Software Development I – Exercises **Winter Term 2012/2013**

Übungen zu Softwareentwicklung 1 **Assignment 9**

Name: _____ Teaching Assistant: _____

Student ID (Matr.Nr.): _____ Points (max. 24): _____

Group: _____ Deadline: Wed, Jan. 23th, 2013 12:00 noon

Instructor: _____ Editing time: _____

Question 1: MiniCAD (Operations on Geometric Shapes) **24 points**

Implement a program to work with different geometric shapes. The user interface of the software consists of a tree view listing all the created shapes, a drawing panel to visualize different objects in various colors and a text box at the bottom to enter commands (see figure on page 2).

The program must be able to work with circles (class **`Circle`**), triangles (class **`Triangle`**), and rectangles (class **`Rectangle`**), all of them derived from the class **`Geometry`**. This class provides basic functionality and fields/attributes for each geometric shape (e.g., unique id, position, paint method, etc.). By default, objects are drawn with only a border (no filling). Rectangles, however, can be either filled or empty (use a boolean attribute to distinguish between outline and filled drawing, see example below).

Each geometric shape needs to have an absolute position (`posX` and `posY` represent the bottom left corner of each shape), a color (border color for all shapes and also fill color for filled rectangles, which can be blue, green, red, and yellow only) and various methods to provide the required functionality. You need to implement at least the following methods:

- **`move(int x, int y)`** – moves the geometric shape by the number of pixels given as parameter (negative values move the object to the left (or down), positive to the right (or up), as mentioned the origin (0,0) is in the bottom left corner)).
- **`scale(double factor)`** – scales the geometric shape by the given factor (e.g., if the given factor is 2.0, the radius of a circle would become twice as long as before, given a value below 1.0 the size would shrink accordingly)
- **`paint()`** – paints the shape to the panel using the position, color and other attributes of the shape

The paint functionality is encapsulated in the class **`Turtle`**, which provides basic functionalities to draw straight lines in a graphics panel. Imagine a turtle holding a pencil that can move on your panel and only understands simple commands like:

- **`forward(double n)`** – moves the turtle n steps (pixels) forward and thereby draws a straight line
- **`left(int angle)`** – turns the turtle <angle> degrees to the left.
- **`right(int angle)`** – turns the turtle <angle> degrees to the right.
- **`setColor(Color color)`** – sets the color to be used for drawing lines.
- **`setPos(int x, int y)`** – Sets the absolute position of the turtle.
- **`setAngle(int angle)`** – Sets the absolute angle of the turtle (0 degrees lets the turtle face to the east).

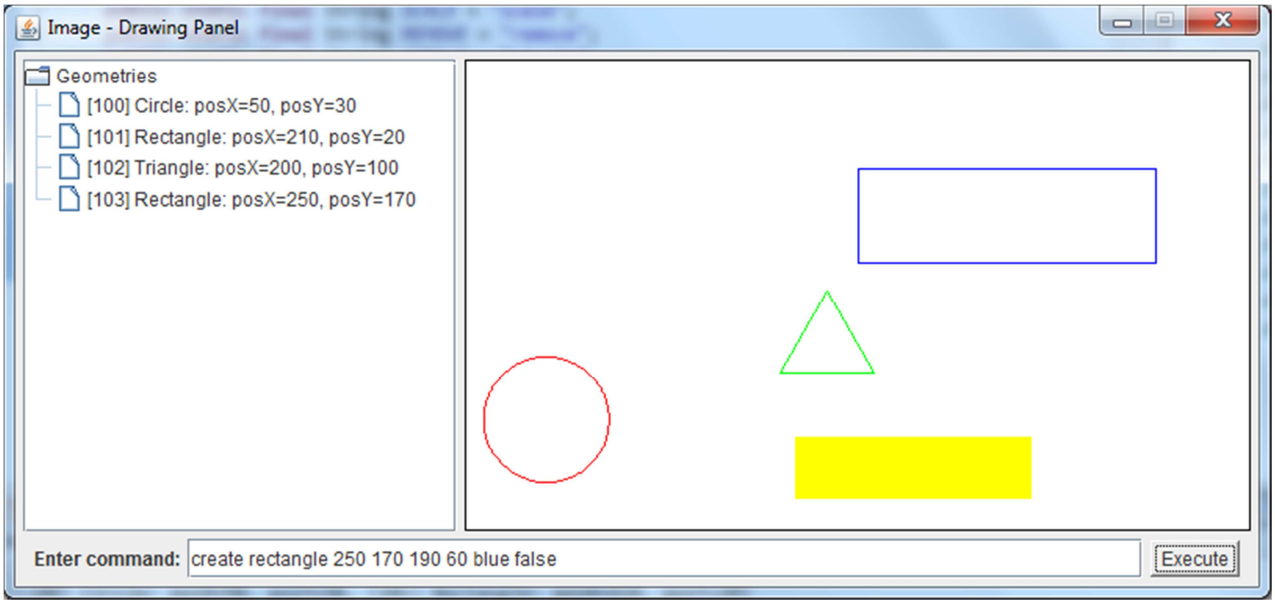
Each method can be accessed statically (e.g., `Turtle.setAngle(270)`) and apart from the above listed methods you **must not use** any other method of the class Turtle. Certainly each shape needs to implement the `paint`-method individually to visualize the according graphics on the panel and may only make use of the `Turtle`-class.

