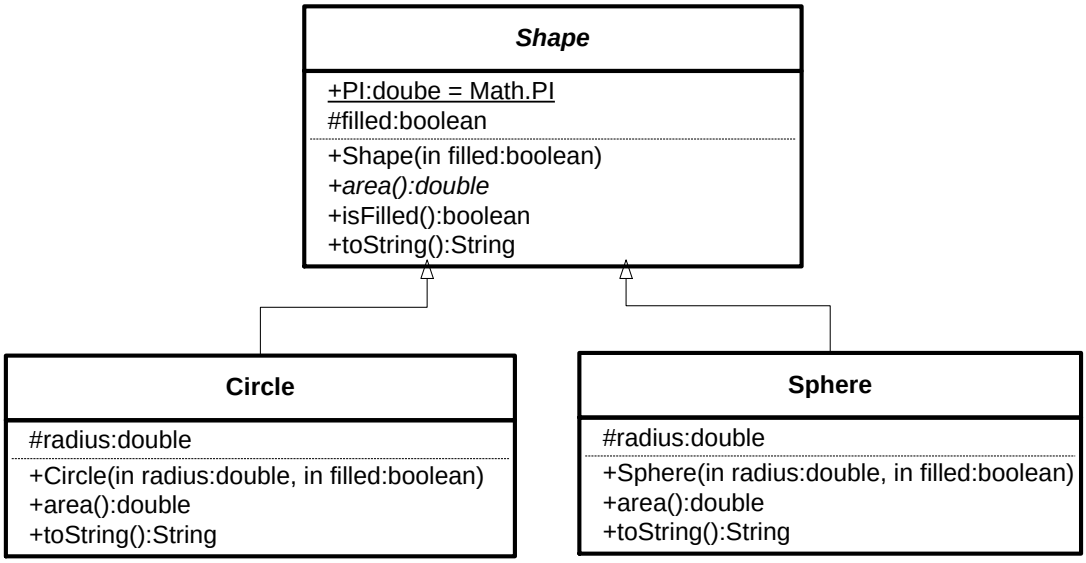
Упражнение №2

Абстракни класове. Интерфейси.
Полиморфизъм чрез абстрактни класове и интерфейси.

I. Създайте нов проект **Lab2a** със следната йерархия от класове:



1. Създайте абстрактен клас **Shape** (**Фигура**).
   а) декларирайте константа **PI** с **public** достъп от тип **double** и поле запълване **filled** с **protected** достъп от тип **boolean**.
   б) добавете конструктор с един параметър за инициализиране на полето;
   в) напишете абстрактен метод **area()** за изчисляване на лице на фигура;
   г) напишете метод за достъп **isFilled()**, който връща дали фигурата е запълнена;
   д) предефинирайте метода **toString()** на класа **Object**, който връща символното представяне на типа.

```
package lab2a;
public abstract class Shape {                  // Клас Фигура
    public static final double PI = Math.PI; // константа π
    protected boolean filled;                  // запълване
    public Shape (boolean filled) {            // Конструктор с един параметър
        this.filled = filled;
    }
    public abstract double area();           // Абстрактен метод за лице
    public boolean isFilled() {              // Връща дали фигурата е запълнена
        return filled;
    }
    @Override
    public String toString() {                 // Връща символното представяне
        return (filled ? "запълнена фигура" : "незапълнена фигура");
    }
}
```

2. Създайте клас **Circle** (**Окръжност**) като наследник на абстрактния клас **Shape** (**Фигура**).
   а) декларирайте поле радиус **radius** с **protected** достъп от тип **double**;

б) добавете конструктор с два параметъра за инициализиране на полетата; абстрактният клас **Shape** не може да създаде инстанция, но конструкторът на **Shape** е с **public** модификатор за достъп и неговите наследници могат да извикат конструктора чрез **super()**;

а) предефинирайте метода **area()** на класа **Shape**, който връща лицето на окръжност;

б) предефинирайте метода **toString()** на класа **Shape**, който връща символното представяне на типа.

```
package lab2a;
public class Circle extends Shape {          // Клас Окръжност
    protected double radius;                 // радиус
    // Конструктор с два параметъра
    public Circle(double radius, boolean filled) {
        super(filled);
        this.radius = radius;
    }
    @Override
    public double area() {                   // Връща лице на окръжност
        return PI*radius*radius;
    }
    @Override
    public String toString() {               // Връща символното представяне
        return "Окръжност - " + super.toString() + " с радиус = " + radius;
    }
}
```

2. Създайте клас **Sphere** (**Сфера**) като наследник на абстрактния клас **Shape** (**Фигура**).

