Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ

Звіт

до лабораторної роботи №3

З дисципліни: «Кросплатформні засоби програмування»

На тему: «Спадкування та інтерфейси»

Варіант 5

Виконала:

ст. гр. КІ-305

Гринь С.М.

Прийняв:

Іванов Ю.С.

Львів 2023

Мета роботи: 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 №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.

Варіант 5: *Вантажна машина*

Код програми:

Class Automobile:

```
package KI305.Hryn.Lab3;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * Клас, що представляє автомобіль та зберігає інформацію про нього.
 */
public abstract class Automobile {
    //Поля класу
    private Company company;
    private Model model;
    private Price price;

    //Поле для запису протоколу
    private PrintWriter logWriter;

    /**
     * Пустий конструктор без аргументів.
```

```
     * Ініціалізує об'єкт Automobile та створює файл протоколу.
     */
    //Пустий конструктор без аргументів
    public Automobile() {
        try {
            logWriter = new PrintWriter(new
FileWriter("C:\\Users\\User\\IdeaProjects\\java_lab\\src\\KI305\\Hryn\\Lab3\\Automobile.txt"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //Конструктор зі всіма аргументами
    public Automobile(Company company, Model model, Price price) {
        this.company = company;
        this.model = model;
        this.price = price;
        try {
            logWriter = new PrintWriter(new
FileWriter("C:\\Users\\User\\IdeaProjects\\java_lab\\src\\KI305\\Hryn\\Lab3\\Automobile.txt"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //Абстрактні методи класу
    public abstract void moveObject();
    public abstract void stopObject();

    //Гетери і сетери для об'єктів класу
    public Company getCompany() {
        return company;
    }

    public void setCompany(Company company) {
        this.company = company;
```

```
}

public Model getModel() {
    return model;
}

public void setModel(Model model) {
    this.model = model;
}

public Price getPrice() {
    return price;
}

public void setPrice(Price price) {
    this.price = price;
}

/**
 * Метод для запису дій в файл протоколу.
 *
 * @param activity Дія, яку потрібно зареєструвати в протоколі.
 */

private void logActivity(String activity) {
    if (logWriter != null) {
        logWriter.println(activity);
        logWriter.flush();
    }
}

//Методи для роботи з автомобілем
public void startEngine() {
    System.out.println("Starting the car engine.");
    logActivity("We started the car engine.");//Запис у протокол
}
```

```
public void brake() {
    System.out.println("Turning on the brakes.");
    logActivity("We turned on the brakes.");//Запис у протокол
}

public void turnLeft() {
    System.out.println("Turning to the left.");
    logActivity("We turned to the left.");//Запис у протокол
}

public void turnRight() {
    System.out.println("Turning to the right.");
    logActivity("We turned to the right.");//Запис у протокол
}

public void checkEngineStatus() {
    System.out.println("Checking the condition of the engine.");
    logActivity("We checked the condition of the engine.");//Запис у протокол
}

public void turnOnHeadlights() {
    System.out.println("Turning on the headlights.");
    logActivity("We turned on the headlights.");//Запис у протокол
}

public void checkFuelLevel() {
    System.out.println("Checking the level of the fuel tank in the car.");
    logActivity("We checked the level of the fuel tank in the car.");//Запис у протокол
}

public void lockDoors() {
    System.out.println("Locking the doors.");
    logActivity("We locked the doors.");//Запис у протокол
}
```

```
    public void stopEngine() {
        System.out.println("Stopping the car engine.");
        logActivity("We stopped the car engine.");//Запис у протокол
        closeLogFile();//Закриваємо файл при завершенні роботи
    }

    // Метод для закриття файлу протоколу
    private void closeLogFile() {
        if (logWriter != null) {
            logWriter.close();
        }
    }
    //Метод toString()
    @Override
    public String toString() {
        return "Automobile: Company : " + company + ", Model : " + model + ", Price : " + price;
    }



}
```

