



CS 101 - Problem Solving and Object-Oriented Programming

Lab 4 - Introduction to Java

Due: October 16, in class

Introduction to the Assignment

In this assignment, we will do some small exercises to become acquainted with Java and BlueJ, rather than writing a long program. You will write a small program, but also do some careful experimentation where you record the results of what you observe. You will turn in both your small program and written results from your experimentation.

Getting started with BlueJ

BlueJ can be found in the All Programs submenu of the Start menu of the computers in Visilab. After BlueJ starts, you need to create a project. Open the Project menu and select New Project. Navigate to your folder on the file server. Enter FirstJavaProgram for the project name.

Finally, you need to create a class. Click on the New Class button and enter MyFirstClass as the class name and click OK. This will create a template for a class. Unfortunately, most of what is provided in this template you do not want. Edit what is initially displayed so that it looks like this:

```
import objectdraw.*;

/**
 * Write a description of class MyFirstClass here.
 *
 * @author (your name)
 */
public class MyFirstClass extends WindowController
{

}
```

Edit the comment so that it actually describes your class and includes your name (without the parentheses).

Writing the Program

For our first Java program, we will simply be constructing rectangles on the screen. We will do this in several ways so that you can observe how different event handlers work and how the values of the arguments passed to the constructor affect how the rectangles that we construct look.

begin method

The begin method is similar to the “my first method” of Alice. It is the method that is executed when your Java program begins. In your begin method, we will place an instruction to construct a small black rectangle on the screen. Your begin method should be placed between the left and right curly braces of the MyFirstClass declaration. The class should look like this:

```
import objectdraw.*;

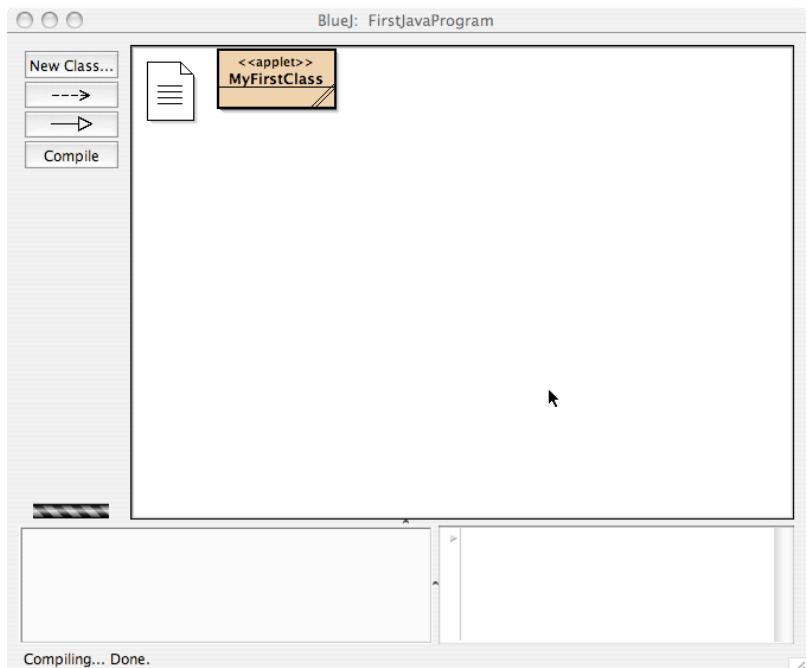
/**
 * Write a description of class MyFirstClass here.
 *
 * @author (your name)
 */
public class MyFirstClass extends WindowController
{
    public void begin() {
        new FilledRect (10, 50, 100, 200, canvas);
    }
}
```

This says that when the program starts, a new black rectangle should be drawn on the screen. The left edge of the rectangle will be 10 pixels from the left edge of the window. The top of the rectangle will be 50 pixels from the top of the window. The rectangle will be 100 pixels wide and 200 pixels tall. The final argument, `canvas`, is a predefined variable that tells Java which portion of the screen to draw on.

Make sure that your class looks exactly the same as the one above. Then click the Compile button at the top of the BlueJ window. At the bottom of the window, it should say “Class compiled - no syntax errors”. If it instead says you have an error, carefully compare what you typed with what is shown above. Java is very fussy about punctuation and spelling, so be sure those are all identical. If you have an error but cannot find the discrepancy, please ask for help.

Once your class compiles, you are ready to execute it. Go to the BlueJ project window, which looks like the figure on the right. Right-click on the rectangle labeled MyFirstClass and select “Run controller” (the last item in the menu). This should create a new window drawing the rectangle that you asked to be constructed. It should look like the window on the next page.

