University of Washington
Computer Science & Engineering 142 (Computer Programming I), Summer 2005

Programming Assignment #3 (SimpleFigure and Circles)
Due: Tuesday, 7/12/2005, 6:00 PM

Program Description:

This assignment will give you practice with value parameters, using Java objects, and graphics. This assignment has 2 parts; therefore you should turn in two Java files.

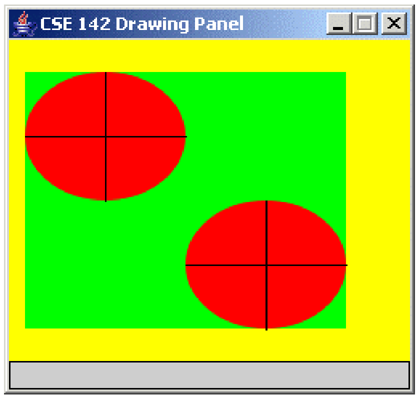
You will be using a special class called `DrawingPanel` written by the instructor, and classes called `Graphics` and `Color` that are part of the Java class libraries.

Part 1 of 2 (4 points)
Simple Figure, or your own drawing

For the first part of this assignment, turn in a file named `SimpleFigure.java` that draws a figure on the screen using the `DrawingPanel` provided in class. You may draw any figure you like that meets the following "external correctness" criteria:

- The `DrawingPanel`'s size must be at least 100 x 100 pixels.
- You must draw at least one line, one rectangle, and one oval that are visible on the screen.
- You must use at least three different visible colors in your figure.

The loose guidelines for Part 1 mean that if you like, you can be creative and draw any figure you like! After the assignment is turned in, the instructor may anonymously show off some of the neat figures drawn by students for everyone to see.



If you do not want to create your own figure, the following is a default figure you may draw for Part 1. If your code draws this figure correctly, you will receive full credit for Part 1. This figure contains the following elements:

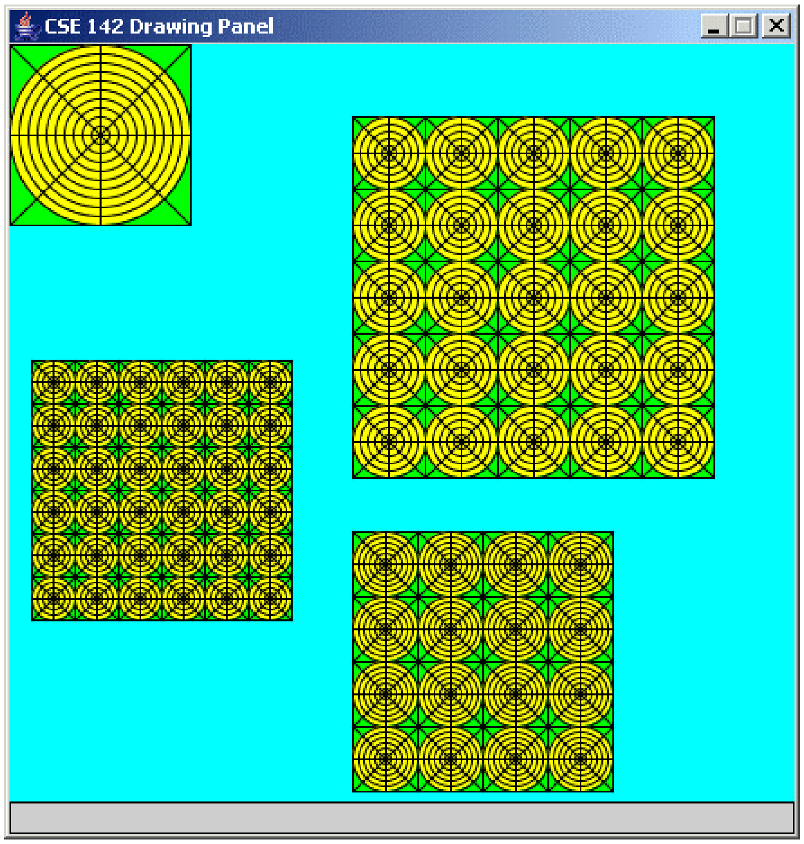
- A `DrawingPanel` of size 250 x 200 with a yellow background.
- A green rectangle with top-left corner at (10, 20) and size 200 x 160 pixels.
- Red ovals with top-left corners at (10, 20) and (110, 100) of size 100 x 80 pixels.
- Black lines from (10, 60) to (110, 60), from (60, 20) to (60, 100), from (110, 140) to (210, 140), and from (160, 100) to (160, 180).

Since a major new concept for this assignment is parameter-passing, you may optionally wish to try using parameterized methods to draw your figure. For example, the default picture above has a repeated figure of an oval with black lines through it. This figure could be turned into a parameterized static method with parameters for the figure's position.

However, parameterized methods are not required for Part 1, and you will not be penalized if you do not use them. Your entire score for Part 1 will be based on its external correctness as defined above.

Part 2 of 2 (16 points)
Circles

Part 2 of this assignment asks you to draw a complex figure using parameterized methods. You must reproduce exactly the figure shown in the picture on this page; you may not create your own figure for this part. Name your Java class `Circles` and your file `Circles.java`.



You will use complex computations to draw a figure that has several levels of structure. You are to produce the figure image as output. Your program should exactly reproduce the following image. (The image in this document was taken when running the program on Windows; if you use another platform, the decorations around the window may look slightly different, but the actual contents of the window should be identical.)

