Tutorial - Week 05 5COSC019W – Object Oriented Programming – Java

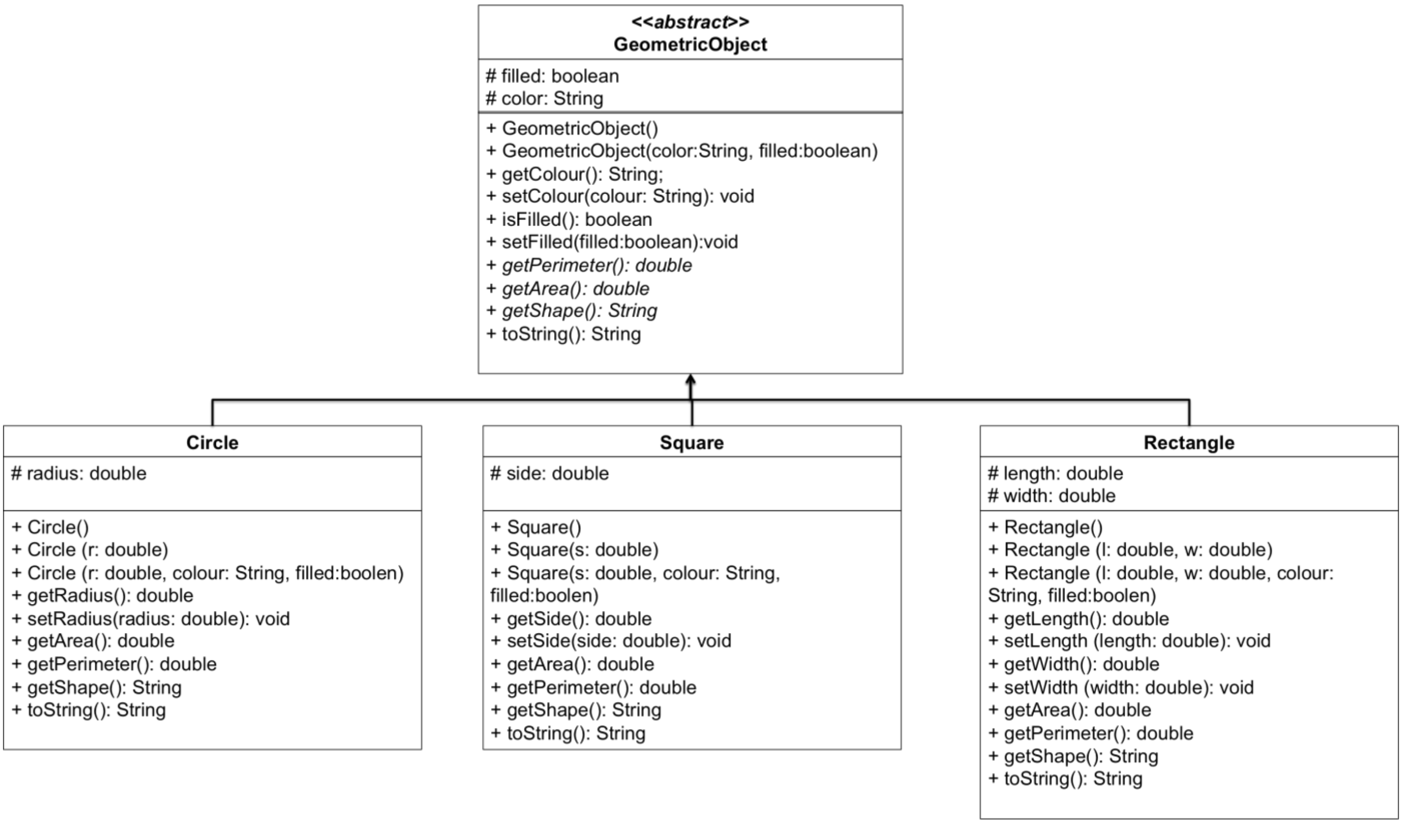
Abstract Class and Interfaces

24October-25October

Abstract Class and its subclasses

- A class that is declared using "abstract" keyword is known as abstract class.
- It may or may not include abstract methods which means in abstract class you can have concrete methods (methods with implementation) as well along with abstract methods (without an implementation, without braces, and followed by a semicolon).
- An abstract class can not be instantiated (you are not allowed to create object of Abstract class).

