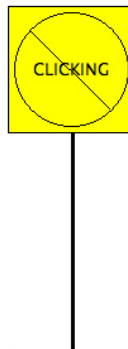


Lab 1

An Introduction to Java and Objectdraw

Objective: To demonstrate the use of the lab computers, Java, and graphics primitives.

This lab introduces you to the tools that you will use this semester. Your task is to construct a Java program that draws an image resembling a roadside warning sign, but with a message more appropriate for a computer screen:



First you will need to determine the Java commands required to draw such a picture. Then you will write a Java program that draws and modifies the picture in simple ways in response to actions the user performs with the mouse.

Part 1. Objectdraw Primitives

To help you learn how to create graphical objects, we have constructed a simple tool in which you can experiment interactively with the graphics primitives. This tool lets you create and modify objects by clicking buttons and filling in fields that describe the locations and dimensions of the objects. If you think of this program as a drawing program (like Adobe Photoshop) you will quickly conclude it is the worst drawing program in the world. Its interface is designed specifically to use mechanisms similar to those you will use from within your programs, and certainly not for usability as a drawing program. Hence its name: `NotPhotoShop`.

This part of the lab involves two steps: (i) completing a tutorial on `NotPhotoShop` and (ii) using it to draw your own rendition of our “no clicking” sign.

Logging In. To start, you must log in to one of the Macintosh computers in our lab. Begin by entering the user name and password for your computer science account in the login window. You must obtain these from your instructor. (Your computer science account is not the same as the one provided by the Office of Information Technology.) Then to log in, click the arrow or press “return”. Once you log in, the screen should display a window containing icons for some of the files you can access on the machine.

Before proceeding, change your password:

- Point the mouse at the Apple icon in the upper left corner of the screen. Press and hold down the mouse button to open a menu.
- Select “System Preferences” from this menu.
- Click once on the “Users & Groups” icon in the fourth row of icons in the window that appears. Now click on “Change Password...”.

- A window will appear where you should enter your original password and your new password in the “New Password” and “Verify” fields.
- Click the “Change Password” button.

Now, log out and log in again with your new password. To log out click on the apple icon in the top left corner of the computer screen and select “Log Out ...”. Then click the “Log Out” button that appears in the dialog box. Before continuing with the lab, be sure both partners have changed their passwords.

The files you create for our course laboratory projects will not actually be stored in the computer on which you are working. Instead, they will be stored on fuji, our department’s file server. You can access your files on fuji from any computer in our lab by simply logging in to that computer.

We have just a few more steps to set up your account. First, we will make it easier to navigate in the Finder:

- Click anywhere on the background image to switch to the Finder.
- From the Finder menu select “Preferences...”
- Set the “New Finder Windows Show” pop-up menu to your home directory, which will have a small house icon next to it in the pop-up menu’s list.
- Click on “SideBar” in the “Finder Preferences” dialog box and add a check mark next your home directory, which will, again, have a small house icon next to it.
- Close the Finder Preferences dialog box.

Next, create a folder for your cs134 work:

- Open a new Finder window by selecting the “New Finder Window” item from the File menu or, if you prefer, by clicking on the happy face on the left side of the dock at the bottom of the screen.
- Click once on the house icon on the left side of the window that appears.
- Create a new folder by selecting “New Folder” from the File menu. This will create a new folder called “untitled folder” the window.
- Change the name of this folder by clicking once on the “untitled folder” icon, then choosing “Get Info” from the File menu. In the dialog box that appears, click on the triangle next to “Name & Extension” to open up a text box where you can enter a new name for this folder. Enter “cs134”.

Safari. We’ll now set up Safari to download files to the cs134 folder you just created. Start the Safari

web browser by clicking on the compass icon  in the dock. After it starts, open the “Safari” menu. Select “Preferences”.

About three quarters of the way down under “General” is “File download location”. Click on the menu to the right of this entry. Select “Other...”. This will open a file browser, which you should use to find your cs134 folder. After clicking on that folder, use the “Select” button to cause Safari to download files to this folder.

All web browsers restrict which applets you may run to prevent potential security attacks launched by sneaky programs. Before continuing, we will configure Safari to permit you to run the sample programs on our website. (They won’t be sneaky — we promise!) You’ll only need to do this once, and you can follow these same steps to configure Safari on your own computer if you’d like to run our applets outside of lab. The website has instructions for setting up other browsers.

- Select “System Preferences” from the Apple menu at the top of the screen. Click once on the “Java” icon in the fifth row of icons in the window that appears.

- In the Java Preferences window that appears, click on the “Security” tab, and verify that the “Security level” selected is “High”.
- Then click on “Edit Site List...”. Add the following two servers to the list:

```
http://dept.cs.williams.edu/  
http://www.cs.williams.edu/  
http://cs.williams.edu/
```

You will be warned that including these locations is considered a security risk. Click “Continue”. Then click “OK”, and quit System Preferences.

