

Міністерство освіти і науки України  
Національний університет «Львівська політехніка»  
Кафедра «Електронних обчислювальних машин»



**Звіт**  
з лабораторної роботи № 3  
з дисципліни: «Кросплатформенні засоби програмування»  
**На тему: «Спадкування та інтерфейси»**

**Виконав:**

студент групи КІ-307  
Бажулін С.В.

**Прийняв:**

доцент кафедри ЕОМ  
Іванов Ю. С.

**Мета роботи:** ознайомитися з спадкуванням та інтерфейсами у мові Java.

### **Завдання(варіант №2):**

1. Написати та налагодити програму на мові Java, що розширює клас, що реалізований у лабораторній роботі №2, для реалізації предметної області заданої варіантом. Суперклас, що реалізований у лабораторній роботі №2, зробити абстрактним. Розроблений підклас має забезпечувати механізми свого коректного функціонування та реалізовувати мінімум один інтерфейс. Програма має розміщуватися в пакеті Група.Прізвище.Lab3 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.
2. Автоматично згенерувати документацію до розробленого пакету.
3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.
4. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.
5. Дати відповідь на контрольні запитання.

### **Індивідуальне завдання:**

#### **2. Багаторазовий космічний корабель**

### **Вихідний код програми:**

#### **Файл SpaceshipApp.java:**

```
package KI307.Bazhulin.Lab3;

import static java.lang.System.out;

import java.io.*;

/**
 * Spaceship Application class implement main methods for Spaceship class possibilities
 * demonstration
 * @author SERHIY BAZHULIN
 * @version 1.0
 */
```

```

public class SpaceshipApp {

    /**
     * @param args
     * @throws FileNotFoundException
     */
    public static void main(String[] args) throws FileNotFoundException {

        // TODO Auto-generated method stub

        ControlPanel.Direction direction = null;
        ReusableSpaceship reusableSpaceship = new ReusableSpaceship();
        reusableSpaceship.RefuelSpaceship(50);
        out.print(reusableSpaceship.getFuelStatus()+"\n");

        reusableSpaceship.CloseDoor();
        reusableSpaceship.StartSpaceship();
        reusableSpaceship.SetSpeed(100);
        reusableSpaceship.TakeOff();
        out.print(reusableSpaceship.getStatus()+"\n");

        reusableSpaceship.TurnRightSpaceship();
        direction = reusableSpaceship.getDirectionSpaceship();
        if(direction==ControlPanel.Direction.FORWARD)
            out.print("Forward direction\n");
        else if(direction == ControlPanel.Direction.LEFT)
            out.print("Left direction\n");
        else
            out.print("Right direction\n");

        reusableSpaceship.Land();
        out.print(reusableSpaceship.getStatus()+"\n");
        reusableSpaceship.SwitchOffSapceship();
    }
}

```

```

        reusableSpaceship.OpenDoor();
        reusableSpaceship.dispose();
    }

}

```

## Файл Spaceship.java

```

/**
 *
 */
package KI307.Bazhulin.Lab3;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

/**
 * Class <code>Spaceship</code> implements spaceship
 * @author SERHIY BAZHULIN
 * @version 1.0
 */
public abstract class Spaceship {

    protected Engine engine;
    private ControlPanel controlPanel;
    private Door door;
    protected PrintWriter fout;

    /**
     * Constructor
     * @throws FileNotFoundException
     */
    public Spaceship() throws FileNotFoundException

```

```

{
    engine = new Engine();
    controlPanel = new ControlPanel();
    door = new Door();

    fout = new PrintWriter(new File("Log.txt"));
}

/**
 * Constructor
 * @param <code>fuel</code> Fuel of engine
 * @throws FileNotFoundException
 */
public Spaceship(double fuel) throws FileNotFoundException
{
    engine = new Engine(fuel);
    controlPanel = new ControlPanel();
    door = new Door();

    fout = new PrintWriter(new File("Log.txt"));
}

/**
 * Method implements fuel of engine in spaceship
 * @param <code>fuel</code> Fuel of engine in spaceship
 */
public abstract void RefuelSpaceship(double fuel);

/**
 * Method starts the engine
 */
public abstract void StartSpaceship();

```

```

/**
 * Method switches off the engine
 */
public abstract void SwitchOffSapceship();

/**
 * Method returns fuel in engine
 * @return Fuel of engine
 */
public double getFuelStatus()
{
    double fuel = engine.getFuel();
    return fuel;
}

/**
 * Method returns status of engine
 * @return Engine status
 */
public boolean getEngineSpaceshipStatus()
{
    boolean status = engine.getEngineStatus();
    return status;
}

/**
 * Method turns left a spaceship
 */
public void TurnLeftSpaceship()
{
    controlPanel.TurnLeft();
}

