



ASSIGNMENT 7 - SOLUTION

GENERAL INFORMATION

This assignment is made public on Friday, March 21, 2003. The assignment is due on

Friday, March 28, 1 PM.

Hand in your assignment to the teaching assistant running your lab session.

The first page of your (written) assignment has to contain at least the following information:

- the course name (Introduction to Programming - Concepts and Tools)
- your name and your student number
- name and student number of the fellow student if you submit in pairs
- assignment number

Please staple your assignment!

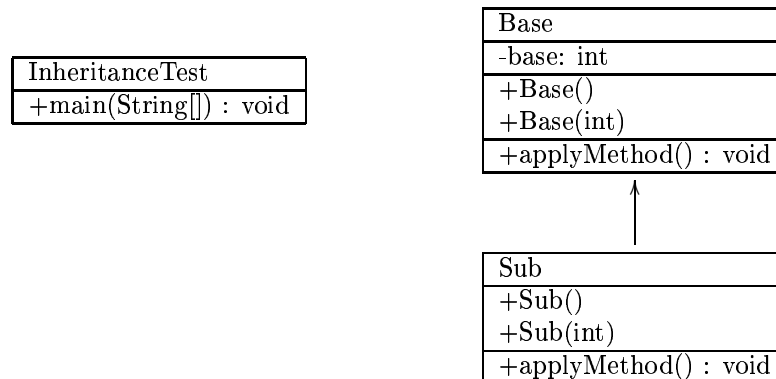
You will get back the graded assignment one week after submission deadline.

QUESTIONS

1. (*Inheritance, constructors, methods*)

SOLUTION

- (a) The UML diagram is the following:



- (b) The following is printed on the screen:

```

Base constructor with no argument called.
Instance variable base has value 0.
Base constructor with argument 3 called.
Instance variable base has value 3.
Base constructor with no argument called.
Instance variable base has value 0.
Sub constructor with no argument called.
Instance variable base has value 0.
Base constructor with no argument called.
Instance variable base has value 0.
Sub constructor with argument 4 called.
    
```

Instance variable base has value 0.

Method from Base class applied.

Method from Sub class applied.

Method from Sub class applied.

Method from Sub class applied.

The question is why:

- The line `Base b1 = new Base();` creates an object of class `Base` and calls the `Base` constructor with no argument as is clear from the syntax.
- The line `Base b2 = new Base(3);` creates an object of class `Base` and calls the `Base` constructor with one integer argument as is clear from the syntax.
- The line `Sub s1 = new Sub();` is more tricky. In theory the constructor from the derived class `Sub` is called with no argument (as can be seen from the syntax). However, the very first command is an explicit or implicit call of a constructor of the base class. The question is which one. The code of the constructor `Sub()` does not contain an explicit call to a constructor from the base class (if it would, this call would have to be the first command in the implementation of the constructor), hence the “default” one with zero arguments is called. Here default refers to having no arguments, but this is not the one provided by the compiler (why not?). Thus the constructor `Base()` is called *first*, setting the variable `base` to 0, and then the code of the constructor of the derived class is executed, printing out the message.
- The next line `Sub s2 = new Sub(4);` is similar as above. Since the implementation of the constructor `Sub(int)` does not contain an explicit call to a constructor of the base class, the constructor `Base()` will be called *first*, setting the variable `base` to 0, after which the code of the constructor `Sub(int)` is executed, which does not change the value of `base`.
- For the next four lines the point to note is that *at run-time* it is determined what kind of object is calling a method, and the appropriate method will be used. The type of `b1` and `s1` are `Base` and `Sub` respectively, so the methods from those two classes are called. After the assignment `b1 = s2` the pointer `b1` actually points to an object of class `Sub` (which is also an object of class `Base`). At run-time this can be determined, so it is the method from the *subclass* that is actually called in the line `b1.applyMethod();` Finally, there is no change to `s1`, so again it is the method of the subclass that is called.

2. (*Pixel class*)

SOLUTION

(a) The UML class diagram is as follows:

Pixel
-x : int -y : int
+Pixel(int, int)
+xCoord() : int +yCoord() : int +translate(int, int) : void

(b) class Pixel{

```
private int x;  
private int y;
```

```
Pixel(int x, int y){  
    this.x = x;  
    this.y = y;  
}
```

```
public int xCoord(){return x;}
```

```
public int yCoord(){return y;}
```

```
public void translate(int deltaX, int deltaY){  
    x+=deltaX;  
    y+=deltaY;  
}
```

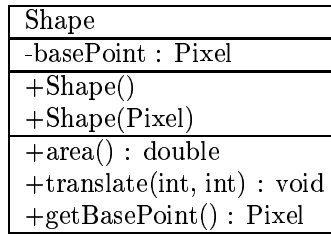
```
}
```

(c) I leave it to you how you test your class Pixel.

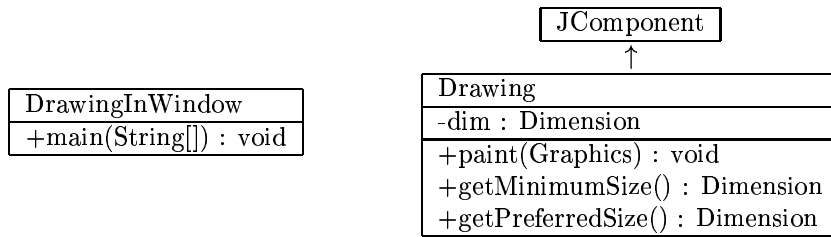
3. (*DrawingInWindow*)

SOLUTION

(a)



- (b) The class is *abstract*. This means in particular that it usually contains abstract methods (as is here the case), i.e., methods, which have to be implemented in derived classes. In particular it is not possible to instantiate objects of class `Shape` (directly), even though we have constructors! The only way to get objects of class `Shape` is to instantiate objects of a derived class which is not abstract. Those objects will in particular be of class `Shape`.
- (c) The two UML class diagrams are the following. To make it clear we also included a reference to class `JComponent` from which `Drawing` is derived. Of course we do not list all attributes and methods of the class `JComponent`.