It’s a bit hard to see the window border, but you can tell that the rectangle is very close to the left edge of the window (10 pixels),



further from the top edge (50 pixels) and is about twice as tall (200 pixels) as it is wide (100 pixels). If your window does not look like this and you do not understand how to fix it, please ask for help.

Next, we will experiment with the various event handlers that Java provides to deal with the mouse.

onMousePress method

The `onMousePress` method is used to cause the program to do something when the user depresses the mouse button. We will take the statement that you just wrote, cut it from the `begin` method and place it inside the `onMousePress` method. After doing that, your class should look like this:

```
public class MyFirstClass extends WindowController
{
    public void begin() {
    }

    public void onMousePress (Location point) {
        new FilledRect (10, 50, 100, 200, canvas);
    }
}
```

(Your program should still have the import statements and comment as well, but I will not show those in this or future examples.)

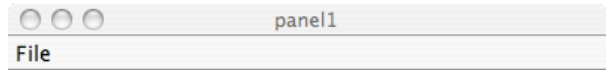
When you compile and execute this program, initially, you should have a blank screen. When the user depresses the mouse button, the rectangle will appear.

onMouseRelease method

In Java, the act of pressing the mouse button can be treated separately from the action of releasing the mouse button. When the user releases the mouse button, another event handler is called. The **signature** (the first line) of the method is identical to `onMousePress`, except for the method's name. It looks like this:

```
public void onMouseRelease (Location point) {
```

Write an `onMouseRelease` method that draws another rectangle which does not overlap the first rectangle. Change its size as well, so that it is clearly different from the first rectangle. For example, the new program might result in a drawing like this:



When you run this program, depress the mouse button and hold it down for a while. You should observe that this causes the large rectangle to be drawn. Then release the button and your second rectangle should appear.

onMouseClicked method

A mouse click consists of a mouse press followed by a mouse release with no mouse motion in between. When the user clicks the mouse, 3 event handlers are called: `onMousePress`, `onMouseRelease` and `onMouseClicked`. Depending on how we want our user interface to behave we would decide which of these methods meets our needs. Since this program is an experimentation on the capabilities of Java, rather than doing anything particularly useful, we will use all three. So, we will now add an `onMouseClicked` method to our program.

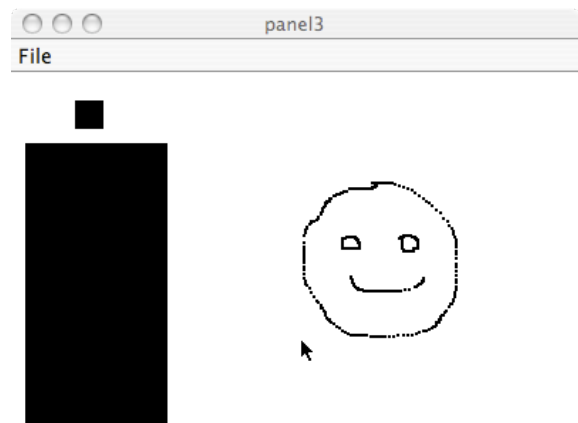
If you have been looking closely at what we have been doing, you may have noticed that the event handling methods that we have defined, except for the `begin` method, all have a parameter declared as "Location point". This is Java's way of telling our program where the mouse is when the method is called. A Location is an x, y pair. So far, when we have created rectangles, we have done so at a fixed location determined when we wrote the program. Instead, we can use the information about where the mouse is to control where we create a rectangle. To do that, we will call the `FilledRect` constructor in a slightly different way, as follows:

```
public void onMouseClick (Location point) {
    new FilledRect (point, 20, 40, canvas);
}
```

Notice that the first argument to the `FilledRect` constructor here is `point`, the parameter to the `onMouseClicked` method. The result will be that Java will now create a rectangle with its upper left corner where the user clicks. Clicking multiple times results in multiple rectangles. Try it!

onMouseDown method

The last event handler we will investigate in this lab is `onMouseDown`. This method is called repeatedly when the user moves the mouse while holding the mouse button down. For this event handler, define an `onMouseDown` handler that draws a very small rectangle, 2 pixels square, at the current mouse location. Now, when you run your program, the user should be able to draw (sort of) with your program to get a result that looks like the figure on the right, for example. It may look as though the user is drawing lines, but really she is just placing a lot of very small rectangles quite close together.