а) декларирайте поле радиус **radius** с **protected** достъп от тип **double**;

б) добавете конструктор с два параметъра за инициализиране на полетата;

в) предефинирайте метода **area()** на класа **Shape**, който връща лицето на сфера;

г) предефинирайте метода **toString()** на класа **Shape**, който връща символното представяне на типа.

```
package lab2a;
public class Sphere extends Shape {          // Клас Сфера
    protected double radius;                 // радиус
    // Конструктор с два параметъра
    public Sphere(double radius, boolean filled) {
        super(filled);
        this.radius = radius;
    }
    @Override
    public double area() {                   // Връща лице на сфера
        return 4*PI*radius*radius;
    }
    @Override
    public String toString() {               // Връща символното представяне
        return "Сфера - " + super.toString() + " с радиус = " + radius;
    }
}
```

II. Използвайте класа **Lab2a** за тестване на класовете и тяхната йерархия.

1. В метода **`main()`** декларирайте:
   а) динамичен масив **`list`** от тип **`ArrayList<Shape>`**, който ще съдържа фигури и импортирайте класа **`ArrayList`** от пакета **`java.util`**;
   б) дефинирайте променлива **`shape`** от абстрактния супер клас **`Shape`**, за да реализирате полиморфно обръщение към класовете **`Circle`** и **`Sphere`**;
   в) създайте инстанции на класовете **`Circle`** и **`Sphere`**, като свържете референцията **`shape`** на абстрактния супер клас **`Shape`** с всеки един създаден обект и добавете всеки обект към динамичния масив **`list`**;
   г) разпечатайте елементите на динамичния масив с данни **`list`** и лицето на всеки елемент.

```
package lab2a;
import java.util.ArrayList;
public class Lab2a {                                          // Тестов клас
    public static void main(String[] args) {
        ArrayList<Shape> list = new ArrayList<>();          // динамичен масив
        Shape shape;
        shape = new Circle(7.0, true);
        list.add(shape);
        shape = new Sphere(10.0, false);
        list.add(shape);
        shape = new Circle(5.0, false);
        list.add(shape);
        shape = new Sphere(3.0, true);
        list.add(shape);
        System.out.println("Списък");
        for (Shape element : list) {
            System.out.format("%s и лице = %.3f\n", element, element.area());
        }
}
```

2. Изпълнете приложението:

   Резултатът е:

```
Списък
Окръжност - запълнена фигура с радиус = 7.0 и лице = 153.938
Сфера - незапълнена фигура с радиус = 10.0 и лице = 1256.637
Окръжност - незапълнена фигура с радиус = 5.0 и лице = 78.540
Сфера - запълнена фигура с радиус = 3.0 и лице = 113.097
```

   По време на изпълнение се определят версиите на извиканите методи **`toString()`** и **`area()`** в зависимост от типа на действителния обект **`element`** (**`късно свързване`**).

3. Към класа **`Lab2a`** добавете статичен метод **`sum()`** с входен параметър **`list`** от тип **`ArrayList<Shape>`** с данни за фигури, който връща като резултат сумата от лицата на фигурите от тип **`double`**.

```
public static double sum(ArrayList<Shape> list) {
    double s = 0.0;
    for(Shape element : list)
        s += element.area();          // полиморфно обръщение към area()
    return s;
}
```

   Реализира се полиморфно обръщение към метода **`area()`** в зависимост от типа на **`element`** (окръжност или сфера) по време на изпълнение.

4. В метода **`main()`** извикайте статичния метод **`sum()`**, за да изчислите сумата от лицата на фигурите.