Class Company:

```
package KI305.Hryn.Lab3;

/**
 * Клас, що представляє інформацію про компанію.
 */
public class Company {
    // Поля класу
    private String nameCompany; // Назва компанії
    private String infoCompany; // Інформація про компанію

    /**
     * Пустий конструктор без аргументів.
     * Ініціалізує об'єкт Company з пустими значеннями.
     */
    public Company(){

    }

    /**
     * Конструктор з усіма аргументами.
     * Ініціалізує об'єкт Company з вказаними параметрами.
     *
     * @param nameCompany Назва компанії.
```

```
     * @param infoCompany Інформація про компанію.
     */
    public Company(String nameCompany, String infoCompany){
        this.nameCompany = nameCompany;
        this.infoCompany = infoCompany;
    }

    /**
     * Перевизначений метод toString().
     *
     * @return Рядок, що представляє об'єкт Company.
     */
    @Override
    public String toString() {
        return "Company{" +
                "name='" + nameCompany + '\'' +
                ", info='" + infoCompany + '\'' +
                '}';
    }
}
```

Class ***Model***:

```
package KI305.Hryn.Lab3;

/**
 * Клас, що представляє модель автомобіля.
 */
public class Model {
    // Поле класу
    private String nameModel; // Назва моделі автомобіля

    /**
     * Пустий конструктор без аргументів.
     * Ініціалізує об'єкт Model з пустою назвою моделі.
     */
    public Model(){

    }

    /**
     * Конструктор з усіма аргументами.
     * Ініціалізує об'єкт Model з вказаною назвою моделі.
     *
     * @param nameModel Назва моделі автомобіля.
     */
    public Model(String nameModel){
        this.nameModel = nameModel;
    }

    /**
     * Перевизначений метод toString().
     *
     * @return Рядок, що представляє об'єкт Model.
```

```
     */
    @Override
    public String toString() {
        return "Model{" +
                "name:'" + nameModel + '\'' +
                '}';
    }
}
```

Class Price:

```
package KI305.Hryn.Lab3;


/**
 * Клас, що представляє ціну.
 */
public class Price {
    // Поле класу
    private int sum; // Сума ціни


    /**
     * Пустий конструктор без аргументів.
     * Ініціалізує об'єкт Price з нульовою сумою.
     */
    public Price() {


    }


    /**
     * Конструктор з усіма аргументами.
     * Ініціалізує об'єкт Price з вказаною сумою ціни.
     *
     * @param sum Сума ціни.
     */
    public Price(int sum) {
        this.sum = sum;
    }


    /**
     * Перевизначений метод toString().
```

```
     *
     * @return Рядок, що представляє об'єкт Price.
     */
    @Override
    public String toString() {
        return "Price{" + "sum=" + sum + "}";
    }
}
```

Class Truck:

```
package KI305.Hryn.Lab3;

/**
 * Клас, який представляє вантажний автомобіль.
 * Вантажний автомобіль є розширенням класу Automobile та реалізує інтерфейс ILights для керування світлами.
 */
public class Truck extends Automobile implements ILights {
    private boolean lights;
    /**
     * Конструктор без параметрів для вантажного автомобіля.
     * Створює вантажний автомобіль з вимкнутими світлами за замовчуванням.
     */
    public Truck(){
        super();
        lights = false;
    }

    public Truck(Company company, Model model, Price price) {
       super(company,model,price);

    }
    /**
     * Перевизначений метод для переміщення вантажного автомобіля.
     */
    @Override
    public void moveObject() {
```

```
        System.out.println("Object is moving");
    }
    /**
     * Перевизначений метод для зупинки вантажного автомобіля.
     */
    @Override
    public void stopObject() {
        System.out.println("Object stopped");
    }

    @Override
    public void setLights(boolean data) {

    }
    /**
     * Реалізація методу інтерфейсу ILights для виведення інформації про стан фар.
     */
    @Override
    public void lightInfo() {
        if (lights)
            System.out.println("Lights is on");
        else
            System.out.println("Lights is off");
    }

    public void setLight(boolean data) {
        this.lights = data;
    }
}
```

Interface ILights:

```
package KI305.Hryn.Lab3;

/**
 * Інтерфейс, що представляє функціональність керування світлами.
```