- Now go back to Safari and visit:

```
http://www.cs.williams.edu/~cs134/
```

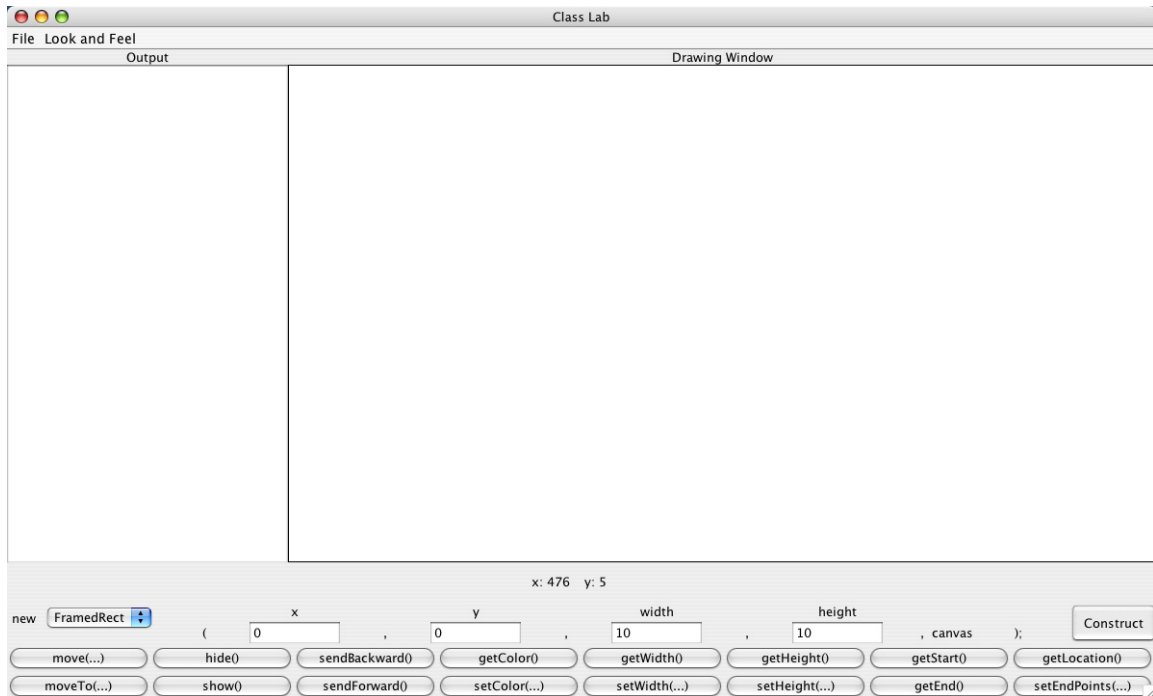
Then follow the link to the Handouts page. Click on the “Demo” link for Lab 1 and verify that our sample solution runs properly. A dialog may ask whether you “trust” our website — just confirm that you do, and then you should be able to run the demo and interact with it.

Download Starter Code. The last step before we can start is to download our starter code, which just contains a few simple things to help you get started. Click on the link for “Starter Code” to download a file archive called `Lab1NoClicking.tar` to your `cs134` folder. The files of this archive will then be extracted to a folder in your `cs134` folder called `Lab1NoClicking`. Sometimes the extraction does not happen. In this case, find the `Lab1NoClicking.tar` file in your `cs134` folder and double click on it. This should extract the files. (If asked about updating the software used to extract the downloaded files, say “Not now.”) You can delete the `Lab1NoClicking.tar` file by dragging it to the trash can



icon at the bottom right of the screen. You should rename your `Lab1NoClicking` folder so that it reads `Lab1Freund` where `Freund` is replaced by your last name. To do this, just follow the same instructions for changing the name of the “New Folder” from above. That is, highlight the folder, select “Get Info”, etc. (If you’re working with a partner, please rename the folder to contain both last names.) From now on we’ll refer to this folder as `Lab1NoClicking` since `Lab1YourLastNameNoClicking` is long. Now double-click on the copy of `Lab1NoClicking` in your `cs134` folder to see the contents of the folder.

NotPhotoShop. We’ll get back to this folder later, when it’s time for you to write your first Java program. But first you’ll complete the tutorial on `NotPhotoShop`. First, return to the Finder. You can do this by clicking on the smiling Macintosh icon in your dock. Click on Applications in the list that appears at the left of the window. Scroll down to find `NotPhotoShop.jar`, and double-click on it. Our drawing program will display a large window on your screen that should look like this:



The components in this window are divided into five main units.

The Drawing Window: The large rectangle appearing on the upper portion of the right side of the window is the area in which the drawing you create will appear.

The Mouse Tracker: As you move the mouse in the drawing window, the coordinates of the mouse's position within the window will be displayed next to the labels "x" and "y" that appear immediately below the drawing window. Try it! You will often need the coordinates of screen locations to control the placement of the objects you want to create.

