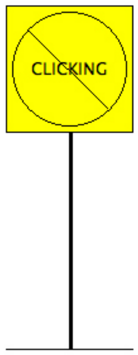
CS Programming Exercise:
An Introduction to Java and the ObjectDraw Library

Objective: To demonstrate the use of objectdraw graphics primitives and Java programming tools

This lab will 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.



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.

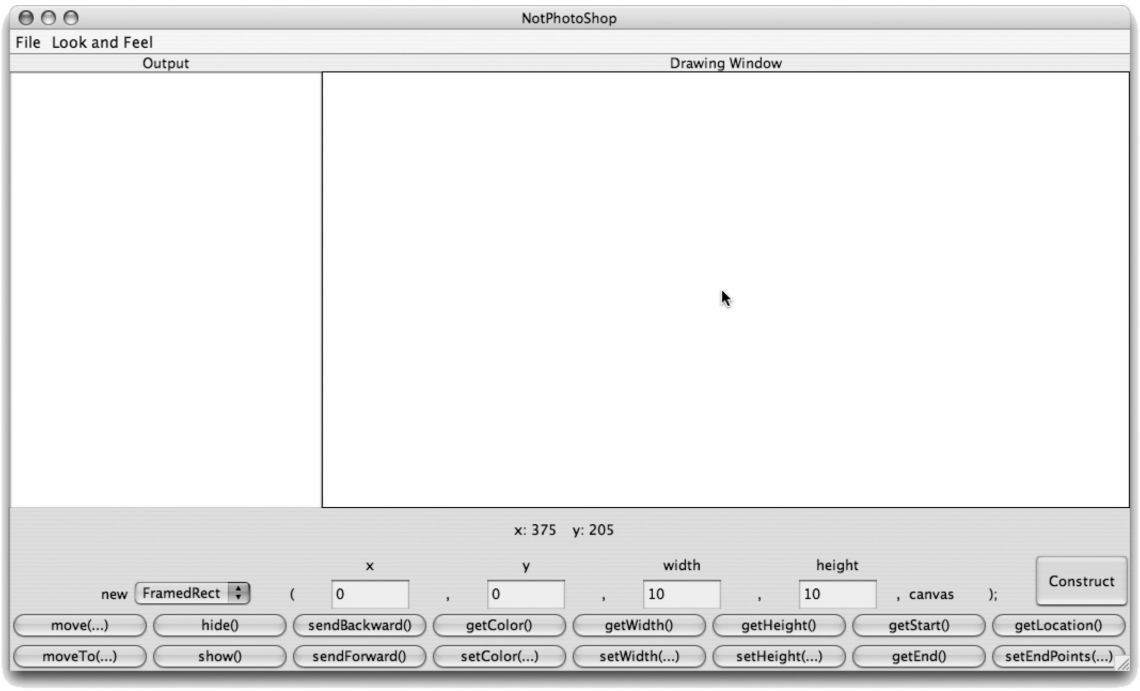
Part 1. An Interactive Introduction to the ObjectDraw Primitives

To help you learn the primitives for creating graphical objects from within a program before dealing with the details of writing a program, we have constructed an environment in which you can experiment interactively with the graphics primitives. This environment is a program that 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 an image editing program (like Adobe Photoshop) you will quickly conclude it is the worst image editing 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.

To start, you must log in to one of the computers in the lab. Your instructor will provide you with information on how to log in and download the files for this laboratory to the computer you are using.

When you have completed downloading the files, you should have a folder on your computer called `introStarter`. Double-click on the folder to see the contents. You will see several items related to today's lab. The first one you will use is called "NotPhotoShop" (or "NotPhotoShop.jar"). To run NotPhotoShop, double-click on the "NotPhotoShop" icon in your introStarter folder. Our drawing program will display a large window on your screen that should look something like the following:



x: 375 y: 205

	x		y		width		height	
new FramedRect (	0	,	0	,	10	,	10	, canvas);

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 about 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 two points — its end points. 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 rectanglular 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 100 $\times$ 100 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 tracker” 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 corner, 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 only 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. Of course, 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 (I like orange) by clicking on its name and then click the "OK" button.

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.

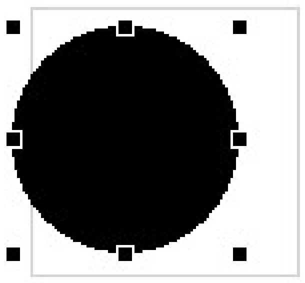
Working with Multiple Objects

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.