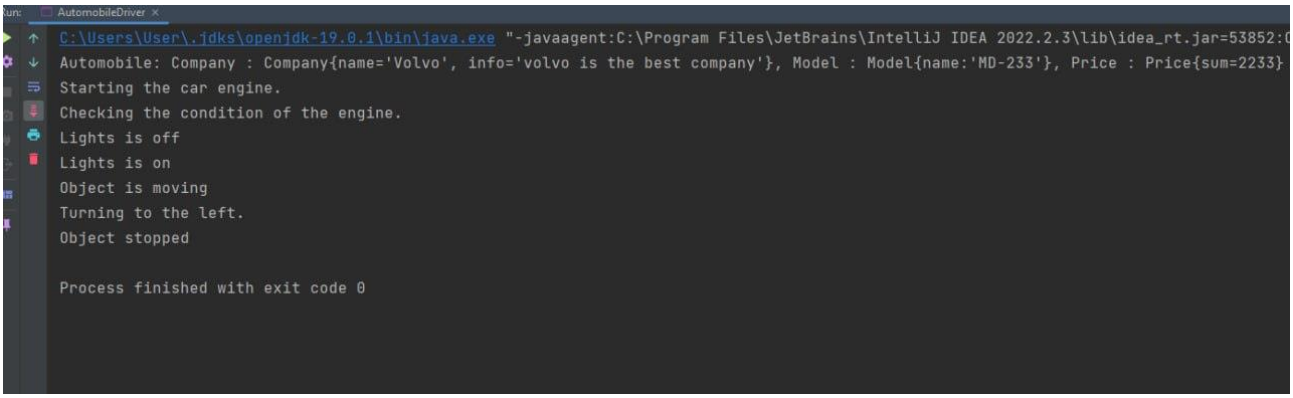
```
 * Реалізуючі класи повинні реалізовувати методи цього інтерфейсу для управління світлами і надавати інформацію про їх стан.
 */
public interface ILights {

    /**
     * Встановлює стан світла (увімкнено або вимкнено).
     *
     * @param data true, якщо світло має бути увімкнено, false, якщо світло має бути вимкнено.
     */
    void setLights(boolean data);

    /**
     * Виводить інформацію про поточний стан світла.
     */
    void lightInfo();
}
```

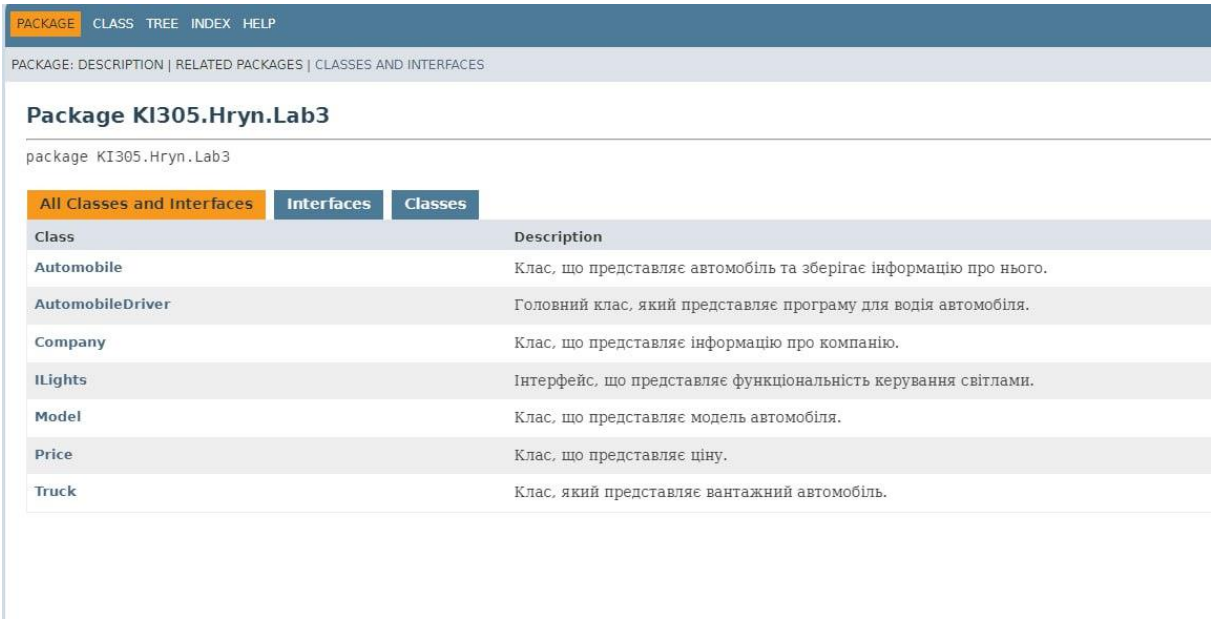
Class AutomobileDriver:

```
package KI305.Hryn.Lab3;

/**
 * Головний клас, який представляє програму для водія автомобіля.
 */
public class AutomobileDriver {
    /**
     * Головний метод програми.
     *
     * @param args Масив рядків аргументів командного рядка.
     */
    public static void main(String[] args) {
        // Ініціалізація полів

        Truck tr = new Truck(new Company("Volvo", "volvo is the best company"), new Model("MD-233"), new Price(2233));
        System.out.println(tr);
        tr.startEngine();
        tr.checkEngineStatus();
        tr.lightInfo();
        tr.setLight(true);
        tr.lightInfo();
        tr.moveObject();
        tr.turnLeft();
```

```
        tr.stopObject();

    }
}
```