The Object Constructor: The shaded area just below the mouse tracker contains the mechanisms you will need to describe and create the components of a drawing. The layout of the items in this region is designed so that the form of the information you input to create an object within this program will resemble the text you would have to enter to create a similar object if you were actually writing a Java program.

The Method Invocation Buttons: There are a number of methods you can apply to modify or obtain information about any of the graphical objects you create in a program. The collection of buttons below the constructor can be used to apply such methods to any of the objects currently present in the drawing window.

The Output Area: Of course, things can go wrong. If you specify some improper information when trying to create or modify an object, the system will display a brief message in the output area that appears along the left margin of the program's window. This area is also used to display descriptions of object properties requested by pressing certain method invocation buttons.

Creating Graphical Objects

To see how all these components work, let's add an object to the display.

You can begin by specifying the object type. The type of object to be created is controlled by the pop-up menu located just to the right of the word "new" in the constructor area. By default, the program starts with the "FramedRect" item selected in the menu. If you depress the mouse on the box labeled "FramedRect," a list of the other choices will appear. Six options are available:

- FramedRect
- FilledRect
- FramedOval
- FilledOval
- Line
- Text

Using the mouse, select “Line” instead of “FramedRect”. As you do this, notice that the labels on the text entry boxes to the right of the menu change to reflect the kinds of information you need to enter to specify the appearance of a line rather than a rectangle. For a framed rectangle, you need to enter the coordinates of its upper left corner and its width and height. For a line, you need to enter the coordinates of the two endpoints. Use the menu to select other options to see what information they require. To display a piece of text you enter the text to be displayed (although the box for entering the text is rather small) and the coordinates of the upper left corner of the rectangle in which the text should be placed. Ovals require the same input as rectangles; the values that you enter for an oval define rectangular bounds within which the oval is drawn.

To continue, make sure that “FramedRect” is the selected menu item. Fill in the boxes to the right of the menu with values to create a 100x100 rectangle at a location 70 pixels from the top of the drawing area and roughly centered between the left and right boundaries of the drawing area. To determine the actual x value to enter for this rectangle, position the mouse cursor where you estimate the left edge of the rectangle should be and use the “mouse meter” to determine the x coordinate of that point. Enter this value in the box labeled “x” immediately to the right of the menu. Then enter 70 as the “y” coordinate and 100 for both the width and height.

Once all these values have been entered, press the “Construct” button and the rectangle will appear.

Changing Object Attributes

When your rectangle appears, it is “decorated” with small black boxes on the edges and corners, indicating that it is the “selected” graphical object. By default, the most recently constructed object will be selected. When multiple objects are displayed, you can change the selected object by clicking within the object you wish to select.

The buttons below the object construction controls can be used to modify the currently selected object. For example, if your rectangle isn’t positioned quite where you wanted it, you can move it around using the “move” and “moveTo” buttons. To try this out:

- Press the “move” button once.
- Enter the amount to move horizontally in the x box and the amount to move vertically in the y box. Positive values will result in motion to the right and down the screen. If you just want the rectangle to move in one direction, enter 0 as the value for the other direction. If you want to move an object to the left or up, enter a negative value. If you are already pretty happy with your rectangle’s location, enter small values (3 or 4).
- Press “OK” and watch the rectangle move.

If you enter any invalid information, a brief message will be displayed in the output area, and the dialog box will remain on the screen so that you can correct the error. Common errors include entering non-numeric characters in numeric fields or leaving fields blank.

As with object construction, the text shown in the “move” dialog box is designed to resemble the text you would include in a program if you wanted to instruct the computer to move an object.

If you press “moveTo”, you will be asked to specify new coordinates for the origin of the selected object. In the case of a rectangle, the rectangle will be moved so that its upper left corner is placed at the specified point. The “setWidth” and “setHeight” buttons invoke methods that let you change the

dimensions of an object. They display a dialog box in which you must enter the new value to be used for the dimension being changed. It is not possible to set the width or height of a Line. If you try to use `setWidth` or `setHeight` while a Line is the selected object, the program will simply display a complaint in the output area.

There is one other button that allows you to change the appearance of an object after it is created. If you find your black rectangle boring, you can easily change its color by pressing the “setColor” button. To do this:

- Click once on the “setColor” button. A dialog box will appear.
- Select one of the colors listed in the dialog box (we like orange) by clicking on its name, and then click the “OK” button.

The dialog that appears when you use the “setColor” button does not attempt to present a display that mimics what you would use to change the color of an object from within a program. The conversion, however, is simple. The items within the dialog box are each a color name preceded by the word “Color”. If “selected” is the name of an object whose color you want to change to yellow, then within your program you would just type

```
selected.setColor(Color.yellow);
```

Later in the semester, we will explain how to select colors other than the small set with which names are associated.