- Move the mouse so that it is within the original rectangle but outside the bounding box of the oval,
- Click the mouse once. The eight small boxes now move to the edges of your original rectangle, indicating that it is selected.
- Move the mouse back into the center of the oval and click again to make it 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 black 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 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. This stacking happens even if the objects do not overlap on the screen. Try this:

- Use the mouse and the "send" buttons to ensure that the oval is on top of the rectangle and selected.
- Use the "move" button to move the oval right 150 units (enter 0 for the change in the y coordinate).
- Now, press the "sendBackward" button. Nothing in the display will appear to change.
- Move the oval left 150 units, to its original position. You should discover that it is now below the rectangle in the stack.
- Finally, press the "sendForward" button to restore the oval to the top layer.

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. (You can't tell this looking at the black-and-white printed version of this writeup, but you can do it on your computer's screen.) If you run into problems, ask your instructor 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. The construction of a single Java program involves the use of several files. One file will contain Java code, the actual text of your program. Another file is required to tell the Java system run time details like the size of the window your program should be given. In addition, access to many of the facilities of Java and our graphics system are provided through other files, including "objectdraw.jar".

Your instructor will provide you with instructions on how to create a Java program and the local tools for developing and running programs. You will create or download some files, one of which will be a file named "NoClicking.java", which will contain the text of your first Java program.

The text of the program as it should initially appear is shown on the next page. The text we have placed in "NoClicking.java" 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. Your development environment will provide the familiar capabilities of a typical

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

//   Programming Exercise
//   Enter your name, class and lab section, and the date here.

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
    }

}
```

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 `onMouseClick` method that will draw a "no clicking" warning sign. This is the first method skeleton we have included in "NoClicking.java".

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 `onMouseClick` (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 on its icon in the dock.

- 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 your development environment.

Add a `new FramedRect` command to your program with the appropriate values.

Some development environments report errors as you are typing in a program, while others report errors only when the program is compiled. Unfortunately, some of the messages associated with errors are not easy to understand for beginners. Understanding these messages will get easier as the semester progresses.

Take your time when you receive error messages. First, errors often occur simply because you are in the middle of doing something and these errors go away when you complete the task. So, it is generally a good idea to ignore the warnings that come up while you are editing. Second, read the messages carefully. They often have good advice. (But not always! Computers are not actually very smart!) Third, if your development environment provides the option of selecting a change to be applied automatically, make sure you understand its suggestion. If you don't understand it, it's probably not the right thing to do! Ask an instructor or TA for help.

Your rectangle constructor should now be below the comment in the `onMouseClick` method. Now that it does something, it would be a good idea to update the comment in the `onMouseClick` 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.

Before running your program, be sure to save it. Once it is saved you must compile it and run it. Some development environments invoke these two steps with one "run" command. See your instructor for the details of how to do this in your local environment.

When your program is compiled, it checks to make sure that your program is syntactically correct. This involves making sure that each of the library methods has the appropriate number and type of arguments, making sure the method and command names are spelled correctly, making sure the punctuation in you program is correct, etc.

If the compiler finds any problems, it will display a list of the errors along with a short description that includes the line number and a brief explanation of each. If the problem list is not empty, you must fix the problems and recompile until there are no errors reported.

Once all the problems are fixed, you can run your program. The details of how this is done depend on your local environment. Your instructor will provide you with information on how to do this. In doing this, be sure to record in the appropriate on-line file that the window should be 700 pixels wide and 450 pixels high.

When your program runs, a new window will appear on your screen. This is your program's window. It should appear blank at first. Move the mouse into the window and click, and a rectangle should appear because Java invokes your `onMouseClick` method, which includes the instruction to construct a rectangle. If this happens, smile. If not, select quit, and return to your development system. Examine the instructions you typed carefully to see if you can find any mistakes. If so, correct them and run your program again. Otherwise, ask your instructor or a TA for assistance.

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 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 "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 `onMouseClick` method. There is a skeleton for a method named `begin` immediately after the `onMouseClick` method definition in the "NoClicking.java" file. 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 `onMouseClick` method into the `begin` method. Don't forget to update your comments. Run your program again. It should now display the sign even if you don't click the mouse.

Notice that it makes more sense logically to have the `begin` method occur before the `onMouseClick` method. We arranged them in the opposite order so that you would be able to find `onMouseClick` 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 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 a line containing this declaration to your program immediately *before* the heading of the `onMouseClick` 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, 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.

Submitting Your Work

Congratulations, you have written a Java program! You are encouraged to continue experimenting with it. When you are done, please submit the program according to the directions provided by your instructor.