Результат роботи програми:





PACKAGE CLASS TREE INDEX HELP

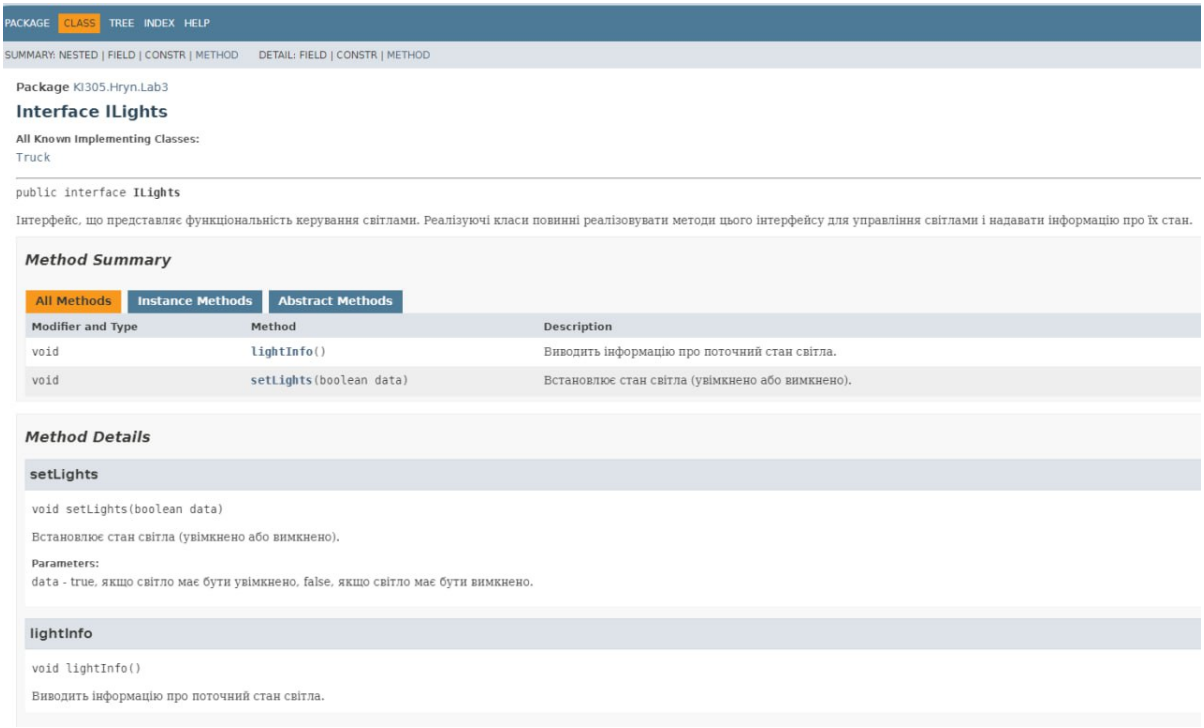
PACKAGE: DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES

Package KI305.Hryn.Lab3

package KI305.Hryn.Lab3

All Classes and Interfaces | Interfaces | Classes

Class	Description
Automobile	Клас, що представляє автомобіль та зберігає інформацію про нього.
AutomobileDriver	Головний клас, який представляє програму для водія автомобіля.
Company	Клас, що представляє інформацію про компанію.
ILights	Інтерфейс, що представляє функціональність керування світлами.
Model	Клас, що представляє модель автомобіля.
Price	Клас, що представляє ціну.
Truck	Клас, який представляє вантажний автомобіль.