Determining Object Attributes

After you have moved an object up and down a few times to get it right where you want it, you may not actually know where it is. That is, you may not know the exact coordinates of its upper left hand corner or remember its dimensions. Luckily the computer knows and will tell you if you press the correct buttons. The “getLocation” button near the right end of the top row of buttons will display the x and y coordinates of the upper left corner of any object (except a Line). Try it. The coordinates will be displayed in the output area. Similarly, the “getHeight” and “getWidth” buttons can be used to determine the dimensions of such objects.

To determine the endpoints of a Line, you can use the “getEnd” and “getStart” buttons. You can set the endpoints of a Line using the “setEndpoints” button. Finally, you can determine the color of any object by clicking on the “getColor” button.

As with the other buttons, these buttons correspond directly to operations you can perform within a Java program. Within a few days, we will discuss how information returned by these methods can be used within a program.

Working with Multiple Objects

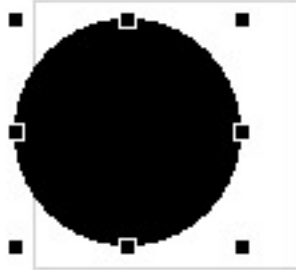
The remaining method invocation buttons are relevant primarily when you are working with a drawing composed of multiple overlapping objects. To demonstrate, let’s create another object.

Create an oval and position it so that it overlaps with your rectangle.

- Select the “FilledOval” choice from the pop-up menu in the object creation area.
- Press the “getLocation” button to determine the current location of the rectangle.
- Enter a value 5 to 10 units smaller than the x coordinate of the rectangle as the x coordinate for the new oval.
- Enter a value 5 to 10 units larger than the y coordinate of the rectangle as the y coordinate for the new oval.
- Set the width and height for the oval to 85 by 85.

- Press “Construct”.

If you entered all the coordinates and dimensions properly, then the image in your drawing area should look something like this:



The new oval is now the selected object. Its bounding rectangle is decorated with eight small black boxes.

You can now use the mouse to change the selected object. Set the color of the oval to magenta by click on “setColor(...)” and choosing magenta from the list. In addition to being selected, a new object (like the oval) is located logically “above” previously created objects (like the rectangle). Notice that where the rectangle’s left edge overlaps the oval, all you see is the magenta interior of the oval, obscuring that part of the rectangle. Since the oval is logically above the rectangle, it hides that part of the edge of the rectangle.

There are buttons in the program’s controls corresponding to two methods you can use within a program to alter the logical stacking of graphical objects. They are labeled “sendBackward” and “sendForward”. To see how they change the stacking:

- Make sure the oval is the selected object.
- Press “sendBackward” once to move the currently selected object (the oval) to the bottom of the stack. Now, the left edge of the rectangle should appear to run through the magenta oval.
- Now press the “sendForward” button to see it move back.

When multiple objects overlap, you may need to use the “send” buttons to change the selected object. If you can’t click on the object you would like to select because it is obscured by another object, first select the other object and use “sendBackward” to move it beneath the object you wish to select. Then click to select the object you desire.

You can think of each object as being drawn on a separate, large sheet of transparent plastic and the image you see is produced by stacking all these sheets of plastic. Changing the stacking is like pulling out the selected sheet of plastic and moving it up or down the stack.

Hiding and Removing Objects

There are two ways to make an object no longer appear in the canvas: hiding the object and removing the object from the canvas. Hiding the object makes it invisible, but a subsequent “show” operation will make it visible once more. Removing the object permanently removes it from the canvas. We discuss only hiding in today’s lab. While both of these options will be available to you when you write programs, only hide and show can be accessed using `NotPhotoShop`.

So, let’s see what happens if we hide the oval. To do this:

- First, make sure that the oval is selected.
- Press the “hide” button. The oval disappears, but it is still the selected object. As a result, while you can not see the oval, you can see its bounding box!

- Click the “show” button. Since this oval is still the selected object, “show” makes it reappear.
- Click “hide” again.
- Now, select the rectangle by clicking somewhere inside the rectangle but outside the oval’s bounding box.
- Try re-selecting the oval by clicking in its previous location. You can select a hidden object as long as it is not hidden under some other object. Once selected, you can make it visible again by clicking “show”.

Draw Your Sign


You are now ready to construct a rendition of the no-clicking sign shown at the beginning of this document. The interior of the sign should be colored yellow like a good warning sign. To produce such a sign you will use a `FilledRect` for the sign’s background and a `FramedRect` for the edge. If you run into problems, ask your instructor or a teaching assistant for help.

When you are done, **DO NOT QUIT THE DRAWING PROGRAM!** You will need the picture you have created for the next task.

Part 2. Writing a Java Program

Now, you will construct a Java program to draw the warning sign you have constructed. You will do this using a program named BlueJ which provides everything you need to type in and run Java programs.

