Національний університет «Львівська Політехніка»

Інститут комп’ютерних технологій, автоматики та метрології

Кафедра електронних обчислювальних машин



Звіт

Про виконання лабораторної роботи №3

З дисципліни «Кросплатформлені засоби програмування»

**Виконав:**

студент групи КІ-305

Ключко Д.С.

**Перевірив:**

Доцент кафедри ЕОМ

Іванов Ю. С.

Львів – 2023

**Тема:** Спадкування та інтерфейси.

**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 №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.

**Завдання згідно варіанту:**

**Варіант: 12**

****

**Виконання:**

**Код програми:**

**Water.java:**

**package** lab2Klyuchko305;

**import** java.io.\*;

**public** **class** Water {

**private** **static** Square *Squareofwater*;

**private** EcologyWater Ecologysytuation;

**private** **double** temperature;

**private** Fullinformation Information;

**private** PrintWriter fout;

/\*\*

\* The Water class represents properties and functionalities related to a body of water.

\*/

**public** Water() **throws** FileNotFoundException{

// Class attributes including water square, ecological situation, temperature, information, and file output

*Squareofwater* = **new** Square();

Ecologysytuation = **new** EcologyWater();

temperature =0;

Information = **new** Fullinformation();

fout = **new** PrintWriter ("Log.txt");

}

/\*\*

\* Constructor with a temperature parameter initializing water properties with given temperature.

\* **@param** xtemperature the temperature of the water

\* **@throws** FileNotFoundException if Log.txt file is not found

\*/

**public** Water(**double** xtemperature) **throws** FileNotFoundException{

*Squareofwater* = **new** Square();

Ecologysytuation = **new** EcologyWater();

temperature = xtemperature;

Information = **new** Fullinformation();

fout = **new** PrintWriter ("Log.txt");

}

/\*\*

\* Retrieves the calculated square of the water body.

\* **@return** the calculated square of the water body

\*/

**public** **double** GetCalcSquare()

{

**return** *Squareofwater*.calcSquare();

}

/\*\*

\* Determines and displays the current depth of the water body based on temperature conditions.

\*/

**public** **void** CurrentDepth()

{

**double** curentdepth;

**if**(temperature >= 25 && temperature <=45)

{

curentdepth = *Squareofwater*.getDepth()-0.75;

}

**else** {

curentdepth = *Squareofwater*.getDepth();

}

System.***out***.print("Поточна глибина: ");

System.***out***.print(curentdepth);

System.***out***.println(" m ");

fout.print("Поточна глибина: ");

fout.print(curentdepth);

fout.println(" m");

}

/\*\*

\* Sets the width and length parameters for the water body's square.

\* **@param** widthSquareofwater the width of the water body

\* **@param** lengthSquareofwater the length of the water body

\*/

**public** **void** setParametersforSquare(**double** widthSquareofwater, **double** lengthSquareofwater){

*Squareofwater*.setLengts(lengthSquareofwater);

*Squareofwater*.setWidth(widthSquareofwater);

}

/\*\*

\* Retrieves the width of the water body.

\* **@return** the width of the water body

\*/

**public** **double** getWidthforWater()

{

**return** *Squareofwater*.getWidth();

}

/\*\*

\* Retrieves the length of the water body.

\* **@return** the length of the water body

\*/

**public** **double** getLengthforWater()

{

**return** *Squareofwater*.getLength();

}

/\*\*

\* Sets parameters for the ecological situation such as color and transparency of the water.

\* **@param** colorwaterEcologysytuation the color of the water

\* **@param** transparencyEcologysytuation the transparency level of the water

\*/

**public** **void** setParametersforEcologysituation(String colorwaterEcologysytuation, **int** transparencyEcologysytuation)

{

Ecologysytuation.setColorwater(colorwaterEcologysytuation);

Ecologysytuation.setTransparency(transparencyEcologysytuation);

}

/\*\*

\* Displays the current ecological situation of the water body.

\*/

**public** **void** CurentEcologysituation()

{

Ecologysytuation.Ecologysituation();

}

/\*\*

\* Displays the maximum depth of the water body.

\*/

**public** **void** Depth()

{

fout.print("Максимальна глибина: ");

fout.print(*Squareofwater*.getDepth());

fout.println(" m");

System.***out***.print("Максимальна глибина: ");

System.***out***.print (*Squareofwater*.getDepth());

System.***out***.println(" m ");

}

/\*\*

\* Sets parameters for the geographical information of the water body.