```

```

        fout.print("Spaceship turned left\n");
    }

    /**
     * Method turns right a spaceship
     */
    public void TurnRightSpaceship()
    {
        controlPanel.TurnRight();
        fout.print("Spaceship turned right\n");
    }

    /**
     * Method forward a spaceship
     */
    public void ForwardSpacehip()
    {
        controlPanel.Forward();
        fout.print("Spaceship forward\n");
    }

    /**
     * Method sets a speed for spaceship
     * @param <code>speed</code> Speed for spaceship
     */
    public void SetSpeed(int speed)
    {
        controlPanel.SpeedChange(speed);
        fout.print("Spaceship speed set "+speed+"\n");
    }

    /**

```

```

    * Method returns speed of spaceship
    * @return Speed of spaceship
    */
    public int getSpeedSpaceship()
    {
        return controlPanel.getSpeed();
    }

    /**
    * Method opens door of spaceship
    */
    public void OpenDoor()
    {
        door.Open();
        fout.print("Door of spaceship was opened\n");
    }

    /**
    * Method closes door of spaceship
    */
    public void CloseDoor()
    {
        door.Close();
        fout.print("Door of spaceship was closed\n");
    }

    /**
    * Method returns status of door
    * @return Door status
    */
    public boolean getStatusDoor()
    {

```



```

        return door.getDoorStatus();
    }

    /**
     * Method returns spaceship direction
     * @return Direction of spaceship <code>ControlPanel.Direction</code> type
     */
    public ControlPanel.Direction getDirectionSpaceship()
    {
        return controlPanel.getDirection();
    }

    /**
     * Method releases used recourses
     */
    public void dispose()
    {
        fout.close();
    }
}

/**
 * @author BAZHULIN SERHIY
 * Class <code>Engine</code> implements engine
 */
class Engine
{
    private double fuel;
    private boolean isStarted;

    /**
     * Constructor

```

```

    * @throws FileNotFoundException
    */
    public Engine() throws FileNotFoundException
    {
        fuel = 0;
        isStarted = false;
    }

    /**
     * Constructor
     * @param fuel
     * @throws FileNotFoundException
     */
    public Engine(double fuel) throws FileNotFoundException
    {
        this.fuel = fuel;
    }

    /**
     * Constructor
     * @param fuel
     * @param isStarted
     * @throws FileNotFoundException
     */
    public Engine(double fuel, boolean isStarted) throws FileNotFoundException
    {
        this.fuel=fuel;
        this.isStarted = isStarted;
    }

    /**
     * Method starts the engine

```

```

*/
public void StartEngine()
{
    if(fuel>0)
        isStarted = true;
    else
        System.out.print("Fuel is not enough for starting of engine");
}

/**
 * Method switches off the engine
 */
public void SwitchOffEngine()
{
    isStarted = false;
}

/**
 * Method refuel the engine
 * @param fuel
 */
public void Refuel(double fuel)
{
    this.fuel = fuel;
}

/**
 * Method returns the fuel of engine
 * @return The fuel value
 */
public double getFuel()
{

```

```

        return fuel;
    }

    /**
     * Method returns a status of engine
     * @return A status of engine
     */
    public boolean getEngineStatus()
    {
        return isStarted;
    }
}

/**
 * @author BAZHULIN SERHIY
 * Class <code>ControlPanel</code> implements control panel
 */
class ControlPanel
{
    enum Direction {RIGHT, LEFT, FORWARD}

    private int speed;
    private Direction direction;

    /**
     * Constructor
     */
    public ControlPanel()
    {
        speed = 0;
        direction=Direction.FORWARD;
    }
}

```

```
/**
 * Constructor
 * @param <code>speed</code>
 * @param <code>direction</code>
 */
public ControlPanel(int speed, Direction direction)
{
    this.speed = speed;
    this.direction = direction;
}
```

```
/**
 * Method set a speed for control panel
 * @param <code>speed</code>
 */
public void SpeedChange(int speed)
{
    this.speed=speed;
}
```

```
/**
 * Method returns the set speed on control panel
 * @return The speed value
 */
public int getSpeed()
{
    return speed;
}
```

```
/**
 * Method sets the right direction to control panel
```

```

    */
    public void TurnRight()
    {
        direction = Direction.RIGHT;
    }

    /**
     * Method sets the left direction to control panel
     */
    public void TurnLeft()
    {
        direction = Direction.LEFT;
    }

    /**
     * Method sets the forward direction to control panel
     */
    public void Forward()
    {
        direction = Direction.FORWARD;
    }

    /**
     * Method returns direction of control panel
     * @return Direction of control panel
     */
    public Direction getDirection()
    {
        return direction;
    }
}