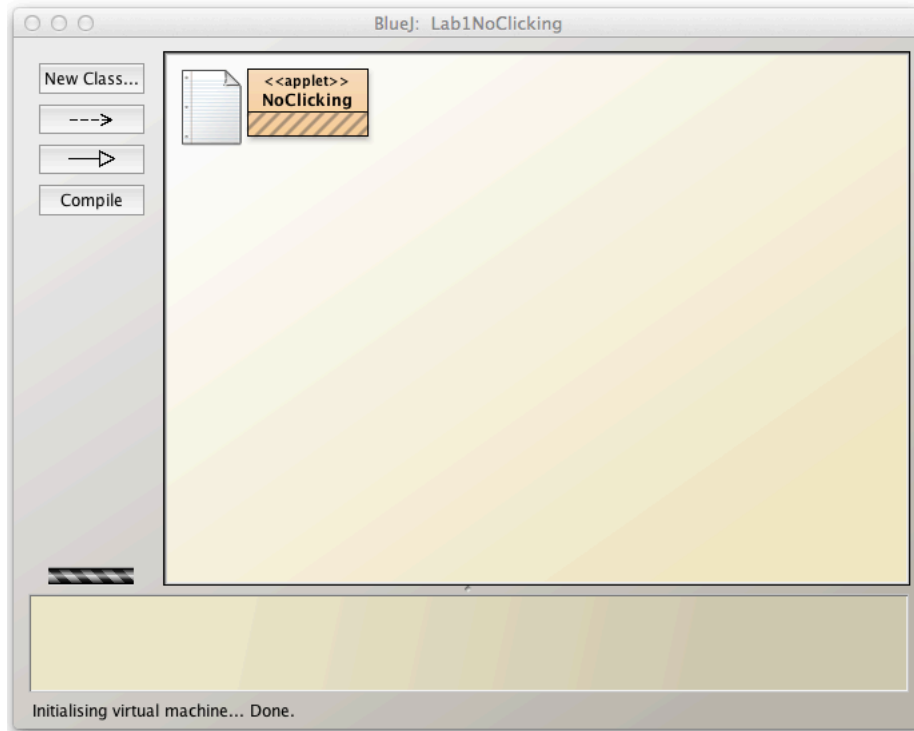
First, return to the Finder, which you can do by clicking on the smiling Macintosh icon in your dock. Click on Applications in the list that appears at the left of the window. Scroll down to find the BlueJ

icon . Because you will be using BlueJ throughout the semester, it makes sense to add it to the dock. To do this, drag the BlueJ icon and drop it on the dock. You should see the other icons in the dock move aside to make room for BlueJ.

Now you can start BlueJ by clicking on the BlueJ icon in the dock. If this is not your first computer science course at Williams (i.e., if you already had a computer science account before this semester), BlueJ will likely ask you if you permit data collection on your use of BlueJ. We suggest you decline.

Once the BlueJ window has appeared, open the “Project” menu and select “Open Project...”. Select your “cs134” folder and click “open”. You should see the “Lab1NoClicking” project. Select it and then click “open”.

At this point your BlueJ window should look similar to the image shown below:



This window has two main regions. The bottom area is called the *instance area*. It displays running instances of your Java programs. For now you can ignore this area. The top area displays the names of the Java classes in the project. When there are multiple classes in a project it also displays the relationships of these classes to one another. Each class is really a file. For example, if you double-click on the `NoClicking` class, a text-editor will pop-up with the contents of the “`NoClicking.java`” file displayed within it. Try double-clicking on the `NoClicking` class. The text of the program as it should initially appear is shown below:

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

/**
 * CS 134 Laboratory 1 / Section X
 * YOUR NAME
 * DATE
 */
public class NoClicking extends WindowController {

    public void onMouseClick(Location point) {
        // commands placed here are executed after the user clicks the mouse
    }

    public void begin() {
        // commands placed here are executed when the program begins to execute
    }

    public void onMouseEnter(Location point) {
        // commands placed here are executed when the mouse enters the program window
    }

    public void onMouseExit(Location point) {
```

```

    // commands placed here are executed when the mouse leaves the program window
}

public void onMousePress(Location point) {
    // commands placed here are executed when the mouse button is depressed
}

public void onMouseRelease(Location point) {
    // commands placed here are executed when the mouse button is released
}
}

```

The text we have placed in the “NoClicking.java” file is the skeleton of a complete Java program. It includes the header for the definition of a class `NoClicking` that extends `WindowController` and within the class, headers for the event-handling methods you will use. We have not, however, included any Java commands within the bodies of these methods, only a Java comment that reminds you when the Java system will follow any instructions you might add to the method body.

You should follow our lead and begin by typing comments rather than actual Java commands. Near the top of the file we have included a temporary comment telling you to enter your name, lab section and the current date. Such identifying comments are always good practice, and in a class like this they make the grader’s job much easier. BlueJ provides the familiar capabilities of a typical word processor or text editor. You can position the cursor using the mouse or arrow keys. You can enter text and/or cut and paste text using items in the edit menu. Use these features to replace our comment with your name, etc.

Next, you will add Java instructions to the body of the `onMouseClicked` method that will draw a “no clicking” warning sign. This is the first method skeleton we have included in the `NoClicking` class.

As a first step, let’s add the single instruction needed to draw the rectangle that frames the contents of the sign. The form of the command you will need to enter is:

```
new FramedRect(..., ..., ..., ..., canvas);
```

where all the ...’s need to be replaced by the numbers describing the position and size of the rectangle. This line should be placed immediately after the comment line in `onMouseClicked` (i.e., just before the line containing the “}” that ends the method body).

If you remember all the values that should replace our dots, just type them in where we have shown the dots. Otherwise, you can return to the `NotPhotoShop` program you used in the first part of the lab to determine the values:

- Switch back to `NotPhotoShop` by either clicking any portion of its window that may still be visible on your screen, or by clicking once on its icon in the dock. (Its icon includes an image of a coffee cup. The word `java` will appear above the icon when you first point at it.)
- Select the framing rectangle.
- Press the “getLocation”, “getWidth”, and “getHeight” buttons. The coordinates and dimensions of the rectangle will be displayed in the output area.
- Return to BlueJ by clicking on its window or its icon in the dock.

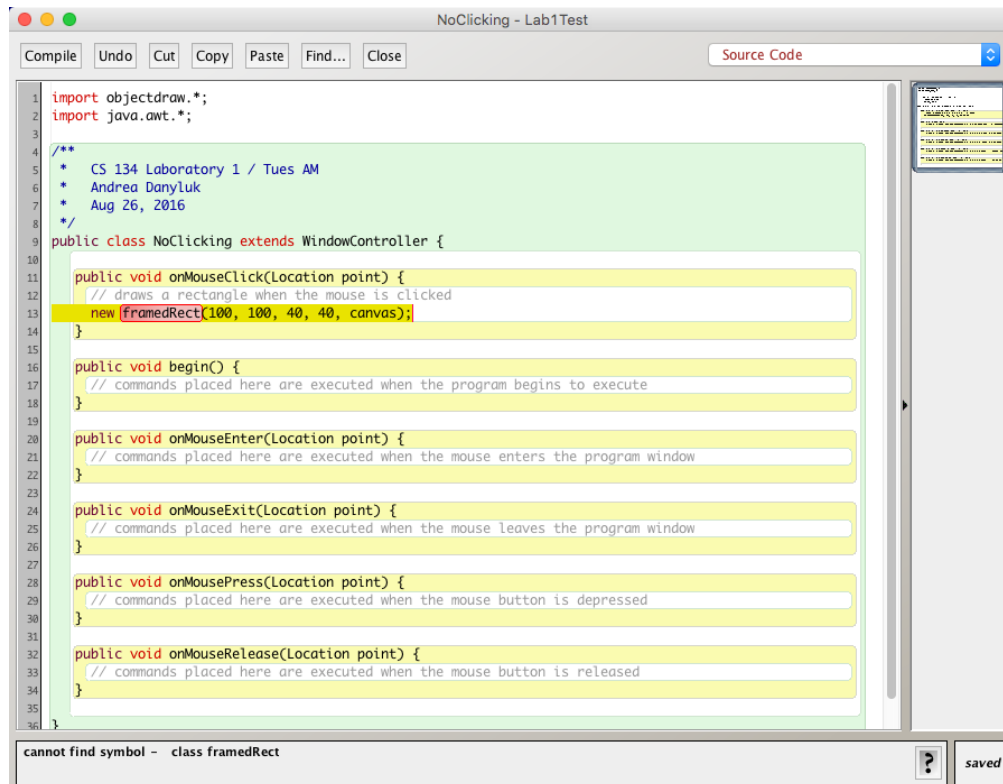
Use BlueJ to add a new `FramedRect(...)` command to your program with the appropriate values. If you have any problems with this step, ask an instructor or TA for help.

Your rectangle constructor should now be below the comment in the `onMouseClicked` method. Now that it does something, it would be a good idea to update the comment in the `onMouseClicked` method to say what it does so far. Replace our comment with something more appropriate, like “draws a rectangle when the mouse is clicked.” You should get in the habit of updating comments to keep them as accurate as possible as you add instructions to your programs. Not only does this mean you will not have to go

back and add comments when you're done, it will help you to remember what everything does as you are writing your programs.

Save your program by selecting “Save” from the “Class” menu. We will now compile the program. To do this, select the “Tools” menu and then “Compile”. You may find yourself compiling often. To save time, BlueJ provides a shortcut. Hold down the propeller key (it's to the left of the space bar) and type *k*.

Compiling may catch some errors in your program. For example, maybe you forgot to capitalize the first letter of `FramedRect`. Fortunately, BlueJ highlights the problematic line of code in yellow and displays a specific error message. In the example above, the error message is *cannot find symbol – class framedRect*. This example is depicted visually below.



You should fix any problems that BlueJ finds with your code, save your class file, and recompile. If you recompile without saving, BlueJ will automatically save the file for you. Once all the problems are fixed, you can run your program. To do this,

- Return to the project window. This is the window with the rectangles representing your classes.
- Hold down the control key and click on the “NoClicking” rectangle.
- You should see a menu. Select “Run Controller”. It should be the final item in the menu. Do not select “Run Applet”.

You should see a window appear with the title “panel0”. This is your running *NoClicking* program. Click on the window to bring it into focus. It should appear blank at first. Move the mouse into the window and click. A rectangle should appear because Java invokes your `onClick` method, which includes the instruction to construct a rectangle. If this happens, smile. If not, select “Quit BlueJ Virtual Machine” from the “BlueJ Virtual Machine” menu to leave your program. This will bring you back to the BlueJ project window. Examine the instructions you typed carefully to see if you can find any mistakes. If so, correct them, recompile and run your program again. Otherwise, ask your instructor or a TA for assistance.

When you ask BlueJ to “Run” your program, it starts up a new program that is separate from BlueJ. This is the BlueJ Virtual Machine. While this program is running, you can return to BlueJ by just clicking on any portion of the BlueJ window that is visible on your screen. You should also see a red rectangle in the *instance area* of the project window. This is because by running the program you created a new instance of the `NoClicking` class. This instance represents the running program. To avoid confusion, it is a good idea to quit your program by selecting “Quit BlueJ Virtual Machine” before asking BlueJ to run your program again (via the “Run Controller” item).