\* **@param** countryInformation the country of the water body

\* **@param** regionInformation the region of the water body

\* **@param** typeInformation the type of the water body

\*/

**public** **void** setParametersforInformation(String countryInformation,String regionInformation, String typeInformation)

{

Information.setCountry(countryInformation);

Information.setRegions(regionInformation);

Information.setType(typeInformation);

}

/\*\*

\* Displays the geographical information of the water body.

\*/

**public** **void** getInformation()

{

System.***out***.print("Країна: ");

System.***out***.println(Information.getCountry());

System.***out***.print("Регіон: ");

System.***out***.println(Information.getRegion());

System.***out***.print("Тип водойми: ");

System.***out***.println(Information.getType());

fout.print ("Країна: ");

fout.println(Information.getRegion());

fout.print("Регіон: ");

fout.println(Information.getCountry());

fout.print("Тип водойми: ");

fout.println(Information.getType());

fout.flush();

}

//class for calcSquare

/\*\*

\* The Square class represents the dimensions and calculations related to the square of a water body.

\*/

**class** Square{

**private** **double** width;

**private** **double** length;

/\*\*

\* Default constructor initializing width and length to zero.

\*/

**public** Square(){

width = 0.0;

length =0.0;

}

/\*\*

\* Constructor initializing width and length with provided values.

\* **@param** widthSquareofwater the width of the water body

\* **@param** lengthSquareofwater the length of the water body

\*/

**public** Square(**double** widthSquareofwater, **double** lengthSquareofwater) {

width = widthSquareofwater;

length = lengthSquareofwater;

}

// Methods get

/\*\*

\* Gets the width of the water body.

\* **@return** the width of the water body

\*/

**public** **double** getWidth()

{

**return** width;

}

/\*\*

\* Gets the length of the water body.

\* **@return** the length of the water body

\*/

**public** **double** getLength()

{

**return** length;

}

//Methods set

/\*\*

\* Sets the width and length of another Square object.

\* **@param** obj the Square object to set width and length

\*/

**public** **void** getSquare(Square obj) {

obj.width = width;

obj.length = length;

}

/\*\*

\* Sets the width of the water body.

\* **@param** widthSquareofwater the width to set

\*/

**public** **void** setWidth(**double** widthSquareofwater)

{

width = widthSquareofwater;

}

/\*\*

\* Sets the length of the water body.

\* **@param** lengthSquareofwater the length to set

\*/

**public** **void** setLengts(**double** lengthSquareofwater)

{

length = lengthSquareofwater;

}

/\*\*

\* Calculates the depth of the water body based on width and length.

\* **@return** the depth of the water body

\*/

**public** **double** getDepth() {

**double** depth;

depth = 4\*(width)/(length);

**return** depth;

}

/\*\*

\* Calculates the square of the water body and displays it.

\* **@return** the calculated square of the water body

\*/

**public** **double** calcSquare() {

**double** square;

square = *Squareofwater*.getLength() \* *Squareofwater*.getWidth();

fout.print("Площа водойми: ");

fout.print(square);

fout.println(" m^2");

System.***out***.print("Площа водойми: ");

System.***out***.print(square);

System.***out***.println(" m^2");

**return** square;

}

}

/\*\*

\* The EcologyWater class manages ecological parameters and checks for a water body.

\*/

**class** EcologyWater{

**private** String colorwater;

**private** **int** transparency;

/\*\*

\* Default constructor initializing default color and transparency values.

\*/

**public** EcologyWater(){

colorwater = "Бірюзовий";

transparency = 70;

}

/\*\*

\* Constructor initializing color and transparency with provided values.

\* **@param** colorwaterEcologysytuation the color of the water

\* **@param** transparencyEcologysytuation the transparency level of the water

\*/

**public** EcologyWater (String colorwaterEcologysytuation, **int** transparencyEcologysytuation) {

colorwater = colorwaterEcologysytuation;

transparency = transparencyEcologysytuation;

}

/\*\*

\* Retrieves the color of the water.

\* **@return** the color of the water

\*/

**public** String getColorwater()

{

**return** colorwater;

}

/\*\*

\* Retrieves the transparency level of the water.

\* **@return** the transparency level of the water

\*/

**public** **int** getTransparency()

{

**return** transparency;

}

/\*\*

\* Sets the color and transparency of another EcologyWater object.