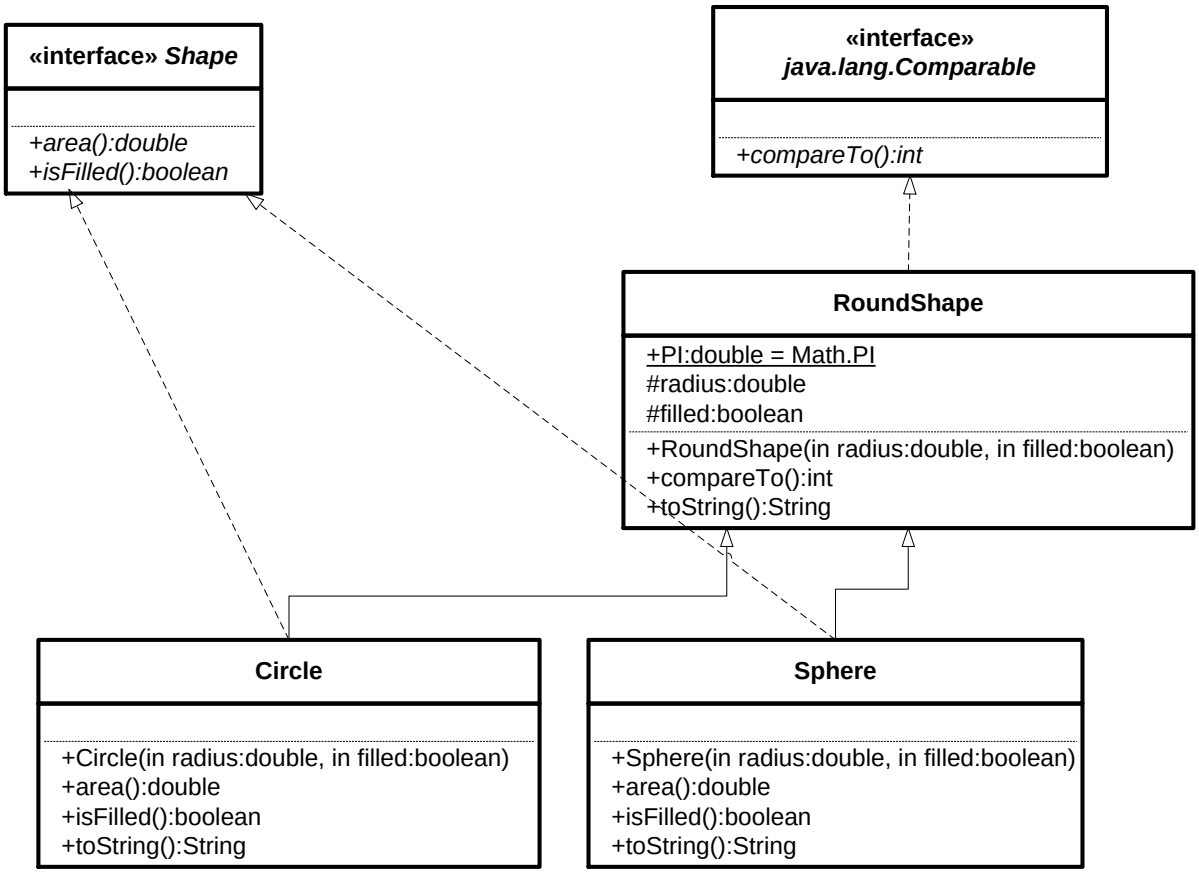
```
double s = sum(list);
System.out.format("Сумата от лицата на фигурите е %.3f\n", s);
```

5. Изпълнете приложението.

Резултатът е:

```
Сумата от лицата на фигурите е 1602.212
```

III. Създайте нов проект **`Lab2b`** със следната йерархия от класове и интерфейси:



1. Създайте интрефейс **`Shape`** (**`Фигура`**), който дефинира следния контракт: всяка фигура съдържа метод **`area()`** (**`Лице`**) и метод **`isFilled()`** (**`Запълнена фигура`**).

```
package lab2b;
public interface Shape {                    // Интерфейс Фигура
    public double area();                   // Връща лице на фигура
    public boolean isFilled();              // Връща дали фигурата е запълнена
}
```