(d) There is nothing to be said here.

4. (Class *Circle*, class *Rectangle*)

SOLUTION

(a) class *Circle* extends *Shape*{

```
    private int diameter;

    Circle(Pixel p, int d){
        super(p);
        diameter = d;
    }

    double area(){
        return diameter*diameter*Math.PI/4;
    }

    void translate(int deltaX, int deltaY){
        basePoint.translate(deltaX,deltaY);
    }

    int getDiameter(){return diameter;}
}
```

(b) I leave it to you how your test your class.

(c) class *Rectangle* extends *Shape*{

```
    private int width;
    private int height;

    Rectangle(Pixel ul, int width, int height){

        super(ul);
        this.width = width;
        this.height = height;
    }

    double area(){
        return width*height;
    }

    void translate(int deltaX, int deltaY){
        basePoint.translate(deltaX,deltaY);
    }

    int getWidth(){return width;}

    int getHeight(){return height;}

}
```

(d) Again it is left to you how you test your class.

(e) There is nothing to be said here.

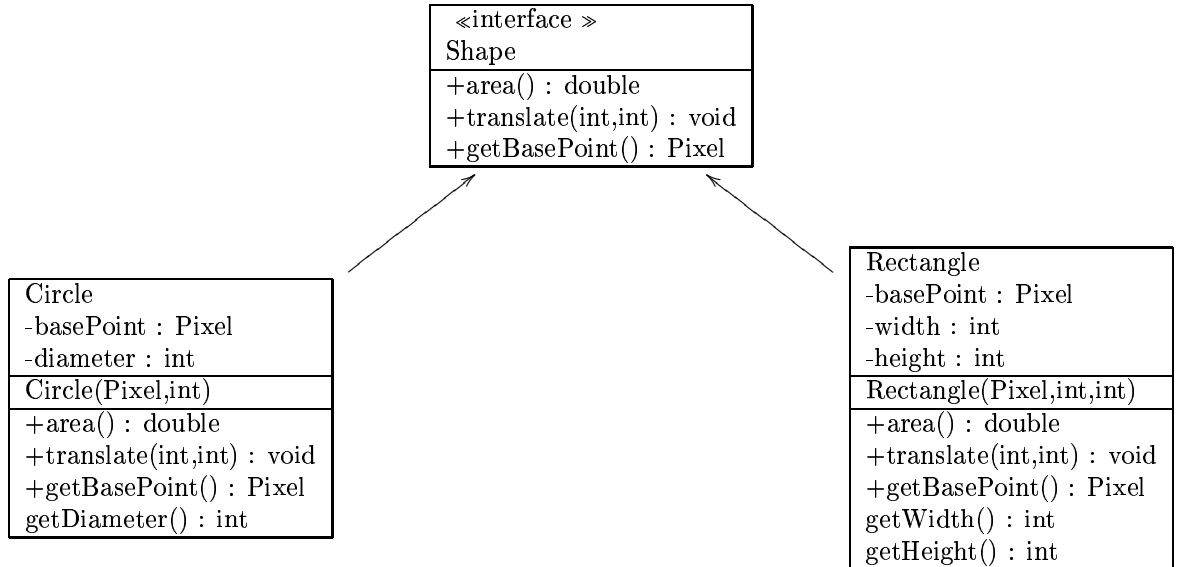
(f) For example, you could translate the rectangle *r1* again, adding (copying) the code

```
r1.translate(30,30);
g.drawRect(r1.getBasePoint().xCoord(),r1.getBasePoint().yCoord(),
r1.getWidth(),r1.getHeight());
```

5. (Interfaces)

SOLUTION

- (a) Note that `Shape` does not contain any constructors, that we had to move the instance variable `basePoint` into the separate classes (and duplicate some code!), and that the methods mentioned in the interface *have* to be public. Interface methods are always public, and if you do not declare their implementations as public you are narrowing the access to methods, which is forbidden.



- (b) `class Circle implements Shape{`

```

    private Pixel basePoint;
    private int diameter;

    Circle(Pixel p, int d){
        basePoint = p;
        diameter = d;
    }

    public double area(){
        return diameter*diameter*Math.PI/4;
    }

    public void translate(int deltaX, int deltaY){
        basePoint.translate(deltaX,deltaY);
    }

    public Pixel getBasePoint(){return basePoint;}

    int getDiameter(){return diameter;}
}

```

- (c) `class Rectangle implements Shape{`

```

    private Pixel basePoint;
    private int width;
    private int height;

    Rectangle(Pixel ul, int width, int height){

```

```
    basePoint = ul;
    this.width = width;
    this.height = height;
}

public double area(){
    return width*height;
}

public void translate(int deltaX, int deltaY){
    basePoint.translate(deltaX,deltaY);
}

public Pixel getBasePoint(){return basePoint;}

int getWidth(){return width;}

int getHeight(){return height;}
}
```

(d) There is nothing to be said here.