1) Implement the superclass `GeometricObject` and its subclasses `Circle`, `Rectangle` and `Square`, as shown in the class diagram.



GeometricObject Class

GeometricShape class is defined as abstract class, which contains:

- Two protected instance variables `colour:String` and `filled:boolean`. The protected variable can be accessed by its sublclasses and classes in the same package. They are defined with a '#' sign in the class diagram.
- Getter and setter methods for the instance variables and the method `toString()`.
- Three Abstract methods:
  - *getArea()*: return the area of the geometric object
  - *getPerimeter()*: return the perimeter of the geometric object
  - *getShape()*: return a String that indicate the shape of the object (e.g. "Circle" if the object is circle, "Rectangle" if the object is a rectangle" and "Square" if it is a square.

  These methods are showed in italics in the class diagram since are abstract! The subclasses shall override these abstract methods providing the proper implementation.

```
public abstract class GeometricObject {
    protected String colour;
    protected boolean filled;

    public GeometricObject(){…}

    public GeometricObject(String colour, boolean filled){…}

    public void setColour(String colour){…}

    public String getColour(){…}

    public void setFilled(boolean filled){…}

    public boolean getFilled(){…}

    public String toString {…}

    public abstract double getArea();

    public abstract double getPerimeter();          Abstract methods with no implementation

    public abstract String getShape();

}
```

Circle Class

`Circle` class is a subclass of `GeometricObject`. It inheriths all the instance variables and the methods of the class `GeometricObject`. `Circle` has also an instance variable `radius`. In this class you will override the methods getShape, getArea() and getPerimeter(), calculating the area and the perimeter of a circle.

Remember that the area of a circle is = PI * radius * radius, and the perimeter (circunference) is = 2 * PI * radius.

2) Implement the class Cicle following the specification in the class diagram.

```
public class Circle extends GeometricObject {

    protected double radius;

    public Circle(){…}

    public Circle(double radius){…}

    public Circle(double radius, String colour, boolean filled){…}

    public void setRadius(double radius){…}

    public double getRadius(){…}

    public  double getArea(){…}

    public double getPerimeter(){…}

    public String getShape(){…}

}
```

Rectangle Class

`Rectangle` class is a subclass of `GeometricObject`. It inheriths all the instance variables and the methods of the class `GeometricObject`. `Rectangle` has also two other instance variables: length and width. In this class you will override the methods getshape(), getArea() and getPerimeter(), calculating the area and the perimeter of a rectangle, as you did for the class `Circle`.

Remember that the area of a rectangle is = length * width, and the perimeter is = 2 * length + 2 * width.

3) Implement the class Rectangle following the specifications of the class diagram.

Square Class

Square class is a subclass of GeometricObject. It inheriths all the instance variables and the methods of the class GeometricObject. Square has also an other instance variable: side. In this class you will override the methods getShape(), getArea() and getPerimeter() as you did for the class Circle, calculating the area and the perimeter of a square.

Remember that the area of a square is = side * side, and the perimeter is = 4 * side.

4) Implement the class Square following the specifications of the class diagram.

Interface

Interface looks like class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body). Also, the variables declared in an interface are public, static & final by default.

5) Write a class GeometricShapeCollection that implements the interface ShapeCollection. GeometricShapeCollection has a list of maximum 10 GeometricObjects. The class will display a menu from where the user can decide if adding a shape (a circle, or a square or a rectangle) to this list, or printing the list of shapes with the relative area and perimeter.