PACKAGE CLASS TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package KI305.Hryn.Lab3

Interface ILights

All Known Implementing Classes:
Truck

public interface **ILights**

Інтерфейс, що представляє функціональність керування світлами. Реалізуючі класи повинні реалізовувати методи цього інтерфейсу для управління світлами і надавати інформацію про їх стан.

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method	Description
void	lightInfo()	Виводить інформацію про поточний стан світла.
void	setLights(boolean data)	Встановлює стан світла (увімкнено або вимкнено).

Method Details

setLights

void setLights(boolean data)

Встановлює стан світла (увімкнено або вимкнено).

Parameters:
data - true, якщо світло має бути увімкнено, false, якщо світло має бути вимкнено.

lightInfo

void lightInfo()

Виводить інформацію про поточний стан світла.

PACKAGE CLASS TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package KI305.Hryn.Lab3

Class Truck

java.lang.Object
KI305.Hryn.Lab3.Automobile
KI305.Hryn.Lab3.Truck

All Implemented Interfaces:
ILights

```
public class Truck
extends Automobile
implements ILights
```

Клас, який представляє вантажний автомобіль. Вантажний автомобіль є розширенням класу Automobile та реалізує інтерфейс ILights для керування світлами.

Constructor Summary

Constructors

Constructor	Description
`Truck()`	Конструктор без параметрів для вантажного автомобіля.
`Truck(Company company, Model model, Price price)`	

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
void	`lightInfo()`	Реалізація методу інтерфейсу ILights для виведення інформації про стан фар.
void	`moveObject()`	Перевизначений метод для переміщення вантажного автомобіля.
void	`setLight(boolean data)`	
void	`setLights(boolean data)`	Встановлює стан світла (увімкнено або вимкнено).
void	`stopObject()`	Перевизначений метод для зупинки вантажного автомобіля.

PACKAGE CLASS TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH Search

Methods inherited from class KI305.Hryn.Lab3.Automobile

brake, checkEngineStatus, checkFuelLevel, getCompany, getModel, getPrice, lockDoors, setCompany, setModel, setPrice, startEngine, stopEngine, toString, turnLeft, turnOnHeadlights, turnRight

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Details

Truck

```
public Truck()
```

Конструктор без параметрів для вантажного автомобіля. Створює вантажний автомобіль з вимкнутими світлами за замовчуванням.

Truck

```
public Truck(Company company,
             Model model,
             Price price)
```

Method Details

moveObject

```
public void moveObject()
```

Перевизначений метод для переміщення вантажного автомобіля.

Specified by:
moveObject in class Automobile

stopObject

stopObject

```
public void stopObject()
```

Перевизначений метод для зупинки вантажного автомобіля.

Specified by:
`stopObject` in class `Automobile`

setLights

```
public void setLights(boolean data)
```

Description copied from interface: `ILights`
Встановлює стан світла (увімкнено або вимкнено).

Specified by:
`setLights` in interface `ILights`

Parameters:
`data` - true, якщо світло має бути увімкнено, false, якщо світло має бути вимкнено.

lightInfo

```
public void lightInfo()
```

Реалізація методу інтерфейсу ILights для виведення інформації про стан фар.

Specified by:
`lightInfo` in interface `ILights`

setLight

```
public void setLight(boolean data)
```


Package KI305.Hryn.Lab3

Class AutomobileDriver

java.lang.Object
KI305.Hryn.Lab3.AutomobileDriver

```
public class AutomobileDriver
extends Object
```

Головний клас, який представляє програму для водія автомобіля.

Constructor Summary

Constructors

Constructor	Description
`AutomobileDriver()`	

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method	Description
static void	`main(String[] args)`	Головний метод програми.

Methods inherited from class java.lang.Object

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Constructor Details

AutomobileDriver

```
public AutomobileDriver()
```

PACKAGE CLASS TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package KI305.Hryn.Lab3

Class Automobile

java.lang.Object
KI305.Hryn.Lab3.Automobile

Direct Known Subclasses:

Truck

```
public abstract class Automobile
extends Object
```

Клас, що представляє автомобіль та зберігає інформацію про нього.

