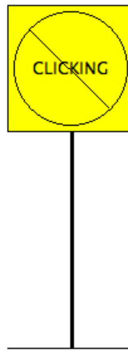
Lab 1
A “No CLICKING” Sign[1]
CSCI 1101B – Spring 2015

Due: February 3, 10 pm

Objective: To gain familiarity with `objectdraw` graphics capabilities and simple Java programming.

This lab will continue to introduce you to many of the tools with which you will be working during this course. Your task is fairly simple. You will construct a Java program that draws an image resembling a roadside warning sign, but with a message more appropriate for a computer screen, as shown below. The image will change in various ways depending on mouse activity, as detailed later in this handout.



Before you start, create a new `objectdraw` project by following the directions in the handout “Using the `objectdraw` library with BlueJ.”

Writing a Java Program

If you haven't already, open up your project in BlueJ by double-clicking on the `package.bluej` file. Then double click on the box labeled “Events” to open up a code editor window showing the contents of `Events.java`. The existing text in the Events file (or `Events` *class*) is the skeleton of a complete `objectdraw` Java program, including the `begin` method and all possible mouse event methods. We have not, of course, included any Java commands within the bodies of the methods, only a Java comment that reminds you when the Java system will follow any instructions you might add to the method body. Note that you will not need all these mouse event methods. They're included in the `Events` class so they're available for future programs, **but please delete all the methods that you do not use in your program**.

You should begin by filling in the comments near the top of the `Events.java` file in your project (e.g., name, class, date, etc). Such identifying comments are always good practice in the real world and, in a class like this, they make the grader's job easier. You also need to document

[1]Adapted from a lab provided with *Java: An Eventful Approach*, K. Bruce, A. Danyluk, and T. Murtagh

your code – in particular, you should write a program summary at the top of the class that gives a high-level description of the behavior of the program, and you should also write comments interspersed throughout the code explaining what the various sections code are doing. **Comments are critical both to help anyone reading your program as well as to remind you of code you previously wrote.** The program summary at the top should briefly describe how the program works, e.g. "Draws a yellow, rectangular traffic sign labeled CLICKING with a circle around the label and a slash through the label. When the mouse button is depressed...[you finish this!]". Since we have not yet specified how the image will change, you should return to these comments afterwards to complete them.

Your task is to complete the code in the `Events` class to draw the "no clicking" warning sign and then make it response to the user in various ways. First, you will add Java instructions to the body of the `begin` method that will draw the starting sign. As a first step, let's specify the size of the window our program will draw in. Java will create a window of default size for your program to draw in, but you can specify the window size by adding a command like this as the first command within the `begin` method:

```
resize(400, 450);
```

This line should replace the comment line that is currently in the `begin` method:

```
// executed when the program begins
```

Additionally, the command should be indented the same amount as the comment was indented. The first number in the `resize` command is the width of the window (in pixels), and the second number is the height. Thus, the above command will set the size of the window to 400 pixels wide and 450 pixels high (feel free to adjust these values). You can also change the size of the graphics window while your program is running by dragging the lower right corner of the window.

Now 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 (x, y, width, height) of the rectangle. These values (and the `canvas`), i.e. the values that are specified when the method is called, are referred to as "arguments." Decide (roughly) what the position and size values should be and type the command in.

Now that your program does something, you should add a comment above the command you added to say what it does, for example:

```
// draws a rectangle when the program begins
```

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. However, you do **not** need to, nor should you, comment every command you write. If there are several lines that logically belong together, e.g. the commands that construct the yellow

part of the sign with its black outline, it would be better to put in a single comment above the lines that do this briefly explaining what they do. This means that you might have to go back after your program is finished and edit your comments. Remember – comments are there to enhance the readability of your program. Every comment you write should give some extra information about the program that may not be immediately obvious from the code itself. If a line or brief section of code is completely self-explanatory (which is a good thing!), no comment for that code is necessary. Learning when to appropriately comment will become easier with time and experience!

Compiling and Running Your Program

Now that you've changed your program, compile it by clicking "Compile" in the upper left of the editor window. If you made a syntax mistake in your program, BlueJ will highlight the error in the editor when you try to compile your program. Checks that BlueJ will perform include making sure that each of the library methods has the appropriate number and type of arguments, method and command names are spelled correctly, punctuation in your program is correct, etc. When an error line is highlighted, BlueJ will display a descriptive error message at the bottom of the editor window. Take your time when you receive error messages and read the messages carefully. They often provide the information you need to fix your program. Unfortunately, messages associated with errors are not always easy to understand for beginners (or sometimes even for experienced programmers). Understanding these messages will get easier as the semester progresses. If you don't understand any of them, ask me, your lab instructor, or one of the TAs or QR mentors.