Once your rectangle program is working, you should add more instructions to turn it into a complete warning sign drawing program. Back in BlueJ, immediately beneath the line you added to draw the rectangle, add additional lines to create new `Text(...)`, a new `FramedOval(...)` and all the other components you created using our `NotPhotoShop` program. In the `NotPhotoShop` version of the drawing, we told you to have a yellow background for the sign. Leave that part out of the version your program draws for now; we will add it later. Again, you may want to refer back to `NotPhotoShop`’s “getLocation” and related method buttons to help you figure out the coordinates and dimensions to use for each object. As you work, it is a good idea to compile and run your program every time you add a line or two to ensure that you catch mistakes early. Also, make sure your comment line gets updated by the time you are finished. When you are done, your program should draw a sign when you click in the window, and then do nothing until you tell it to quit. In reality, the program draws another copy of the sign each time you click, but you can’t tell, as each new sign is drawn exactly on top of the previous one.

Making Your Program Responsive

Now, to explore event-handling methods a bit more, revise your program so that it reacts to the mouse in more interesting ways. In the revised version, the sign will be drawn as soon as the program begins to run. The sign will change, however, as the program runs. In particular, when the user moves the mouse into the program window or presses the mouse, your revised code will alter the appearance of the sign.

Making the program draw the sign when it first starts is easy. You initially placed the code to draw the sign in the `onMouseClicked` method. There is a skeleton for a method named `begin` immediately after the `onMouseClicked` method definition in the `NoClicking` class. As you might guess, the instructions in `begin` are followed when your program first begins to run. Use cut and paste to move all the drawing instructions from the `onMouseClicked` method into the `begin` method. Don’t forget to update your comments. Run your program again. It should display the sign immediately.

Notice that it makes more sense logically to have the `begin` method occur before the `onMouseClicked` method. We arranged them in the opposite order so that you would be able to find `onMouseClicked` more easily. The point is that the computer does not care which order you write them in your program. Only the names are important (though we urge you to put them in a logical order so that it will be easier for all of us to read!). The order of instructions within a method are important, but the order in which you define the methods within the program is not.

Making the sign change in response to the mouse is a bit trickier. Suppose, for example, that you want to emphasize the sign’s warning by changing the color of the word “CLICKING” from black to red when the user moves the mouse into the program’s window. As you saw in `NotPhotoShop`, there is a `setColor` operation that you can use to change the color of the text. There is also an obvious place to tell Java to make this change. The `onMouseEnter` method is executed whenever the mouse moves into your program window. Placing an appropriate `setColor` in that method would do the trick.

The problem is that you can’t simply say `setColor` in the `onMouseEnter` method. If that was all you said, Java would have no way of knowing which object’s color to change. It could change the rectangle, the oval, the text, all of them, etc. Your code has to be more specific and identify the object that should change.

To be able to specify the text object as the object whose color we wish to set, we will have to give it a name. We will use the name `message`, but we could use any name that seems appropriate.

Associating a name with an object requires two steps. First, we have to include a line that declares or introduces the name. This line informs Java that we plan to use a particular name within the

program. At this point, we don't need to tell Java which object it should be associated with, but we do need to specify which sort of object it will eventually be associated with. We plan to associate the name `message` with an object created as `new Text(...)`, so we have to tell Java that the name will be associated with a `Text` object. The form of a Java declaration is simply the name being declared preceded by the keyword `private` and by the type of object with which it will be associated. So, the form for our declaration is:

```
private Text message;
```

You should add a line containing this declaration to your program immediately *before* the heading of the `onMouseClicked` method.

Now you have to tell Java which object to associate with the name `message`. Your program currently contains a construction of the form:

```
new Text("CLICKING", ..., canvas);
```

that creates the text on the screen. To associate the name `message` with the text object this line creates, revise this line so that it looks like:

```
message = new Text("CLICKING", ..., canvas);
```

This is an example of a construct called an *assignment statement*. It tells Java that in addition to creating the new object, it should associate a name with it. Shortly, you will convert some of your other constructors into assignments.

Now that the text has a name, we can use the name to change its color. Within the body of the `onMouseEnter` method add the line:

```
message.setColor(Color.red);
```

Then, compile and run your program (correcting any errors as needed).

The program isn't quite complete. It draws the sign immediately and makes the text turn red when the mouse enters the window, but it doesn't make the text black again when the mouse is moved back out of the window. You should be able to figure out what to add to make it black when the mouse exits. Give it a try.

To get more practice using names and other event handling methods, we would like you to modify your program a bit more. First, change the program so that while the user is depressing the mouse button, the circle with the diagonal line through the text disappears. (Of course, it should reappear when the mouse is released.) This will require that you declare names for the circle and the line and associate them with the correct objects. These declarations should appear right after the declaration of `message`. Your code to make the objects disappear and reappear goes in the `onMousePress` and `onMouseRelease` methods. You can use the `hide` and `show` methods to handle the disappearing and reappearing.

Finally, add the yellow background to the sign. We didn't have you do this earlier because you had not yet seen how to associate names with objects. Associate a name with the background rectangle when you create it and then use the name to set its color to yellow. Think carefully about where to put this code so that the shapes are stacked in the right order.

Grading Guidelines

Both functionality and programming style are important when writing code, just as both the content and the writing style are important when writing an essay. In this program, the some of the specific functional requirements we will test for include:

- Drawing the sign when the program starts.
- Changing the color of the text.
- Making the slash and circle disappear and reappear.

Stylistically, we expect to see programs that exhibit the following:

- Meaningful names used in declarations.
- Informative comments.
- Good and consistent formatting.
- Good choice of Java commands.

There is some subjectivity to what makes good style, but the basic goal is to make your ideas as clear and easy to follow as possible.

Programming labs will be graded on the following scale:

- A+* An absolutely fantastic submission of the sort that will only come along a few times during the semester.
- A* A submission that exceeds our standard expectation for the assignment. The program must reflect additional work beyond the requirements or get the job done in a particularly elegant way.
- A-* A submission that satisfies all the requirements for the assignment — a job well done.
- B+* A submission that meets the requirements for the assignment, possibly with a few small problems.
- B* A submission that has problems serious enough to fall short of the requirements for the assignment.
- C* A submission that has extremely serious problems, but nonetheless shows some effort and understanding.
- D* A submission that shows little effort and does not represent passing work.

Submitting Your Work

Congratulations, you have written a Java program! You are encouraged to continue experimenting with it. Before you do, however, you should submit it as a completed assignment. The submission procedure is electronic and will be basically the same every week.

Once you have saved your work in BlueJ, please perform the following steps to submit your assignment:

- First, return to the Finder. You can do this by clicking on the smiling Macintosh icon in your dock.
- From the “Go” menu at the top of the screen, select “Connect to Server...”.
- For the server address, type in “afp://Guest@fuji” and click “Connect”.
- A selection box should appear. Select “Courses” and click “Ok”.
- You should now see a Finder window with a “cs134” folder. Open this folder.
- You should now see the drop-off folders for the three lab sections. Drag your “Lab1NoClicking” folder into the appropriate lab section folder. When you do this, the Mac will warn you that you will not be able to look at this folder. That is fine. Just click “OK”.
- Log off of the computer before you leave.

(Don’t worry about remembering how to find this folder— all the lab handouts will remind you of how to find it.)

You can submit your work up to 11 p.m. on Wednesday if you’re in the Monday night lab; up to 6 p.m. on Thursday if you’re in the Tuesday morning lab; and up to 11 p.m. on Thursday if you’re in

the Tuesday afternoon lab. If you submit and later discover that your submission was flawed, you can submit again. We will grade the latest submission made before your lab deadline. The Mac will not let you submit again unless you change the name of your folder slightly. It does this to prevent another student from accidentally overwriting one of your submissions. Just add something to the folder name (like the word “revised”) and the re-submission will work fine.