

# Class Specification and Implementation Review

## Purpose:

The purpose of this lab is to review fundamental aspects of objects and of applications composed with objects.

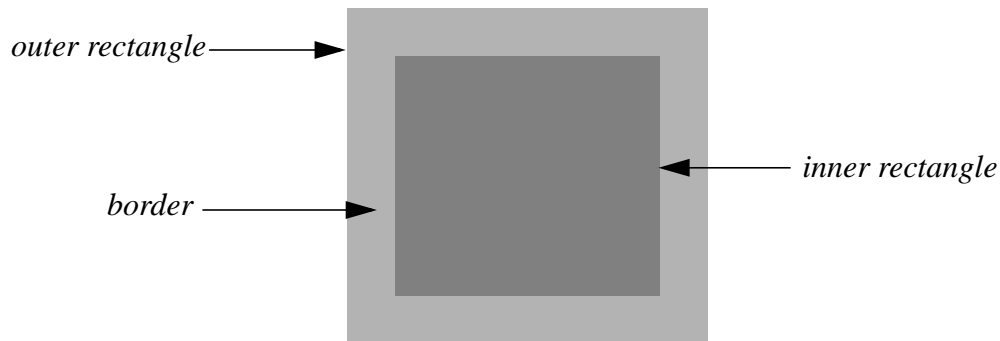
## The library class *java.awt.Color*:

For the application to be specified in this lab, we will need a class to model colors. The standard Java library provides such a class.

- Open and browse the specification for the standard library class *java.awt.Color*. Standard Java API specifications can be found at <http://java.sun.com/j2se/1.3/docs/api/>.
- Answer the following questions:
  - What is a class specification? What does it contain?
  - For whom is a class specification written?
  - How do you produce a specification document such as the one you are browsing?
  - How are common colors like red, blue, green, specified in the class *Color*?
  - A client class of the class *Color* will likely have a *Color* attribute. How do we specify such an attribute in Java?
  - How do we make the attribute reference the *Color* instance blue?

## Functional requirements of an application:

We want to write a simple application that draws two rectangles, one inside the other as pictured below.



Each rectangle is a different color, and the inner rectangle has smaller dimensions than the outer rectangle and is centered in the outer rectangle. The inner rectangle appears on top, so that only the edges of the outer rectangle are visible forming a border around the inner rectangle.

The rectangles are measured in pixels, and the maximum size of the outer rectangle is 250 x 250 pixels.

A user should be able to enlarge or reduce each rectangle, subject to the constraint that the inner rectangle remains smaller than the outer rectangle, and remains centered in it. A user can change the color of either rectangle, and swap the rectangle colors. A user can request the area of each rectangle, as well as the area of the border.

- Copy the files from `~labCourse/Labs/Lab21/flipper/` into a subdirectory of your *Java* directory named *Lab21.flipper*.
- Run *Lab21.flipper.RecFlipperGUIStart*. This will give you an idea of what the application should look like.

### **Note:**

To avoid confusion with classes you will be constructing, it may be useful at this point to delete the file *NestedRectangles.class* from your *flipper* directory.

### **Design of the model classes:**

#### **The class *Rectangle*:**

One object that we can design is a rectangle with color; let's call the class *Rectangle*. The application consists of two nested *Rectangles* with some extra functionality.

The specification for the class *Rectangle* is *here*. An implementation is provided. Read the file *Rectangle.java* and note the use of the class *Color*.

#### **The class *NestedRectangles*:**

We will use the *Rectangle* class to design and implement a class that allows us to model two nested rectangles. We'll call this class *NestedRectangles*. A *NestedRectangle* will have two instance variables, each referencing a *Rectangle*.

- What is the relationship between the class `Rectangle` and the class *NestedRectangle*?

A careful reading of the requirements reveals that the class *NestedRectangles* must provide the following functionality:

- give the area of the outer rectangle (*area*);
- give the area of the inner rectangle (*innerArea*);
- give the area of the border (*borderArea*);
- change the width of the inner rectangle (*setInnerWidth*);
- change the height of the inner rectangle (*setInnerHeight*);
- change the width of the outer rectangle (*setOuterWidth*);
- change the height of the outer rectangle (*setOuterHeight*);
- change the color of the outer rectangle (*setBackground*);

- change the color of the inner rectangle (*setForeground*);
- swap the colors of the rectangles (*flipColors*).