Style Issues

Indentation

Curly braces are used in Java to denote the beginning and the end of something, like a class declaration or a method declaration. Anything placed within `{...}` should be indented further than the curly braces, as in the examples above.

BlueJ helps with indentation. When you hit "return/enter" after a curly brace, Blue will automatically indent the next line further. Similarly, if you type } alone on a line, BlueJ will reduce the indentation of that line so that the } lines up with beginning of the declaration that it closes.

Be sure that the code you turn in uses indentation consistently in this way.

Experimentation

Make a copy of the program that you created up to this point. You will want to turn that program in. In this section, I ask you to make some changes to the program to experiment with Java. Many of these changes will break your program, so you will want to do this on a copy of the working program. Please answer the questions below and turn in a written copy of the results of your experimentation.

1. Run your working program. Experiment with dragging the mouse at different speeds. What do you observe about the effect of the speed of your drag with the drawings that are produced? Why do you think that happens?
2. One thing that makes Java harder to use than Alice is that it is not menu-based. Instead, you need to remember where all the punctuation goes, and Java really cares about this! You also need to remember the names of the methods you can use and what the parameters are. Alice's menu system was very convenient because it always reminded you what extra information, like distances and directions, you needed to provide. In Java, there are simply too many options for that type of user interface to work. Instead, you need to learn these things yourself. If you make a mistake in remembering these things, the Java compiler will tell you. For these questions, we will deliberately introduce mistakes in your program. I want you to record what error message you see and what you need to do to fix the error. Sometimes the same error message may indicate different problems, but it is still a good idea to get used to the error messages and what they mean. I want you to turn in a record of these observations, but also keep a copy for yourself that you can refer to in future labs. For each of these, make the change I suggest, record the error message and how to fix it. Then fix it before going on to the next item in the list.
 - a. Erase one of the semicolons. Compile the class. Record the error and how to fix it.
 - b. Erase one of the right parentheses. Compile, record, ...
 - c. Erase one of the left parentheses.
 - d. Erase one of the commas.
 - e. Erase the left curly brace that begins the class.
 - f. Erase the right curly brace that ends the class.
 - g. Erase one of the left curly braces that begins a method.
 - h. Erase one of the right curly braces that ends a method.
 - i. Add an extra right curly brace at the end of the class
 - j. Omit "void" from one of the method signatures.
 - k. Mistype "Location" in one of the method signatures.
 - l. Omit "new" from one of the constructor calls.
 - m. Omit one of the arguments in the constructor call.
 - n. Make "canvas" be the first argument rather than the last argument in a constructor call.
 - o. In a constructor call where you use the "point" parameter, mistype it.
 - p. Remove "import objectdraw.*;" from the top of the file.
 - q. Remove "extends WindowController" from the class declaration.
3. Sometimes when you make mistakes in Java programming nothing happens at all! The compiler does not complain. Instead Java just seems to ignore some of your instructions. To see this happen, try the things below. Even though these do not result in error messages, add them to your list of potential programming problems to watch out for.

- a. Change the name of one of the event handling methods to be all lower case. Java will consider this to be a different name. Compile your program. There should be no errors. Run your program. Whichever event handling method you changed the name of will not do anything. This is because Java can't find it! (It's very easy to trick Java!)
- b. Fix the name you changed in a. Now, remove the parameter declaration and try again. You should get the same result. By changing the signature, you have again tricked Java. You need to remember that these mouse event handlers always have a parameter telling you where the mouse is.

A simple way to determine if your program is executing a method that you think it should execute is to add a line like the following at the start of the code:

```
public void onmousepress (Location point) {
    System.out.println ("in onmousepress");
    new FilledRect (10, 50, 100, 200, canvas);
}
```

When you run your program with the method name mistyped, nothing will happen. After correcting the name and running your program, a Terminal window will pop up in BlueJ, with the words "in onmousepress" in it. This just lets you know that the program executed code that you were expecting it to execute. Please add this tip to your log.

Grading

25	Event handling program
5	Consistent indentation
20	Experimental write-up

Turning in Your Work

Go to ella. Click on COMSC 101 in the toolbar menu across the top. Then click on Assignments in the left column. Click the submit link for this assignment. Click the Add Attachment button. Use the Browse... button to upload a local file. You only need to submit the MyFirstClass.java file, not the other files created by BlueJ. Click Continue. Then click the Submit button. You can submit your experimental write-up either electronically or on paper.