The program must support **creating** new shapes, **removing** existing objects, **moving** certain geometries on the panel and **scaling** any existing object by providing particular commands as illustrated with the examples below:

Creating new shapes
Creating a new shape should be possible by entering a command starting with "*create*" followed by the type of the shape (i.e., circle, rectangle, triangle) and the type-specific parameters for each geometry (e.g., radius, length, width, position, color, etc.)

The following commands produce the shapes illustrated in the figure below:

```
create circle 50 30 40 red
create rectangle 210 20 150 40 yellow true // "true" → filled
create triangle 200 100 60 green
create rectangle 250 170 190 60 blue false // "false" → no fill
```



Removing existing shapes
For removing existing shapes you should use the unique id of each geometric shape (an increasing number starting with 100) and for example simply execute the following command:

```
remove 100
```

This command removes the object with the id 100 from the list and automatically updates the panel as well.

Moving existing shapes
For moving existing shapes also use the unique id of the object. The following command moves the object with the given id 40 pixels to the right and 50 pixels up.

```
move 101 40 50
```

Scaling existing shapes
For scaling existing shapes you should be required to enter a command like the following:

```
scale 102 1.8
```

Implementation details
In order to make the exercise easier for you, you are given an **existing Java framework** which you need to complete. The places to insert your Java code are clearly marked with the "FIXME" keyword and some additional comments can also be found there. In detail you are required to complete the following classes:

- Package ***geometry*** – Implement the required functionality for geometric shapes like painting, moving and scaling and bear in mind the **concepts of polymorphism**.
  - `Geometry.java`
  - `Circle.java`
  - `Rectangle.java`
  - `Triangle.java`
- Package ***model*** – Complete the methods to remove, scale and move geometries.
  - `Model.java`
- Package ***gui*** – Complete the method required for parsing the commands entered by the user
  - `ConsolePanel.java`

Apart from the above mentioned classes <u>you must not change any other class</u> in the framework!

Hints
- Since the focus of this exercise is polymorphism, please try to put as much functionality into the **super class Geometry** and **derive** all the other shapes from this class.
- Also make use of **super calls** in the constructors and think about methods that can probably already be implemented in the super class (i.e., `Geometry`).
- For simplicity reasons triangles can only be equilateral (the third parameter is the length of each side)
- For drawing filled rectangles you can simply draw multiple lines of equal length next to each other (only rectangles need to be filled)
- You do not need to account for detailed error messages, but if a command cannot be parsed successfully or an operation fails for any reason please provide a popup to inform the user about the problem (you can use the Java class `JOptionPane` for this purpose – see example below)
- You can make use of the class `util.Text` to avoid string constants in your code

To show a simple popup with an error message you may use the following code snippet:
```
JOptionPane.showMessageDialog(getParent(), "Error message",
"Title", JOptionPane.ERROR_MESSAGE)
```

Write the program in Java. Provide all the material requested below at the very bottom ("Requested material…")

Requested material for all programming problems:

- **For each exercise, hand in the following:**
  a) **The idea for your solution written in text form**
  b) **Source code (Java classes including English(!) comments)**
  c) **Test plan for analyzing boundary values (e.g., minimal temperature allowed, maximum number of input, etc.) and exceptional cases (e.g., textual input when a number is required, etc.). State the expected behavior of the program for each input and make sure there is no "undefined" behavior leading to runtime exceptions. List all your test cases in a table (test case #, description, user input, expected (return) values).**
  d) **The output of your java program for all test cases in your test plan**
- Pay attention to using adequate and reasonable data types for your implementation, check the user input carefully and print out meaningful error messages.