\* **@param** obj the EcologyWater object to set color and transparency

\*/

**public** **void** getEcologyParameters(EcologyWater obj) {

obj.colorwater = colorwater;

obj.transparency = transparency;

}

/\*\*

\* Sets the color of the water.

\* **@param** colorwaterEcologysytuation the color to set

\*/

**public** **void** setColorwater(String colorwaterEcologysytuation)

{

colorwater = colorwaterEcologysytuation;

}

/\*\*

\* Sets the transparency level of the water.

\* **@param** transparencyEcologysytuation the transparency level to set

\*/

**public** **void** setTransparency(**int** transparencyEcologysytuation)

{

transparency = transparencyEcologysytuation;

}

/\*\*

\* Checks the ecological situation of the water body based on color and transparency.

\* Prints information about water cleanliness based on defined criteria.

\*/

**public** **void** Ecologysituation()

{

**if**(colorwater=="Зелений"||colorwater=="Жовтий"||colorwater=="Коричневий")

{

System.***out***.println("Колір води є незадовільним! В даній водоймі брудна вода!!!");

fout.println("Колір води є незадовільним! В даній водоймі брудна вода!!!");

}

**else**

{

System.***out***.println("За результатами перевірки кольору води, водойма є чистою!");

fout.println("За результатами перевірки кольору води, водойма є чистою!");

}

**if**(transparency<60)

{

System.***out***.println("Вода не є прозорою! В даній водоймі брудна вода!!!");

fout.println("Вода не є прозорою! В даній водоймі брудна вода!!!");

}

**else**

{

System.***out***.println("Вода прозора,отже водойма є чистою!");

fout.println("Вода прозора,отже водойма є чистою!");

}

}

}

/\*\*

\* The Fullinformation class manages geographical information about a water body.

\*/

**class** Fullinformation{

**private** String country;

**private** String region;

**private** String type;

/\*\*

\* Default constructor initializing default country, region, and type values.

\*/

**public** Fullinformation(){

country = "default";

region = "default";

type = "default";

}

/\*\*

\* Constructor initializing country, region, and type with provided values.

\* **@param** countryInformation the country of the water body

\* **@param** regionInformation the region of the water body

\* **@param** typeInformation the type of the water body

\*/

**public** Fullinformation (String countryInformation,String regionInformation, String typeInformation) {

country = countryInformation;

region =regionInformation;

type = typeInformation;

}

/\*\*

\* Retrieves the country of the water body.

\* **@return** the country of the water body

\*/

**public** String getCountry()

{

**return** country;

}

/\*\*

\* Retrieves the region of the water body.

\* **@return** the region of the water body

\*/

**public** String getRegion()

{

**return** region;

}

/\*\*

\* Retrieves the type of the water body.

\* **@return** the type of the water body

\*/

**public** String getType()

{

**return** type;

}

/\*\*

\* Sets the country of the water body.

\* **@param** countryInformation the country to set

\*/

**public** **void** setCountry(String countryInformation)

{

country = countryInformation;

}

/\*\*

\* Sets the region of the water body.

\* **@param** regionInformation the region to set

\*/

**public** **void** setRegions(String regionInformation)

{

region =regionInformation;

}

/\*\*

\* Sets the type of the water body.

\* **@param** typeInformation the type to set

\*/

**public** **void** setType(String typeInformation)

{

type =typeInformation;

}

/\*\*

\* Sets the country, region, and type of another Fullinformation object.

\* **@param** obj the Fullinformation object to set country, region, and type

\*/

**public** **void** getEcologyParameters(Fullinformation obj) {

obj.country = country;

obj.region = region;

obj.type = type;

}

}

}

**Sea.java:**

**package** lab3Klyuchko305;

**import** java.io.FileNotFoundException;

//Interface defining the behavior for shipping balance

**interface** Balance {

**void** Shippingbalance();

}

//Interface defining the behavior for ship growth

**interface** Growth {

**int** Shipgrowth();

}

/\*\*

\* Sea class extends Water and implements Balance and Growth interfaces.

\* Represents attributes and functionalities related to the sea.

\*/

**public** **class** Sea **extends** Water **implements** Balance, Growth {

**private** **int** shipsaresailing;

**private** **int** shipsareinports;

**private** **int** data;

**private** **int** shippinbalance;

/\*\*

\* Constructor initializing Sea object with specified parameters.