Constructor Summary

Constructors

Constructor	Description
`Automobile()`	Пустий конструктор без аргументів.
`Automobile(Company company, Model model, Price price)`	

Method Summary

All Methods | **Instance Methods** | **Abstract Methods** | **Concrete Methods**

Modifier and Type	Method	Description
void	`brake()`	
void	`checkEngineStatus()`	
void	`checkFuelLevel()`	
`Company`	`getCompany()`	
`Model`	`getModel()`	
`Price`	`getPrice()`	
void	`lockDoors()`	
abstract void	`moveObject()`	

PACKAGE CLASS TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package KI305.Hryn.Lab3

Class Company

java.lang.Object
KI305.Hryn.Lab3.Company

```
public class Company
extends Object
```

Клас, що представляє інформацію про компанію.

Constructor Summary

Constructors

Constructor	Description
`Company()`	Пустий конструктор без аргументів.
`Company(String nameCompany, String infoCompany)`	Конструктор з усіма аргументами.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`String`	`toString()`	Перевизначений метод toString().

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Details

Company

```
public Company()
```

Package KI305.Hryn.Lab3

Class Model

java.lang.Object
KI305.Hryn.Lab3.Model

public class **Model**
extends Object

Клас, що представляє модель автомобіля.

Constructor Summary

Constructors

Constructor	Description
`Model()`	Пустий конструктор без аргументів.
`Model(String nameModel)`	Конструктор з усіма аргументами.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
`String`	`toString()`	Перевизначений метод toString().

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Details

Model

```
public Model()
```


Package KI305.Hryn.Lab3

Class Price

java.lang.Object
KI305.Hryn.Lab3.Price

public class Inheritance Tree
extends Object

Клас, що представляє ціну.

Constructor Summary

Constructors

Constructor	Description
`Price()`	Пустий конструктор без аргументів.
`Price(int sum)`	Конструктор з усіма аргументами.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
`String`	`toString()`	Перевизначений метод toString().

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Details

Price

```
public Price()
```

Контрольні запитання

1. *Синтаксис реалізації спадкування.*

Спадкування реалізується шляхом вказування ключового слова `class` після якого вказується назва *підкласу*, ключове слово `extends` та назва *суперкласу*, що розширюється у новому підкласі. Синтаксис реалізації спадкування:

```
class Підклас extends Суперклас
{
      Додаткові поля і методи
}
```

2. *Що таке суперклас та підклас?*

В термінах мови Java базовий клас найчастіше називається *суперкласом*, а похідний клас – *підкласом*. Дана термінологія запозичена з теорії множин, де підмножина міститься у супермножині.

3. *Як звернутися до членів суперкласу з підкласу?*

Перевизначення у підкласах елементів суперкласів (полів або методів) призводить до їх приховування новими елементами. Бувають ситуації, коли у методах підкласу необхідно звернутися до цих прихованих елементів суперкласів. У цій ситуації слід використати ключове слово **`super`**, яке вказує, що елемент до якого йде звернення, розташовується у суперкласі, а не у підкласі. Синтаксис звертання до елементів суперкласу:

```
super.назваМетоду([параметри]); // виклик методу суперкласу
super.назваПоля                 // звертання до поля суперкласу
```

4. *Коли використовується статичне зв'язування при виклику методу?*

Якщо метод є приватним, статичним, фінальним або конструктором, то для нього застосовується механізм статичного зв'язування. Механізм статичного зв'язування передбачає визначення методу, який необхідно викликати, на етапі компіляції.

5. *Як відбувається динамічне зв'язування при виклику методу?*

Якщо для виклику методу використовується динамічне зв'язування, то віртуальна машина повинна викликати версію методу, що відповідає фактичному типу об'єкту на який посилається об'єктна змінна. Оскільки на пошук необхідного методу потрібно багато часу, то віртуальна машина заздалегідь створює для кожного класу таблицю методів, в якій перелічуються сигнатури 5 всіх методів і фактичні методи, що підлягають виклику. При виклику методу віртуальна машина просто переглядає таблицю методів. Якщо відбувається виклик методу з суперкласу за допомогою ключового слова super, то при виклику методу переглядається таблиця методів суперкласу неявного параметру.