2. Създайте супер клас **`RoundShape`** (**`Заоблена фигура`**), който реализира интерфейса **`Comparable`**.
   а) декларирайте константа **`PI`** с **`public`** достъп от тип **`double`** и полета: радиус **`radius`** с **`protected`** достъп от тип **`double`** и запълване **`filled`** с **`protected`** достъп от тип **`boolean`**;
   а) добавете конструктор с два параметъра за инициализиране на полетата;
   б) предефинирайте метода **`compareTo()`** на интерфейса **`Comparable`**, който сравнява заобленитe фигури по поле радиус; връща **`-1`**, **`0`** или **`1`** в зависимост от това дали текущият обект е по-малък, равен или по-голям от сравнявания обект;
   в) предефинирайте метода **`toString()`** на класа **`Object`**, който връща символното представяне на типа.

```
package lab2b;
public class RoundShape implements Comparable { // Клас Заоблена фигура
    public static final double PI = Math.PI;    // константа π
    protected double radius;                    // радиус
    protected boolean filled;                   // запълване
    // Конструктор с два параметъра
    public RoundShape (double radius, boolean filled) {
        this.radius = radius;
        this.filled = filled;
    }
    // Сравнява текущия обект и obj по поле радиус;
    @Override
    public int compareTo (Object obj) {
        if (obj == null) throw new NullPointerException("Нулев обект");
        // Проверява дали obj е от потребителския клас
        if (obj instanceof RoundShape) {
            // Принудително преобразува obj до потребителския клас
            RoundShape rshape = (RoundShape)obj;
            return Double.compare(radius, rshape.radius);
        }
        // Хвърля изключение, ако типът на obj не е от потребителския тип
        throw new ClassCastException ("Обектът не е от тип RoundShape");
    }
    @Override
    public String toString() {              // Връща символното представяне
        return (filled ? "запълнена фигура" : "незапълнена фигура") +
               " с радиус = " + radius;
    }
}
```

Сравняването на обекти от тип **`double`** се осъществява чрез статичния метод **`compare()`** на класа **`Double`**. Методът **`compareTo()`** може да хвърли изключението **`NullPointerException`**, ако сравняваният обект има стойност **`null`** и изключението **`ClassCastException`**, ако сравняваният обект не може да се преобразува до потребителския тип **`RoundShape`**. Обработка на изключения се разглежда в Упражнение №3.

3. Създайте клас **`Circle`** (**`Окръжност`**), който наследява супер класа **`RoundShape`** (**`Заоблена фигура`**) и реализира интерфейса **`Shape`** (**`Фигура`**).
   а) добавете конструктор с два параметъра за инициализиране на наследените полета;
   б) предефинирайте метода **`area()`** на интерфейса **`Shape`**, който връща лицето на окръжност;
   в) предефинирайте метода **`isFilled()`** на интерфейса **`Shape`**, който връща дали окръжността е запълнена;
   г) предефинирайте метода **`toString()`** на класа **`RoundShape`**, който връща символното представяне на типа.

```
package lab2b;
public class Circle extends RoundShape implements Shape { // Клас Окръжност
    // Конструктор с два парметъра
    public Circle(double radius, boolean filled) {
        super(radius, filled);
    }
```

```
    @Override
    public double area() {              // Връща лице на окръжност
        return PI*radius*radius;
    }
    @Override
    public boolean isFilled() {         // Връща дали окръжността е запълнена
        return filled;
    }
    @Override
    public String toString() {          // Връща символното представяне
        return "Окръжност - " + super.toString();
    }
}
```

4. Създайте клас **Sphere** (**Сфера**), който наследява супер класа **RoundShape** (**Заоблена фигура**) и реализира интерфейса **Shape** (**Фигура**).
   а) добавете конструктор с два параметъра за инициализиране на наследените полета;
   б) предефинирайте метода **area()** на интерфейса **Shape**, който връща лицето на сфера;
   в) предефинирайте метода **isFilled()** на интерфейса **Shape**, който връща дали сферата е запълнена;
   г) предефинирайте метода **toString()** на класа **RoundShape**, който връща символното представяне на типа.

```
package lab2b;
public class Sphere extends RoundShape implements Shape { // Клас Сфера
    // Конструктор с два парметъра
    public Sphere(double radius, boolean filled) {
        super(radius, filled);
    }
    @Override
    public double area() {              // Връща лице на сфера
        return 4*PI*radius*radius;
    }
    @Override
    public boolean isFilled() {         // Връща дали сферата е запълнена
        return filled;
    }
    @Override
    public String toString() {          // Връща символното представяне
        return "Сфера - " + super.toString();
    }
}
```