\* **@param** Xshipsaresailing number of ships sailing

\* **@param** Xshipsareinports number of ships in ports

\* **@param** Xdata data parameter

\* **@throws** FileNotFoundException if file not found

\*/

**public** Sea(**int** Xshipsaresailing, **int** Xshipsareinports, **int** Xdata) **throws** FileNotFoundException {

shipsaresailing = Xshipsaresailing;

shipsareinports = Xshipsareinports;

data = Xdata;

shippinbalance = 0;

}

/\*\*

\* Calculates shipping balance based on ships in ports and sailing.

\*/

**public** **void** Shippingbalance() {

shippinbalance = shipsareinports - shipsaresailing;

}

/\*\*

\* Gets the calculated shipping balance.

\* **@return** the calculated shipping balance

\*/

**public** **int** getShippingbalance() {

**return** shippinbalance;

}

/\*\*

\* Calculates ship growth based on ships in ports, sailing, and data.

\* **@return** the calculated ship growth

\*/

**public** **int** Shipgrowth() {

**return** (shipsareinports - shipsaresailing) / data;

}

/\*\*

\* Prints a line.

\* **@param** line the line to print

\*/

**public** **void** Print(**int** line) {

System.***out***.println(line);

}

/\*\*

\* Writes an integer line to the file and closes it if the line is 0.

\* **@param** line the line to write

\*/

**public** **void** WritetoFile(**int** line) {

**if** (line == 0) {

*fout*.close();

}

*fout*.println(line);

*fout*.flush();

}

/\*\*

\* Gets the number of ships sailing.

\* **@return** the number of ships sailing

\*/

**public** **int** getShiparesaling()

{

**return** shipsaresailing;

}

/\*\*

\* Writes a string line to the file and closes it if the line is null.

\* **@param** line the line to write

\*/

**public** **void** WritetoFile(String line) {

**if** (line == **null**) {

*fout*.close();

}

*fout*.print(line);

*fout*.flush();

}

}

**WaterApp.java:**

**package** lab3Klyuchko305;

**import** java.io.\*;

/\*\*

\* Class App realize program to lab work 3

\*

\* **@author** Klyuchko Dmytro

\* **@version** 1.1

\* **@since** version 1.0

\*/

**public** **class** WaterApp {

/\*\*

\* **@param** args

\* **@throws** FileNotFoundException

\*/

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

Sea lake = **new** Sea(22,35,2);

lake.setParametersforSquare(10.5, 12.34);

lake.GetCalcSquare();

lake.setParametersforEcologysituation("Голубий", 70);

lake.CurentEcologysituation();

lake.Depth();

lake.CurrentDepth();

lake.setParametersforInformation("Україна", "Закарпаття", "Озеро");

lake.getInformation();

lake.Shippingbalance();

System.***out***.print("Сальдо суднолавства: ");

lake.Print(lake.getShippingbalance());

System.***out***.print("Приріст кораблів: ");

lake.Print(lake.Shipgrowth());

lake.WritetoFile("Сальдо суднолавства: ");

lake.WritetoFile(lake.getShippingbalance());

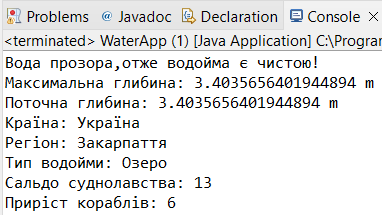
lake.WritetoFile("Приріст кораблів: ");

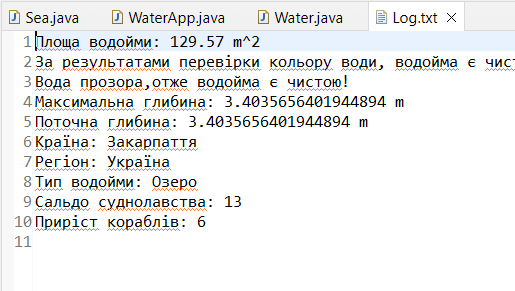
lake.WritetoFile(lake.Shipgrowth());