In addition, we'll provide the following queries:

- give the height of the outer rectangle (*outerHeight*);
- give the width of the outer rectangle (*outerWidth*);
- give the color of the outer rectangle (*backgroundColor*);
- give the height of the inner rectangle (*innerHeight*);
- give the width of the inner rectangle (*innerWidth*);
- give the color of the inner rectangle (*foregroundColor*);

Finally we specify three constructors:

```
public NestedRectangles (int outerHeight, int outerWidth,
    int innerHeight, int innerWidth, Color background, Color foreground)

public NestedRectangles (int outerHeight, int outerWidth,
    int innerHeight, int innerWidth)

public NestedRectangles (Rectangle outerRec, Rectangle innerRec)
```

with the obvious semantics. The second constructor uses black as the default background color, and white as the default foreground color.

- Write a Java specification for the class *Lab21.flipper.NestedRectangles* containing the methods and constructors given above. (Do not write a complete implementation at this point.)

An invariant condition for this class requires that the dimensions of the inner rectangle are less than the corresponding dimensions of the outer rectangle.

- Include a statement of this invariant in your specification.
- Be sure to include doc comments for the class and for each method and constructor. Use *require* and *ensure* clauses as appropriate.
- As directed by your instructor, include a comment at the beginning of the file with your name, course, and section.

Recall that documentation convention is that class names, method names, *etc.* are written in “code” font. To produce source for *javadoc* that includes appropriate HTML tags, you can use the utility *prejavadoc*. This is a Perl script located in *~labCourse/utilities*. Input for the script (standard input) is a Java source file, and output (standard out) is the file with HTML tags added in doc comments. The script will put appropriate HTML tags around preconditions and postconditions, and will wrap `<code>...</code>` tags around words prefixed with `cx` and around lines prefixed with `cz` in doc comments.

For example, if the method *setInnerWidth* is written as:

```

/**
 * Change the width of the inner cxRectangle to the specified
 * value.
 *   require:
 *     czvalue > 0 && value < outerWidth().
 *   ensure:
 *     czinnerWidth() == value
 */
public void setInnerWidth (int value){
    ...
}

```

*prejavadoc* will produce the following output

```

/**
 * Change the width of the inner <code>Rectangle</code> to the specified
 * value.
<p><dd><dl>
 *   <dt><b>Require:</b>
 *     <dd><code>value > 0 && value < outerWidth().</code><br>
</dl></dd>
<p><dd><dl>
 *   <dt><b>Ensure:</b>
 *     <dd><code>innerWidth() == value</code><br>
</dl></dd>
 */
public void setInnerWidth (int value){
    ...
}

```

You can, of course, explicitly include your own HTML tags in a doc comment.

To use the *prejavadoc* utility, first make a copy of your source file. For instance,

```
cp NestedRectangles.java temp.java
```

Then use the copy as input. For example,

```
~labCourse/utilities/prejavadoc < temp.java > NestedRectangles.java
```

or

```
perl ~labCourse/utilities/prejavadoc < temp.java > NestedRectangles.java
```

### **Stubbed implementation of the class *NestedRectangles*:**

Next you must build a stubbed implementation of the class so that it will compile with no syntactic errors. Commands and constructors can be stubbed with empty bodies. For instance,

```
public void setInnerWidth (int value){
}

```

Queries can be stubbed with a body consisting on a simple *return* statement. For instance,

```
public int area () {
    return 0;
}

```

```
}
```

Make sure that you can successfully compile your implementation.

- Use *javadoc* to produce HTML specifications.

For instance, create a directory named docs to contain the HTML files. Then execute

```
javadoc -d docs Lab21.flipper.NestedRectangles
```

### **Post-lab:**

Submit answers to the question and a browser printout of the *NestedRectangles* specification.