



ASSIGNMENT 7

GENERAL INFORMATION

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

Friday, March 28, 1pm.

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.

WARM-UP AND SUGGESTIONS FOR FURTHER EXERCISES

As a warm-up I suggest the review questions from Chapter 7: Inheritance (pp. 265/266 of my copy of the textbook).

If you want to do further programming exercises I suggest the following ones from the book (pp. 267–268):

- Exercises 1–6, asking you to modify and extend classes `Person` and `Student` from Chapter 7.
- Exercises 9, 10, and 16, asking you to modify the `Counter` class that was discussed in week 6.

It is up to you whether or not you want to work on these exercises. Those exercises will not be marked.

QUESTIONS

1. (*Inheritance, constructors, methods*)

LEARNING OBJECTIVES: UML class diagrams; dynamic typing (selecting the correct method at run-time).

Consider the following three classes `InheritanceTest.java`, `Base.java` and `Sub.java` which you can download from the course homepage:

```
class InheritanceTest{

    public static void main(String[] args){

        System.out.print("\n");

        /* Test of constructor methods */

        Base b1 = new Base();
        Base b2 = new Base(3);

        Sub s1 = new Sub();
        Sub s2 = new Sub(4);

        System.out.print("\n");

        /* Test of instance methods */

        b1.applyMethod();
        s1.applyMethod();

        b1 = s1;

        b1.applyMethod();
        s1.applyMethod();

        System.out.print("\n");

    }
}

class Base{

    int base = 1;

    Base(){
        base = 0;
        System.out.println("Base constructor with no argument called.");
        System.out.println(" Instance variable base has value " + base + ".");
    }

    Base(int b){
        base = b;
        System.out.println("Base constructor with argument " + b + " called.");
        System.out.println(" Instance variable base has value " + base + ".");
    }

    void applyMethod(){
        System.out.println("Method from Base class applied.");
    }
}
```

```

}

class Sub extends Base{

    Sub(){
        System.out.println("Sub constructor with no argument called.");
        System.out.println(" Instance variable base has value " + base + ".");
    }

    Sub(int b){
        System.out.println("Sub constructor with argument " + b + " called.");
        System.out.println(" Instance variable base has value " + base + ".");
    }

    void applyMethod(){
        System.out.println("Method from Sub class applied.");
    }

}

```

- (a) Draw the UML class diagram for the program.
- (b) What is printed on the screen? Try to think about this *before* running the program. (In the exam you won't have the possibility to run a program!). Explain for each constructor or method call why that particular constructor or method is called.

WHAT TO SUBMIT: Submit the UML class diagram, and a verbal description of what will be seen on screen.

2. (*Pixel class*)

LEARNING OBJECTIVES: Implementation of a simple class; testing of a simple class.

As a warm-up we will design a class `Pixel` similar to the `Point` class we designed earlier. A *pixel* is the smallest unit that we can display on the screen (or another displaying device). The following are the requirements on the class `Pixel`:

- A pixel is represented by two integer values giving its x - and y -coordinate.
- There should be a constructor method that takes two integer values and assigns those values to the two coordinates.
- There should be methods `xCoord` and `yCoord` that return the respective coordinates. (What are the return types of the two methods?)
- There should be a method `translate(int deltaX, int deltaY)` that translates the pixel by `deltaX` in the x -direction and by `deltaY` in the y -direction.

Your tasks are the following:

- (a) Draw the UML class diagram for the class `Pixel`.
- (b) Implement the class `Pixel` according to the requirements above.
- (c) Write a program `PixelTest` that will test your class.

WHAT TO SUBMIT: Submit the UML class diagram, a printout of your class `Pixel`, a printout of your program `PixelTest`, and a printout of what is seen on the screen when running `PixelTest`.

3. (*DrawingInWindow*)

LEARNING OBJECTIVES: UML class diagram; reading code; submitting code to larger projects; sticking to specifications.

Download the following files from the course homepage:

- `DrawingInWindow.java`
- `Drawing.java`
- `Shape.java`
- `Pixel.class`
- `Circle.class`
- `Rectangle.class`

Compile the file `DrawingInWindow.java` which contains the main program, and run it. A window should pop up containing two circles and three rectangles. Ignore the warnings saying something like

```
Font specified in font.properties not found  
[--symbol-medium-r-normal--*-%d-*-*p-*adobe-fontspecific]
```

- Draw the UML class diagram for class `Shape.java`.
- What kind of class is it? What does this mean? What can you say about objects of class `Shape`?
- Draw also UML class diagrams for the classes `Drawing.java` and `DrawingInWindow.java`.
- Delete the file `Pixel.class`. Recompile `DrawingInWindow.java` and run the program. This time the program uses *your* implementation of the `Pixel` class. If your implementation doesn't adhere to the requirement specification you will most likely get an error message. You can take this as another test of your implementation of class `Pixel`.