Hint: From the package that contains the geometric objects, add a new file selecting "Java Interface". Call the interface "ShapeCollection".

```
public interface ShapeCollection {

    public abstract void addShape(GeometricObject shape);
    public abstract void printShapeList();
    public abstract boolean runMenu();
}
```

As you can see the implementation in the above methods is missing. We need to implement them in the class GeometricShapeCollection. So, let's add a class called GeometricShapeCollection to our project. This class implements the interface so we need to declare the class as follow:

```
public class GeometricShapeCollection implements ShapeCollection {
…
}
```

The class GeometriShapeCollection contains:

- two instance variables: an ArrayList of GeometricObject and an integer that rapresents the maximum number of GeometricObject that we want to keep in the list:

  ```
  private ArrayList<GeometricObject> shapeList;
  private int numObject;
  ```

- a constructor that takes as parameter the number of the objects in the list:

  ```
  public GeometricShapeCollection(int listLength){
       this.numObject = listLength;
       shapeList = new ArrayList<GeometricObject>();
    }
  ```

- the implementation of the abstract methods that are declared in the interface:

    - **public void addShape(GeometricObject shape)**

```
public  void addShape(GeometricObject shape){

        // check if there are spaces and add the shape to the list
        if (shapeList.size() < numObject){
            shapeList.add(shape);
        }
        else{
            System.out.println("No more space in the list");
        }
    }
```

    - **public void printShepeList()**

Verify if it is a Circle

```
public void printShapeList(){

        for(int i=0; i < shapeList.size(); i++){

     if(shapeList.get(i).getShape().equals("Circle")){
                System.out.println("Shape = Circle, Area = " +
shapeList.get(i).getArea() + ", Perimeter = " + shapeList.get(i).getPerimeter());
            }

         // continue the implementation of this method and write here code to verify
         if it is a rectangle or a square and print the area and the perimeter

        }
    }
```

    - **public boolean runMenu()**

Console menu from where the user can select one of the three options

```
    public boolean runMenu(){

        boolean exit = false;

        System.out.println("To Add a new shape press 1");
        System.out.println("To print the list of the shapes press 2");
        System.out.println("To exit press 3");

        Scanner s = new Scanner (System.in);
        int choise = s.nextInt();

        switch(choise){
            case 1:
                System.out.println("Press 1 if you want to add a Circle");
                System.out.println("Press 2 if you want to add a Rectangle");
                System.out.println("Press 3 if you want to add a Square");

                int choise2 = s.nextInt();
                s.nextLine();

                System.out.println("Enter the color of your shape");
                String colour = s.nextLine();

                System.out.println("Is it filled? (y/n)");
                String isFilled = s.nextLine();
                boolean filled = false;

                if (isFilled.equals("y"))
```

The user can select if add one of the three shapes

These properties (colour and filled) are common to all shapes!

```
                filled = true;
            else if (isFilled.equals("n"))
                filled = false;
            else
                System.out.println("Please enter y or n, if the
                       shape is filled or not respectively");

            switch (choise2){
                case 1:
                    // it is a circle
                    System.out.println("Insert the radius");
                    int radius = s.nextInt();
                    Circle c = new Circle(radius, colour, filled);
                    this.addShape(c);
                    break;

                case 2:
                    // write here the code if the rectangle is selected

                case 3:
                    // write here the code if the square is selected
            }
        break;

        case 2:

           this.printShapeList();
            break;

        case 3:
            exit = true;
            break;

    }

    return exit;

}
```

If the circle is selected

The other two initial options: print the list or exit

- the implementation of the main where the menu is launched:

```
public static void main(String[] args){

        // create a parking
        ShapeCollection sys = new GeometricShapeCollection(10);
        boolean exit = false;

        while (!exit){
            exit = sys.runMenu();
        }
    }
```

The menu is displayed until the user decides to exit

6) Draw an UML class diagram for the system you just developed and an UML use cases digram.

Some theory questions in preparation for the In-class test:

1) Explain the following:

a. Class;
b. Object;
c. Class variable;
d. Method overloading;
e. Constructor.

2) What is the difference between a public field and a protected field?

3) Using UML class diagram draw an example (you can invent a typical scenario) for each of the following inter-class relationships.

1. composition
2. aggregation
3. inheritance

Something more: Typical Interview Questions/Exercises

Try to solve the following exercise. It is a typical question that is likely to be in an interview test. You will get the solution next week on BB!

1) Implement an algorithm to check if a string has all unique characters. Consider that you cannot use additional data structures.