This image has several levels of structure. You will find that there is a basic subfigure that occurs throughout, which is a green square with concentric circles inside it. The subfigure is repeated to form larger figures.

A major part of this assignment is showing that you understand parameters. Therefore, you should write parameterized methods to represent the tasks of drawing one subfigure and of drawing a larger figure composed of many subfigures.

Figure Specifications:

The overall panel has a cyan background and is 400 pixels wide and 425 pixels high. Each of the subfigures is has an overall rectangular green background, containing a yellow inner circular background, which has black circles, squares, and X lines drawn on it.

The four figures on your drawing panel should have the following properties.

Description	**(x, y) position**	**number of rows and columns**	**size of each subfigure, in pixels**	**number of concentric circles in each subfigure**
top-left figure	(0, 0)	1 x 1	100 x 100	10
bottom-left figure	(18, 175)	6 x 6	24 x 24	4
top-right figure	(180, 25)	5 x 5	40 x 40	5
bottom-right figure	(180, 250)	4 x 4	36 x 36	6

Implementation Guidelines for Part 2:

You are required to have two particular static methods that are described below. You will not be using class constants for this assignment. Instead, you will be exploring the use of value parameters.

This program does not require a lot of lines of code, but the numeric computations and parameter-passing required are not simple. You might very well find yourself overwhelmed with the amount of detail you have to handle all at once. A famous computer scientist named Brian Kernighan once said, “Controlling complexity is the essence of computer programming,” which is why this makes a good exercise for a beginning computer scientist. You will find that the techniques of decomposition and iterative enhancement described in chapter 1 will help you in this situation.

Because it is difficult to correctly implement such a complex program all at once, instead start with one smaller piece of the problem. In particular, you should write a static method that draws one square with concentric circles inside of it. That subfigure is repeated throughout this image, so it makes sense to have a separate method for producing it. Start with the subfigure in the upper-left part of the screen. It is not in a grid pattern; it is just the subfigure itself (one set of concentric circles in a square).

Your first version of this method could draw the specific subfigure in the upper-left corner (always drawing it in that position, always making it 100 pixels wide, always having 10 concentric circles), but you will want to generalize this with the use of parameters. You should be able to call it with different sizes, different locations and different numbers of concentric circles.

The drawing commands we are using are defined in terms of integers, which leaves open the possibility that things won’t divide up evenly. You don’t have to worry about this possibility. In particular, you may assume that the subfigure height will always be a multiple of the number of concentric circles you are supposed to draw (e.g., 10 circles in a figure 100 wide, 6 circles in a figure 36 wide, 5 circles in a figure 40 wide, 4 circles in a figure 24 wide).

Once you have completed the static method that produces one of these subfigures, write another static method that produces a square grid of these subfigures. You will call this method several different times from `main` to produce the grids of the overall figure. It will have a lot of parameters to be flexible enough to draw each of these grids. The key point is that a single method can be used that produces all three grids.

Remember that to run your program, you will have to download the file `DrawingPanel.java` from the class web page and save it in the same folder where you will be storing your classes. The `DrawingPanel.java` file will be available under the "Assignments" link from the class web page.

The `Graphics` and `Color` classes are part of Java, but you will need to include the following line of code at the beginning of your program file to be able to use them:

```
import java.awt.*;
```

Stylistic Guidelines:

For this assignment you are limited to the language features in Chapters 1 through 3; you are not allowed to use more advanced features to solve the problem. You cannot, for example, make use of `if/else` statements to solve this task. You will, however, use the `DrawingPanel` and `Graphics` classes described at the end of Chapter 3 as well as the color constants from the `Color` class, as described in Chapter 3.

Continue to use static methods to structure your solution as described previously; this time, write static methods that use parameters. In grading, we will require two static methods: one that draws a subfigure with just one square and its concentric circles, and one for producing a grid of such subfigures that is called many different times to produce the various figures on the screen.

You are required to properly indent your code and will lose points if you make significant indentation mistakes. See section 2.5.3 of the book for an explanation and examples of proper indentation.

Give meaningful names to methods and variables in your code. Follow Java's naming standards about the format of ClassNames, methodAndVariableNames, and CONSTANT_NAMES. Localize variables whenever possible -- that is, declare them in the smallest scope in which they are needed.

Include a comment at the beginning of your program with basic information and a description of the program and include a comment at the start of each method.

Submission and Grading:

Name your Part 1 file `SimpleFigure.java` and your Part 2 file `Circles.java` and turn them in electronically from the "Assignments" link on the course web page. You do not have to turn in `DrawingPanel.java`. This assignment is worth 20 points total.

Part of your program's score will come from its "external correctness." External correctness for Part 1 measures whether you have met the constraints on the figure as specified on the first page, including having sufficient number of shapes and colors. External correctness for Part 2 measures whether the figure you draw matches <u>exactly</u> what is expected.

The rest of your program's score will come from its "internal correctness." Internal correctness will only be measured for Part 2; you will not be penalized for stylistic issues on Part 1. Internal correctness measures whether your source code follows the stylistic guidelines specified in this document. This includes using for loops to capture repetition in the figures, capturing the structure of the figure using the two parameterized static methods specified, commenting, naming identifiers, and indentation of your source code.