Once any problems with the `new FramedRect(...)` command are fixed, your program will compile without errors. Go ahead and run your program by right-clicking or control-clicking on the box labeled "Events", then selecting "new Events()". Click Ok on the window that appears to start running your program. When your program runs, a new graphics window will appear on your screen. A rectangle should appear because Java invokes your `begin` method, which includes the instruction to construct a rectangle. You'll probably need to play around with the argument values you filled into the `FramedRect` command to get the rectangle in a good place and a decent size. Remember that each time you change your program, you'll need to recompile it and then re-run it to see the effects of your changes.

Once your rectangle program is working, you should add more instructions to it to turn it into a complete warning sign drawing program. Immediately beneath the line you added to draw the rectangle, add additional method calls to create a `new FilledRect(...)` (for the sign post), a `new FramedOval(...)` (to circle the text of the sign), `new Text(...)`, and all the other components you need for the sign. Note two things:

1. The post of the sign should be a `FilledRect` (not a `Line`), and the base should be a `Line`.
2. The sign should have a yellow background, but leave that for the next section.

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 lines get updated by the time you are finished. When you are done, your program should draw a sign when run, and then do nothing.

Making Your Program Responsive

Now, to explore event handling methods a bit, revise your program so that it reacts to the mouse in more interesting ways as the program runs. In particular, when the user moves the mouse into the program window or presses the mouse, your code will alter the appearance of the sign.

First, we 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. 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 of the several objects' 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 this line to your program immediately *before* the heading of the `begin` method (i.e., this declaration will be *inside* the class body, but *outside* any method body).

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 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 constructor calls 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, 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. Go ahead and do that now.

To get more practice using names and other event handling methods, you should 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. I didn't have you do this earlier because you had not yet seen how to associate names with objects. To create the background, use a `FilledRect` in the same area as the previous `FramedRect` you created for the sign. Associate a name with this background rectangle when you create it and then use the name to set its color to yellow. Note that to prevent the sign background from looking like it's covering the outline, be sure you create the background first and then the outline.

Submitting Your Work

Once your program is finished, you should follow the following steps to submit:

1. Save your program and quit BlueJ (this is necessary because BlueJ gets confused if you perform step 2 – renaming your project directory – while the project is open).

2. Rename your project folder (which is the folder that contains `Events.java`, `package.bluej`, and possibly a few other files) so that it is named `username-lab1` (with your actual username). For example, I would rename my folder `sbarker-lab1`.

3. Create a single, compressed `.zip` archive of your project folder. On a Mac, right-click (or, if you have no right mouse button, control-click) on your project folder and select "Compress your-folder-name" from the menu that appears. On a Windows machine, right-click on the folder, select "Send To," and then select "Compressed (zipped) Folder." In either case, you should now have a `.zip` file that contains your project, named something like `sbarker-lab1.zip` (with your actual username).

4. Open a web browser and go to Blackboard, then browse to the `Lab Submissions` folder in your appropriate lab section. Click on `Lab 1` and then `Start New Submission`. In Section 2, you can, but do not need to, provide any comments. Then select `Browse My Computer` and browse to the `.zip` file you created in step 3. Select that file, then click on `Submit`.

You're done submitting your lab, but remember to save a copy of your project folder somewhere other than on the desktop of the machine you are working on. If you just leave it on the desktop, it will only be available on that machine—if you log into any other machine on campus, it will not be there. I would suggest that you keep a `cs1101-labs` folder containing your labs, the `objectdraw` library (the file `objectdraw.jar`), and the starter project in Dropbox (or any similar service) or in your folder on the `microwave` server (see below).

Connecting to the `microwave` server

All students have storage space available on the `microwave` file server, which you can use to store your labs. The benefit of using this server is that you can access your `microwave` folder from any machine on campus, not just the specific lab machine on which you were last working.

To connect to `microwave` on any lab Mac, click on the Desktop, then select "Connect to Server" under the Go menu (or just command-K). In the window that appears, type the following (substituting your actual username for the word `username`):

```
smb://microwave.bowdoin.edu/home/username
```

Click Connect and enter your Bowdoin password when prompted. A folder should then appear on the desktop named with your username. This is your `microwave` folder, in which you can store your labs when you are finished with them (or leaving but returning to work later). However, you should not work directly on files on microwave. While you are actively writing your program, you should copy your project folder to a "local" area first, such as the desktop. Once you are finished working, you can copy your files back to microwave.