}

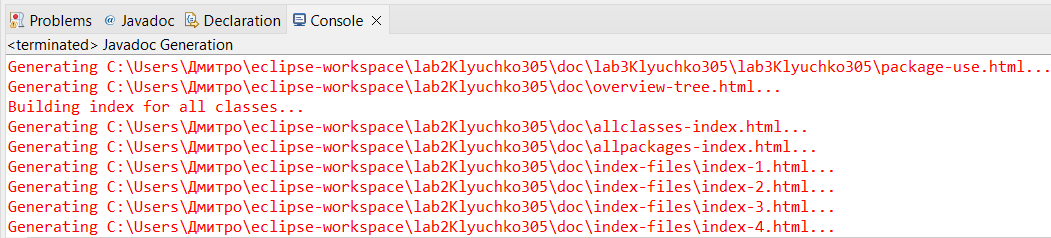
}

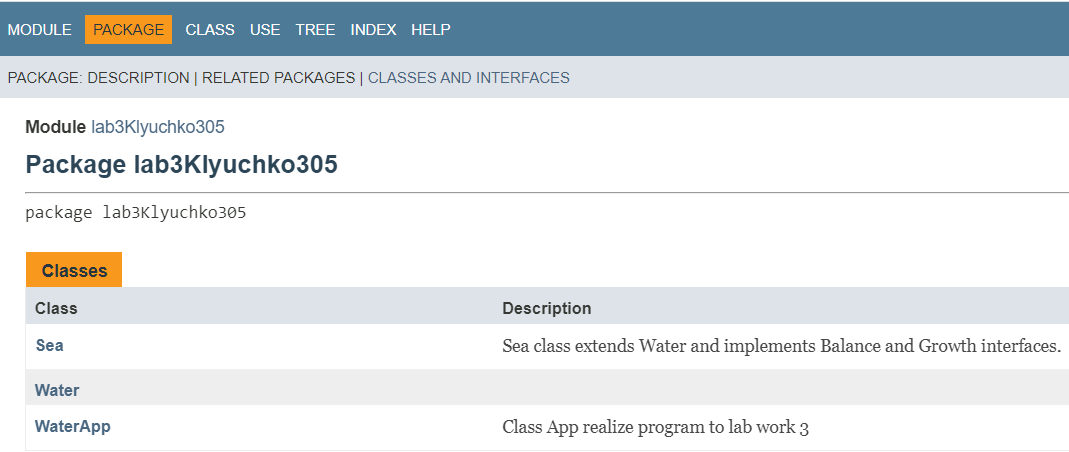
**Результат роботи програми у консолі та файлі:**

