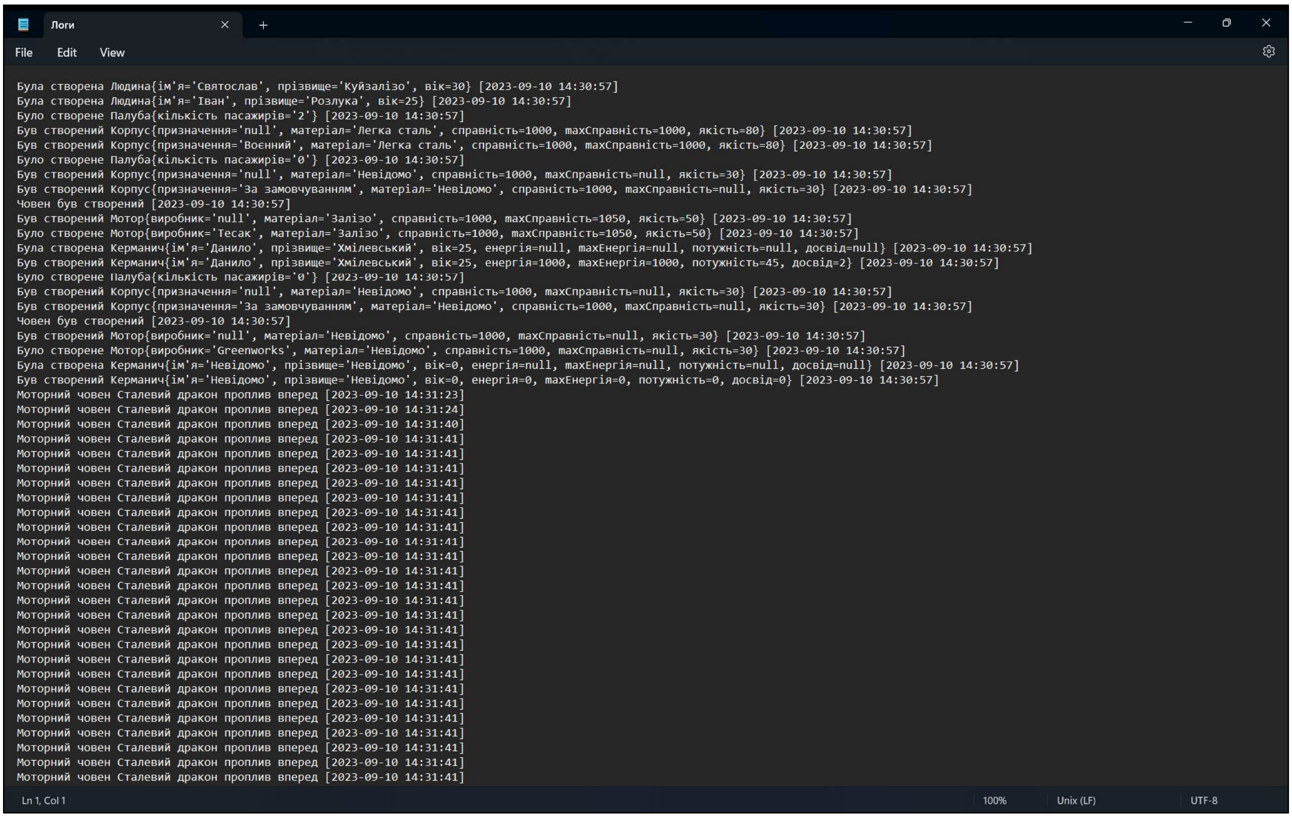


Рисунок 2.4. Програма після натиску клавіші “R”.