IV. Използвайте класа **Lab2b** за тестване на йерархията на класовете и интерфейсите.

1. В метода **main()** декларирайте:
   а) динамичен масив **list** от тип **ArrayList<Shape>**, който ще съдържа фигури и импортирайте пакета **java.util**;
   б) демонстрирайте полиморфизъм чрез интерфейси – дефинирайте променлива **shape** от типа на интерфейса **Shape**;

в) създайте инстанции на класовете **Circle** и **Sphere**, като свържете референцията **shape** на интерфейса **Shape** с всеки един създаден обект и добавете всеки обект към динамичния масив **list**;

г) разпечатайте елементите на динамичния масив с данни **list** и лицето на всеки елемент.

```
package lab2b;
import java.util.*;
public class Lab2b {
    public static void main(String[] args) {
        ArrayList<Shape> list = new ArrayList<>();
        Shape shape;
        shape = new Circle(7.0, true);
        list.add(shape);
        shape = new Sphere(10.0, false);
        list.add(shape);
        shape = new Circle(5.0, false);
        list.add(shape);
        shape = new Sphere(3.0, true);
        list.add(shape);
        System.out.println("Списък");
        for (Shape element : list)
            System.out.format("%s и лице = %.3f\n", element, element.area());
    }
}
```

Изпълнителната система извиква подходящите реализирани методи **area()** и **isFilled()**, дефиниращи контракт в интерфейса **Shape**, според действителния обект **element**.

2. Изпълнете приложението.

   Резултатът е:

```
Списък
Окръжност - запълнена фигура с радиус = 7.0 и лице = 153.938
Сфера - незапълнена фигура с радиус = 10.0 и лице = 1256.637
Окръжност - незапълнена фигура с радиус = 5.0 и лице = 78.540
Сфера - запълнена фигура с радиус = 3.0 и лице = 113.097
```

3. В метода **main()** сортирайте динамичния масив **list** според радиуса на фигурите. Използвайте статичния метод **sort()** на класа **Collections**, като типът на елементите на масива трябва да реализира интерфейса **Comparable**.

   а) създайте динамичен масив **data** от тип **ArrayList<RoundShape>** – типът **RoundShape** реализира интерфейса **Comparable**, класовете **Circle** и **Sphere** наследяват супер класа **RoundShape** и следователно наследяват метода му **compareTo()**;

   б) прехвърлете елементите от динамичния масив **list** в **data**;

   в) извикайте метода **Collections.sort()** за масива **data**;

   г) разпечатайте елементите на сортирания динамичен масив **data**.

```
ArrayList<RoundShape> data = new ArrayList<>();
for (Shape element : list)
    data.add((RoundShape)element);
Collections.sort(data);
System.out.println("Сортиран списък");
for (RoundShape element : data)
    System.out.println(element);
```

4. Изпълнете приложението.

   Резултатът е:

   ```
   Сортиран списък
   Сфера - запълнена фигура с радиус = 3.0
   Окръжност - незапълнена фигура с радиус = 5.0
   Окръжност - запълнена фигура с радиус = 7.0
   Сфера - незапълнена фигура с радиус = 10.0
   ```

V. Към последната йерархия добавете клас **`Rectangle`** (**`Правоъгълник`**), който реализира интерфейса **`Shape`**, има три полета: **`страна`**, **`височина`** и **`запълнен`**, и реализира методите на интерфейса. Създайте инстанции на класа **`Rectangle`**, като свържете референцията **`shape`** на интерфейса **`Shape`** със създадените обекти и добавете създадените обекти към динамичния масив **`list`**. Разпечатайте елементите на динамичния масив **`list`**. Могат ли да бъдат прехвърлени елементите от динамичния масив **`list`** в **`data`**, за да бъдат сортирани елементите на динамичния масив от тип **`Rectangle`**?