```

```

/**
 * @author BAZHULIN SERHIY
 * Class <code>Door</code> implements door
 */
class Door
{
    private boolean isClosed;

    /**
     * Constructor
     * @throws FileNotFoundException
     */
    public Door() throws FileNotFoundException
    {
        isClosed = false;
    }

    /**
     * Constructor
     * @param isClosed
     * @throws FileNotFoundException
     */
    public Door(boolean isClosed) throws FileNotFoundException
    {
        this.isClosed = isClosed;
    }

    /**
     * Method opens door
     */
    public void Open()
    {

```

```

        isClosed = false;
    }

    /**
     * Method closes door
     */
    public void Close()
    {
        isClosed = true;
    }

    /**
     * Method returns status of door
     * @return door status
     */
    public boolean getDoorStatus()
    {
        return isClosed;
    }
}

```

### Файл ReusableSpaceship.java

```

/**
 *
 */
package KI307.Bazhulin.Lab3;

import java.io.FileNotFoundException;

/**
 * Interface <code>Mission</code> implements mission for spaceship
 */
interface Mission

```



```

{

    /**
     * Method taking off a spaceship
     */
    void TakeOff();

    /**
     * Method landing a spaceship
     */
    void Land();

    /**
     * Method returns flight status
     * @return flight status
     */
    boolean getStatus();
}

/**
 * Class <code>ReusableSpaceship</code> implements reusable spaceship
 * @author SERHIY BAZHULIN
 * @version 1.0
 */
public class ReusableSpaceship extends Spaceship implements Mission {

    private int flightsCount;
    private boolean inMission;

    /**
     * Constructor
     * @throws FileNotFoundException
     */

```

```

public ReusableSpaceship() throws FileNotFoundException {

    // TODO Auto-generated constructor stub

    super();

    this.flightsCount=0;

}


/**
 * Constructor
 * @param fuel
 * @param flightsCount
 * @throws FileNotFoundException
 */
public ReusableSpaceship(double fuel, int flightsCount) throws FileNotFoundException {

    // TODO Auto-generated constructor stub

    super(fuel);

    this.flightsCount=flightsCount;

}


/**
 * Method set amount of flights
 * @param count
 */
public void setFlightsCount(int count)
{

    this.flightsCount=count;

}


/**
 * Method get flights count
 * @return Flights count
 */
public int getFlightsCount()

```

```
{  
    return this.flightsCount;  
}
```

@Override

```
public void StartSpaceship() {  
    // TODO Auto-generated method stub  
    engine.StartEngine();  
    fout.print("Engine of reusable spaceship was started\n");  
}
```

@Override

```
public void SwitchOffSapceship() {  
    // TODO Auto-generated method stub  
    engine.SwitchOffEngine();  
    fout.print("Engine of reusable spaceship was switched off\n");  
}
```

@Override

```
public void RefuelSpaceship(double fuel) {  
    // TODO Auto-generated method stub  
    engine.Refuel(fuel);  
    fout.print("Engine of reusable spaceship was refuel: "+fuel+"\n");  
}
```

@Override


```
public void TakeOff() {  
    // TODO Auto-generated method stub  
    this.inMission = true;  
    fout.print("Reusable spaceship is taking off\n");  
}
```

```
@Override
public void Land() {
    // TODO Auto-generated method stub
    this.inMission = false;
    this.flightsCount++;
    fout.print("Reusable spaceship is landing\n");
}
```

```
@Override
public boolean getStatus() {
    // TODO Auto-generated method stub
    return this.inMission;
}
```

```
}
```