Note: The program, i.e., the `main` method in `DrawingInWindow.java` creates an object of class `Drawing`, creates a window and displays that object in that window. The class `Drawing` specifies what will be displayed in the window. In this case three pixels are created as points in a coordinate system. Then two circles and three rectangles are created, and are made part of the drawing.

WHAT TO SUBMIT: Submit the UML class diagrams of the classes `Shape`, `Drawing`, and `DrawingInWindow`. Submit also a description of what type of class `Shape` is, what this means (i.e., how it differs from other classes), answering in particular whether or not one can create objects of that class.

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

LEARNING OBJECTIVES: Implementing classes using inheritance.

In this question you will provide your own implementations of the classes `Circle` and `Rectangle`. The specification for the latter differs from the one that we used earlier.

Both classes should be derived from class `Shape`. A shape is a geometric figure and consists of a base point (which is here the upper left corner of the shape) and methods to return that base point, and to translate the base point. Note that all instance variables below are integer values (this is necessary since the functions drawing the shapes take integer values as input).

- (a) Class `Circle` has the following UML class diagram (showing only those parts new to class `Circle`):

Circle
-diameter : int
Circle(Pixel, int)
+getDiameter() : int +area() : double +translate(int, int) : void

A circle object is thus a shape object with the additional instance variable `diameter` holding the diameter of the circle. The constructor takes a pixel and an integer value and uses the pixel as base point and the integer value as diameter of the circle. (The way we will use the base point later in class `Drawing` is that it specifies the upper left corner of the smallest square containing the circle.) The method `getDiameter()` simply returns the diameter of the circle. The method `area()` calculates the area of the circle, and the method `translate()` translates the circle by the given two parameters. Note that this means that only the base point is moved, since the translated circle has the same diameter as the original one.

Implement class `Circle`.

- (b) Write a short program `CircleTest.java` to test your class.
- (c) Class `Rectangle` has the following UML class diagram (showing again only those parts new to the class):

Rectangle
-width : int -height : int
Rectangle(Pixel, int, int)
+getWidth() : int +getHeight() : int +area() : double +translate(int, int) : void

Thus, a rectangle is a shape with base point (specifying the upper left corner of the rectangle), and additional instance variables for width and height of the rectangle. The constructor takes a pixel as base point, and the width and height of the rectangle as arguments. The meaning of the methods `getWidth()` and `getHeight()` is obvious. The method `area()` returns the area of the rectangle, and the method `translate(int, int)` translates the rectangle according to the parameters. Note again that the way we represent a rectangle this means that we only translate the base point, since width and height remain the same.

Implement class `Rectangle`.

- (d) Write a short program `RectangleTest` that will test your class.
- (e) Recompile the file `DrawingInWindow.java`. This time the compiler will use *your* implementation of the classes `Circle` and `Rectangle`. If you did everything correct the class file should compile without error message and run.
- (f) Add one or more shapes (i.e., circles and rectangles) to your drawing. This is done by adding lines to the class `Drawing.java`.

The window has height and width 500 (pixels). The coordinate system has the upper left corner as origin, the (positive) x -direction pointing to the right, the (positive) y -direction pointing *downwards*.

The method `drawOval` takes as arguments the x and y value of the base point and the width and height of the oval (ellipse), which in case of a circle is just the diameter.

The method `drawRect` also takes as arguments the x and y value of the base point and the width and height of the rectangle.

WHAT TO SUBMIT: Submit printouts showing the code of the following classes: `Circle`, `CircleTest`, `Rectangle`, `RectangleTest`, your modification of `Drawing`, and a printout with the screen shot showing your new drawing.

5. (*Interfaces*)

This question is optional.

LEARNING OBJECTIVES: Interfaces; difference (or non-difference) between abstract classes and interfaces.

The class `Shape` could have been implemented as an interface, with `Circle` and `Rectangle` as implementations of the interface. In this question you are supposed to do so. Not that this task is not as big as it might sound since for these simple classes the changes are minor. *Do not forget to make a copy of the files `Shape.java`, `Circle.java` and `Rectangle.java` before doing so.*

- (a) Draw the new UML class diagram representing now the three classes with `Shape` as interface.
- (b) Write the code for class `Shape`
- (c) Write the code for classes `Circle` and `Rectangle`.
- (d) Test your classes, and submit the code for the interface and the two classes.

WHAT TO SUBMIT: Submit the code of the interface `Shape`, and of the new implementations of the classes `Circle` and `Rectangle`. Also submit the code of the program that tests your classes, plus a screen shot showing the program running.