Відповідно до завдання, програма зберігає логи своєї роботи, які наведені на рисунку 2.5.




```
Була створена Людина{ім'я='Святослав', прізвище='Куйзалізо', вік=30} [2023-09-10 14:30:57]
Була створена Людина{ім'я='Іван', прізвище='Розлука', вік=25} [2023-09-10 14:30:57]
Було створене Палуба{кількість пасажирів='2'} [2023-09-10 14:30:57]
Був створений Корпус{призначення='null', матеріал='Легка сталь', справність=1000, maxСправність=1000, якість=80} [2023-09-10 14:30:57]
Був створений Корпус{призначення='Воєнний', матеріал='Легка сталь', справність=1000, maxСправність=1000, якість=80} [2023-09-10 14:30:57]
Було створене Палуба{кількість пасажирів='0'} [2023-09-10 14:30:57]
Був створений Корпус{призначення='null', матеріал='Невідомо', справність=1000, maxСправність=null, якість=30} [2023-09-10 14:30:57]
Був створений Корпус{призначення='За замовчуванням', матеріал='Невідомо', справність=1000, maxСправність=null, якість=30} [2023-09-10 14:30:57]
Човен був створений [2023-09-10 14:30:57]
Був створений Мотор{виробник='null', матеріал='Залізо', справність=1000, maxСправність=1050, якість=50} [2023-09-10 14:30:57]
Було створене Мотор{виробник='Тесак', матеріал='Залізо', справність=1000, maxСправність=1050, якість=50} [2023-09-10 14:30:57]
Була створена Керманич{ім'я='Данило', прізвище='Хмілевський', вік=25, енергія=null, maxЕнергія=null, потужність=null, досвід=null} [2023-09-10 14:30:57]
Був створений Керманич{ім'я='Данило', прізвище='Хмілевський', вік=25, енергія=1000, maxЕнергія=1000, потужність=45, досвід=2} [2023-09-10 14:30:57]
Було створене Палуба{кількість пасажирів='0'} [2023-09-10 14:30:57]
Був створений Корпус{призначення='null', матеріал='Невідомо', справність=1000, maxСправність=null, якість=30} [2023-09-10 14:30:57]
Був створений Корпус{призначення='За замовчуванням', матеріал='Невідомо', справність=1000, maxСправність=null, якість=30} [2023-09-10 14:30:57]
Човен був створений [2023-09-10 14:30:57]
Був створений Мотор{виробник='null', матеріал='Невідомо', справність=1000, maxСправність=null, якість=30} [2023-09-10 14:30:57]
Було створене Мотор{виробник='Greenworks', матеріал='Невідомо', справність=1000, maxСправність=null, якість=30} [2023-09-10 14:30:57]
Була створена Керманич{ім'я='Невідомо', прізвище='Невідомо', вік=0, енергія=null, maxЕнергія=null, потужність=null, досвід=null} [2023-09-10 14:30:57]
Був створений Керманич{ім'я='Невідомо', прізвище='Невідомо', вік=0, енергія=0, maxЕнергія=0, потужність=0, досвід=0} [2023-09-10 14:30:57]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:23]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:24]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:40]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
Моторний човен Сталевий дракон проплив вперед [2023-09-10 14:31:41]
```

Рисунок 2.5. Логи програми.

ДОКУМЕНТАЦІЯ

Згенерував документацію, фрагмент якої навів на рисунку 2.6.