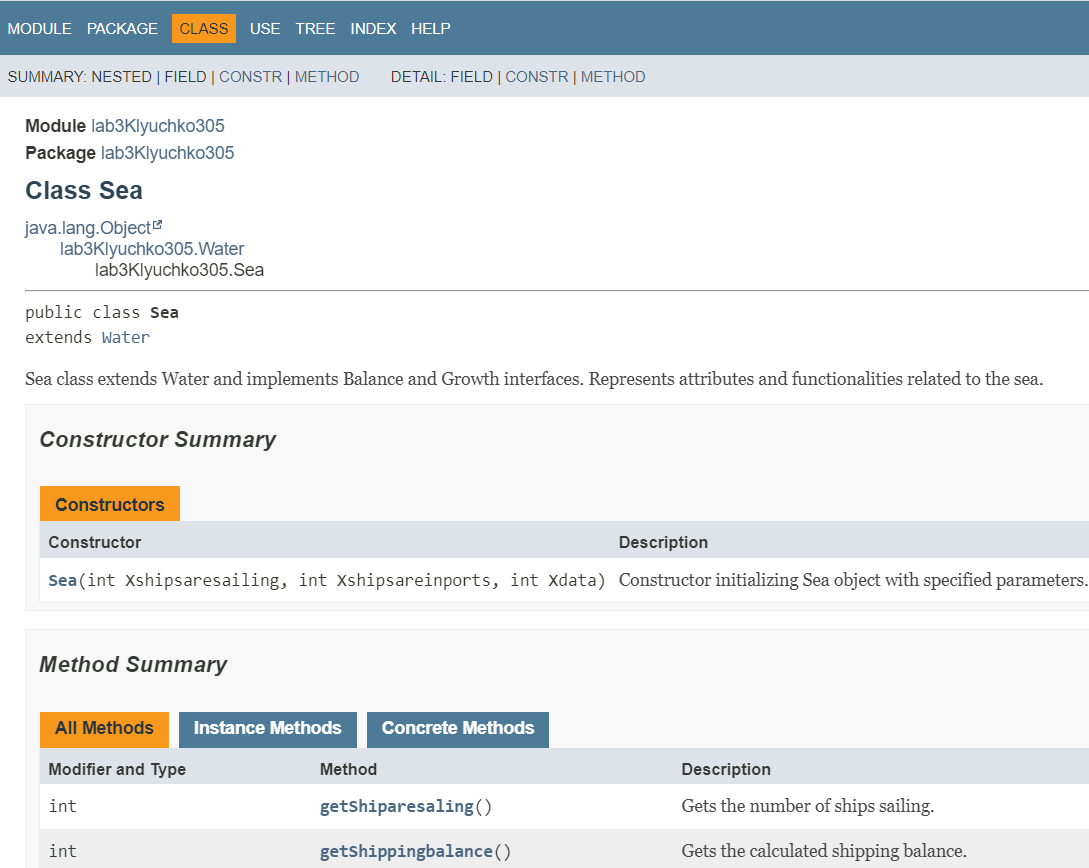
****

****

**Фрагмент генерації документації:**

****

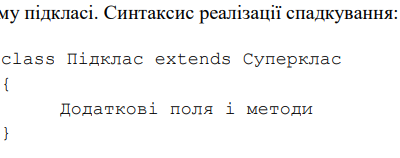
****

****

**КОНТРОЛЬНІ ПИТАННЯ**

1. Синтаксис реалізації спадкування.

**Відповідь:**

****

1. Що таке суперклас та підклас?**Відповідь:**

В термінах мови Java базовий клас найчастіше називається суперкласом, а похідний клас – підкласом

1. Як звернутися до членів суперкласу з підкласу?

**Відповідь:**

Для доступу до членів суперкласу з підкласу в Java використовуються ключові слова super та ідентифікатори класів.

1. Коли використовується статичне зв’язування при виклику методу?

**Відповідь:**

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.

1. Як відбувається динамічне зв’язування при виклику методу?

**Відповідь:**

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.

1. Що таке абстрактний клас та як його реалізувати?

**Відповідь:**

Абстрактні класи призначені бути основою для розробки ієрархій класів та не

дозволяють створювати об’єкти свого класу. Вони реалізуються за допомогою ключового

слова abstract.

Щоб створити абстрактний клас у Java, використовується ключове слово abstract:

abstract class AbstractClass {

// Абстрактний метод без реалізації (тіло відсутнє)

public abstract void abstractMethod();

}

1. Для чого використовується ключове слово instanceof?

**Відповідь:**

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.

1. Як перевірити чи клас є підкласом іншого класу?

**Відповідь:**

У Java використовується ключове слово instanceof для перевірки, чи є об'єкт екземпляром певного класу або чи є певний клас підкласом іншого класу.

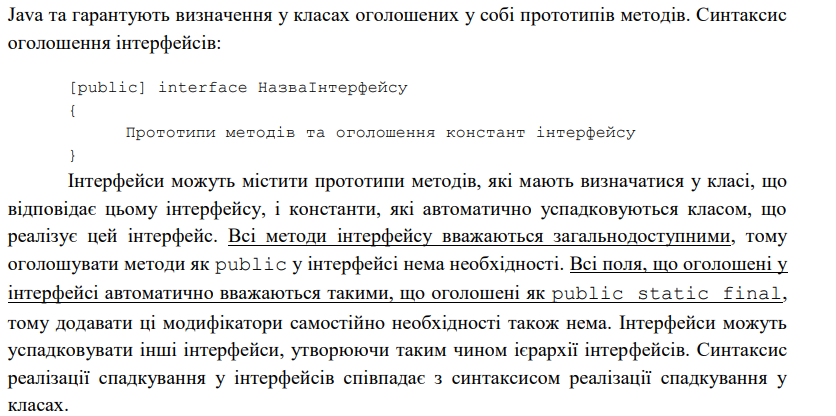
1. Що таке інтерфейс?

**Відповідь:**

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.

1. Як оголосити та застосувати інтерфейс?

**Відповідь:**

****

**Висновок:** на цій лабораторній роботі я ознайомився з спадкуванням та інтерфейсами у мові Java. Перетворив попередній клас водойма у абстрактний і реалізував деякі його методи у новоствореному класі море. Також створив інтерфейс «Приріст кораблів» і реалізував його метод в класі море також.