6. *Що таке абстрактний клас та як його реалізувати?*

Абстрактні класи призначені бути основою для розробки ієрархій класів та не дозволяють створювати об'єкти свого класу. Вони реалізуються за допомогою ключового слова abstract. На відміну від звичайних класів абстрактні класи можуть містити абстрактні методи (а можуть і не містити). Абстрактні методи – це методи, що 6 оголошені з використанням ключового слова abstract і не місять тіла. Розширюючи абстрактний клас можна залишити деякі або всі методи невизначеними. При цьому підклас автоматично стане абстрактним. Перевизначення у підкласі усіх абстрактних методів призведе до того, що підклас не буде абстрактним, що дозволить створювати на його основі об'єкти класу. Синтаксис оголошення абстрактного класу наведено в пункті «Класи та об'єкти». Синтаксис оголошення абстрактного методу:

```
[СпецифікаторДоступу] abstract Тип назваМетоду([параметри]);
```

7. Для чого використовується ключове слово instanceof?

Щоби перевірити коректність приведення супертипу до підтипу слід використовувати оператор instanceof.

8. Як перевірити чи клас є підкласом іншого класу?

Приведення типів у Java відбувається вказуванням у круглих дужках перед змінною, яку необхідно привести до іншого типу, типу до якого її необхідно привести. Синтаксис приведення змінної до іншого типу:

```
(новийТип) змінна
```

У Java усі об'єктні змінні є типізовані. Механізми наслідування і поліморфізму дозволяють створювати нові типи (класи та інтерфейси) на базі вже існуючих та присвоювати об'єкти цих типів посиланням на об'єкти супертипу. В цьому випадку об'єкти підтипів мають ті самі елементи, що й об'єкти супертипу, тож таке висхідне приведення типів є безпечним і здійснюється компілятором автоматично. Проте присвоєння посиланню на об'єкт підтипу об'єкту супертипу не завжди є коректним, тому таке приведення вимагає явного приведення типів. При такому приведенні типів можливі дві ситуації:

- якщо посилання на об'єкт супертипу реально посилається на об'єкт підтипу, то приведення посилання на об'єкт супертипу до типу підтипу є коректним;
- якщо посилання на об'єкт супертипу посилається на об'єкт супертипу, то приведення посилання на об'єкт супертипу до типу підтипу викличе виключну ситуацію `ClassCastExeption`.

Наявність бодай одної такої виключної ситуації призводить до аварійного завершення програми. Щоб уникнути цього слід перед приведенням типів використати оператор `instanceof`, який повертає `true`, якщо посилання посилається на об'єкт фактичний тип якого є не вищим в ієрархії типів, ніж вказаний у операторі `instanceof`, і `false` у протилежному випадку. Синтаксис оператора `instanceof`:

```
посилання instanceof Ім'яТипу
```

9. Що таке інтерфейс?

Інтерфейси вказують що повинен робити клас не вказуючи як саме він це повинен робити. Інтерфейси покликані компенсувати відсутність множинного спадкування у мові Java та гарантують визначення у класах оголошених у собі прототипів методів.

10.Як оголосити та застосувати інтерфейс?

```
[public] interface НазваІнтерфейсу
{
      Прототипи методів та оголошення констант інтерфейсу
}
```

Інтерфейси можуть містити прототипи методів, які мають визначатися у класі, що відповідає цьому інтерфейсу, і константи, які автоматично успадковуються класом, що реалізує цей інтерфейс. Всі методи інтерфейсу вважаються загальнодоступними, тому оголошувати методи як `public` у інтерфейсі нема необхідності. Всі поля, що оголошені у інтерфейсі автоматично вважаються такими, що оголошені як `public static final`, тому додавати ці модифікатори самостійно необхідності також нема. Інтерфейси можуть успадковувати інші інтерфейси, утворюючи таким чином ієрархії інтерфейсів. Синтаксис реалізації спадкування у інтерфейсів співпадає з синтаксисом реалізації спадкування у класах.

Оскільки інтерфейс не є класом, то створити його об'єкт за допомогою оператора `new` неможливо. Проте можна оголосити посилання на інтерфейсний тип та присвоїти цьому посиланню об'єкт, що реалізує цей інтерфейс.

На відміну від спадкування клас може реалізувати кілька інтерфейсів. Ця особливість і компенсує відсутність множинного спадкування у мові Java.

Щоб клас реалізував інтерфейс необхідно:

1. Оголосити за допомогою ключового слова `implements`, що клас реалізує інтерфейс. Якщо клас реалізує кілька інтерфейсів, то вони перелічуються через кому після ключового слова `implements`.
2. Визначити у класі усі методи, що вказані у інтерфейсі.

Висновок: я ознайомилася із спадкуванням та інтерфейсами у мові Java.