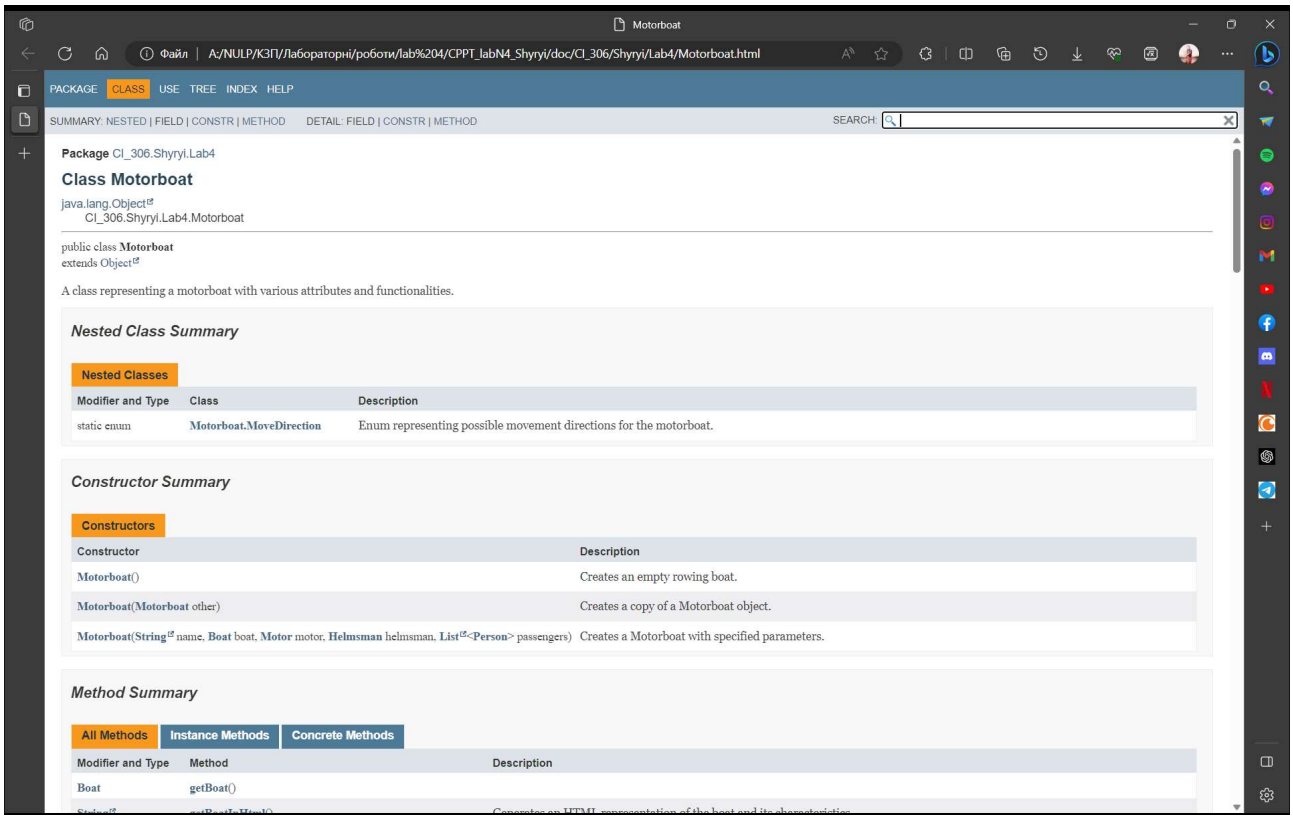


Рисунок 2.6. Фрагмент документації програми.

ВІДПОВІДІ НА КОНТРОЛЬНІ ПИТАННЯ

СИНТАКСИС РЕАЛІЗАЦІЇ СПАДКУВАННЯ.

В Java спадкування реалізується за допомогою ключового слова "extends" для класів і "implements" для інтерфейсів.

ЩО ТАКЕ СУПЕРКЛАС ТА ПІДКЛАС?

Суперклас - це клас, від якого успадковуються властивості і методи. Підклас - це клас, який успадковує властивості і методи від суперкласу.

ЯК ЗВЕРНУТИСЯ ДО ЧЛЕНІВ СУПЕРКЛАСУ З ПІДКЛАСУ?

Звернення до членів суперкласу з підкласу можливе за допомогою ключового слова "super" для методів і полів суперкласу.

КОЛИ ВИКОРИСТОВУЄТЬСЯ СТАТИЧНЕ ЗВ’ЯЗУВАННЯ ПРИ ВИКЛИКУ МЕТОДУ?

Статичне зв’язування відбувається під час компіляції, коли визначений метод викликається на основі типу посилання.

ЯК ВІДБУВАЄТЬСЯ ДИНАМІЧНЕ ЗВ’ЯЗУВАННЯ ПРИ ВИКЛИКУ МЕТОДУ?

Динамічне зв’язування відбувається під час виконання програми, коли метод вибирається на основі типу об'єкта, на який посилається посилання.

ЩО ТАКЕ АБСТРАКТНИЙ КЛАС ТА ЯК ЙОГО РЕАЛІЗУВАТИ?

Абстрактний клас - це клас, який не може бути інстанційованим і може містити абстрактні методи. Для створення абстрактного класу використовується ключове слово "abstract".

ДЛЯ ЧОГО ВИКОРИСТОВУЄТЬСЯ КЛЮЧОВЕ СЛОВО INSTANCEOF?

Ключове слово "instanceof" використовується для перевірки, чи об'єкт є екземпляром певного класу або інтерфейсу.

ЯК ПЕРЕВІРИТИ ЧИ КЛАС Є ПІДКЛАСОМ ІНШОГО КЛАСУ?

Для перевірки, чи клас є підкласом іншого класу, можна використовувати оператор "instanceof" або перевіряти наслідування за допомогою ключового слова "extends".

ЩО ТАКЕ ІНТЕРФЕЙС?

Інтерфейс в Java - це абстрактний клас, в якому всі методи є абстрактними (без реалізації), і всі поля є константами (final). В інтерфейсах описується поведінка, яку класи повинні реалізувати.

ЯК ОГОЛОСИТИ ТА ЗАСТОСУВАТИ ІНТЕРФЕЙС?

Інтерфейс оголошується за допомогою ключового слова "interface", а класи його реалізують за допомогою ключового слова "implements".

ВИСНОВОК

У цій лабораторній роботі суперклас, реалізований у попередній лабораторній роботі, використав як суперклас для цієї роботи, зробивши його абстрактним. Також створив підклас, який успадковує властивості і методи суперкласу і реалізує один інтерфейс. Ця робота дозволила навчитися використовувати спадкування і абстрактні класи для структурування коду і розширення функціональності. Документація до розробленого пакету допоможе зрозуміти іншим, як користуватися моїм кодом.