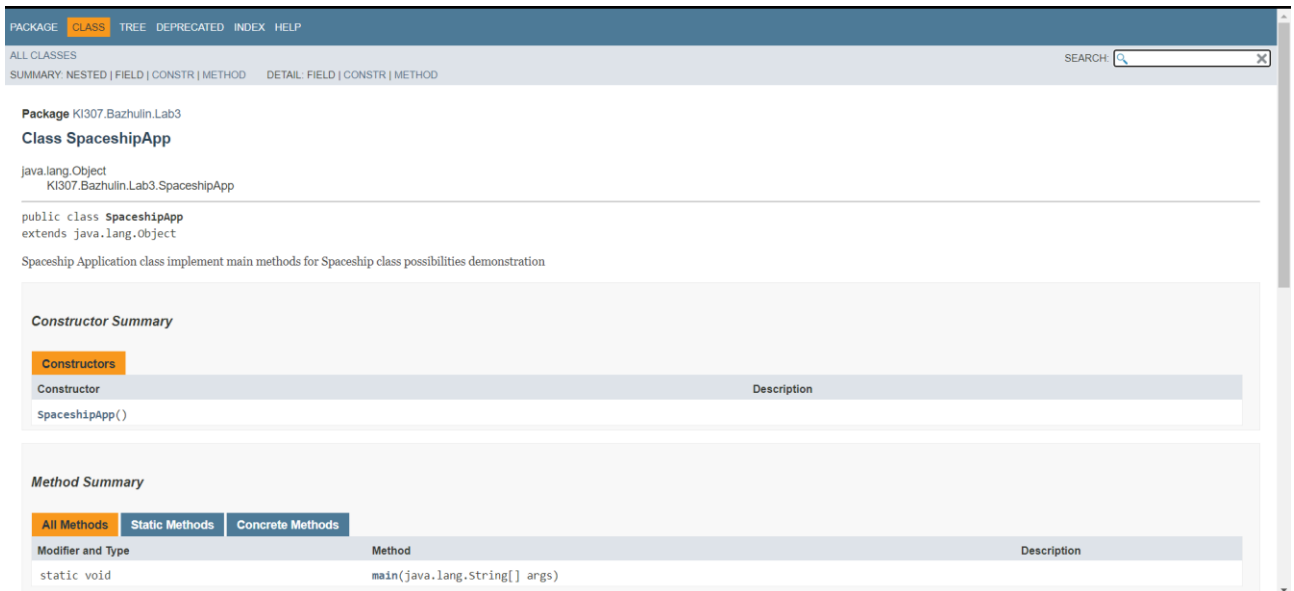
### Результат роботи програми

 Log - Notepad

File Edit Format View Help

```
Engine of reusable spaceship was refuel: 50.0
Door of spaceship was closed
Engine of reusable spaceship was started
Spaceship speed set 100
Reusable spaceship is taking off
Spaceship turned right
Reusable spaceship is landing
Engine of reusable spaceship was switched off
Door of spaceship was opened
```

## Фрагмент згенерованої документації:



## Відповідь на контрольні питання

### Синтаксис реалізації спадкування.

1. Синтаксис реалізації спадкування: ``class ChildClass extends ParentClass { ... }``.

### Що таке суперклас та підклас?

2. Суперклас - це клас, від якого інший клас (підклас) успадковує властивості і методи. Підклас - це клас, який успадковує властивості і методи від суперкласу.

### Як звернутися до членів суперкласу з підкласу?

3. До членів суперкласу з підкласу можна звертатися, використовуючи ключове слово ``super``.

### Коли використовується статичне зв'язування при виклику методу?

4. Статичне зв'язування використовується при виклику статичних методів або методів, які компілятор може визначити на етапі компіляції за типом посилання.

### Як відбувається динамічне зв'язування при виклику методу?

5. Динамічне зв'язування відбувається під час виконання програми, коли викликається метод на об'єкті, і вибір методу залежить від типу об'єкта на етапі виконання.

### Що таке абстрактний клас та як його реалізувати?

6. Абстрактний клас - це клас, який не може бути створений безпосередньо, і він містить абстрактні методи. Для його реалізації використовується ключове слово ``abstract``.

### Для чого використовується ключове слово `instanceof`?

7. Ключове слово ``instanceof`` використовується для перевірки, чи об'єкт є екземпляром певного класу або підкласу.

### **Як перевірити чи клас є підкласом іншого класу?**

8. Для перевірки, чи клас є підкласом іншого класу, можна використовувати ключове слово ``instanceof`` або порівняння типів об'єктів.

### **Що таке інтерфейс?**

9. Інтерфейс - це контракт, який визначає набір методів, які клас повинен реалізувати. Він не містить реалізації методів, тільки їхні сигнатури.

### **Як оголосити та застосувати інтерфейс?**

10. Для оголошення інтерфейсу використовується ключове слово ``interface``, і класи реалізують інтерфейс за допомогою ключового слова ``implements``.

**Висновок:** на цій лабораторній роботі, я ознайомився з спадкуванням та інтерфейсами в мові програмування java. Написав програму згідно до свого варіанту. Навчився оголошувати і використовувати інтерфейси.