



## CS 101 - Problem Solving and Object-Oriented Programming

### Lab 5 - Draw a Penguin

Due: October 28/29

#### Pre-lab Preparation

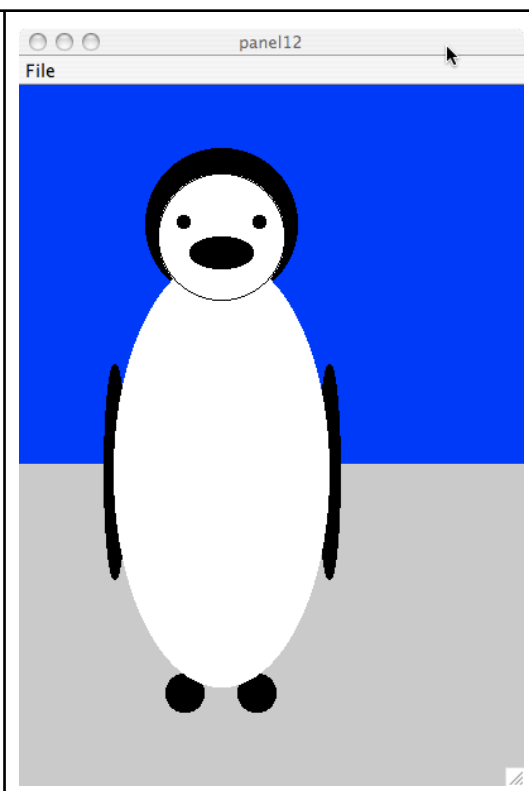
Before coming to lab, you are expected to have:

- Read Bruce chapters 1-3

#### Introduction to the Assignment

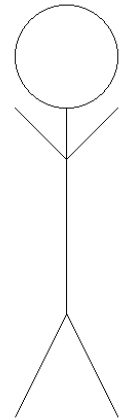
In this lab, you will create a very specialized drawing program. The only thing you can draw with this program is a penguin! When the program starts, it will display a penguin standing on (gray) ice with a bright blue sky.

The drawing that the user sees when the program starts.



## Drawing a Penguin

Before even thinking about writing this program, you should draw a penguin by hand (or closely examine the drawing above). You will use your drawing to figure out how big the parts of the penguin should be and where they should be located. As we saw with the stick figure in class, it is best to use constant values to define the size and location of just one of the shapes that makes up the penguin. Then, define the sizes and locations of the other shapes relative to one that is already defined. For example, our stick figure looked like this:



We defined the size, left and top of the stick figure's head as constants and then calculated the sizes and locations of everything else relative to that. That way if we decide to make the stick figure smaller, for example, we can change the head's size and the sizes of everything else will be suitably scaled. As a reminder, here is what the Java code looks like that calculates the size and location of the body based on the head's size and location.

```
private static final int HEAD_LEFT = 50;
private static final int HEAD_TOP = 50;
private static final int HEAD_SIZE = 100;

private static final int BODY_LEFT = HEAD_LEFT + (HEAD_SIZE / 2);
private static final int BODY_TOP = HEAD_TOP + HEAD_SIZE;
private static final int BODY_SIZE = HEAD_SIZE * 2;

private static final int ARM_Y_OFFSET = HEAD_SIZE / 2;
private static final int ARM_BOTTOM = BODY_TOP + ARM_Y_OFFSET;
```

Before reading on, make a list of all the parts of the penguin. Then pick a fixed size and location for one part and write equations you could use to calculate the sizes and locations of the other parts. Notice that an equation can use any constant that is defined earlier. For example, the ARM\_BOTTOM of the stick figure shown above does not use any of the head constants. Use whatever seems most logical. The only constraint is that an equation can only use constants defined earlier in the class. Don't worry if you don't get it right the first time. In fact, it would be surprising if you did. I surely didn't! (It looked more like Picasso Penguin my first time!) Fill in the following table with this information.

Part	Width	Height	Left	Top

Part	Width	Height	Left	Top

### Pair Programming

In our Java labs, we will use a technique called **pair programming**. In pair programming, 2 programmers work together. Both programmers work on the same computer. One programmer does the typing. In pair programming, there is expected to be a lot of discussion. Our labs should no longer be quiet!

Typically, the 2 programmers discuss how to solve a problem before typing anything. As the typist enters the program, the other programmer should look at it with a critical eye. Is the code being typed going to do what they want it to? Again, more discussion should ensue about the exact way to say what you want in Java. The program should not have "her part" and "my part". The entire program is owned by both students.

Don't be afraid to point out mistakes but be polite doing so. Don't be afraid to say that you don't understand something. That is how you learn. It might even be the case that the typist is the one who is confused. Studies have shown that even very experienced programmers are more productive and more likely to produce correct code using this programming technique! There are many mistakes that even very experienced programmers make but that a second pair of eyes can easily find.

As the typist, take criticism well. Be an "ego-less" programmer!

During lab time, we will be working in pairs. After the lab period is over, you may choose to continue working with the same partner outside of lab time to complete the lab. Alternatively, you may decide to complete the lab individually. In either case, you should identify your lab partner when you submit the lab. If you never switch to working on the lab individually, you only need to submit one assignment as a team.

Each lab you will be expected to work with a different partner.

It is important that you take turns being the typist. For example, one person should be the typist for step 1, then switch for step 2, etc. It is critical that you each learn the material since ultimately you will be tested individually. Don't let your partner do all the work! It will hurt you in the long run!

## Writing the Program

Create a new BlueJ project for this assignment. In the project, create 2 classes: DrawAPenguin and Penguin. DrawAPenguin is the class that extends WindowController. It is the class in which you will define the begin method and the mouse handling method.

The Penguin class will contain the constants that define the size and location of the penguin parts and a constructor that draws a penguin.

As with Alice labs, the key to success is to divide the problem into small pieces and get each piece working before moving on to the next.

### Step 1: Drawing the background

Recall that the initial display is defined by the begin method. Therefore, you should add a begin method to your DrawAPenguin class.

The first thing you should do is set the size of the window. The window used by the demo of this lab is 400 pixels wide and 600 pixels tall. To get your program to make the window that big, put the following line inside the curly braces of the begin method:

```
resize (400, 600);
```

In the initial display, you should draw a filled rectangle to be the sky and another filled rectangle to be the ice.

To create a filled rectangle, you must call the FilledRect constructor. Recall that the FilledRect constructor has five parameters. Therefore, you must provide five arguments in the following order:

- left coordinate
- top coordinate
- width
- height
- canvas

By default a filled rectangle will be black. To change the color of the rectangle, you must do several things:

- Add "import java.awt.Color;" at the beginning of the file.
- Save the rectangle that represents the sky in a variable.
- Call the setColor method on the sky rectangle, passing the sky color as an argument.

Here are the colors that Java knows by name:

- |             |              |          |
|-------------|--------------|----------|
| • BLACK     | • GREEN      | • RED    |
| • BLUE      | • LIGHT_GRAY | • WHITE  |
| • CYAN      | • MAGENTA    | • YELLOW |
| • DARK_GRAY | • ORANGE     |          |
| • GRAY      | • PINK       |          |

To create a green rectangle that represents grass, you would say:

```
FilledRect grass = new FilledRect (10, 20, 100, 200, canvas).;  
grass.setColor(Color.GREEN);
```

Using this information, you should be able to create a rectangle for the sky and for the ice. Place two statements inside your begin method, one to create each rectangle.

Compile and run your program. If your program works properly, it should look like sky and ice (if you are in a generous mood). If it doesn't, quit your program and go back to BlueJ to edit your class.

## Step 2: Constructing a penguin

To draw a penguin, you need to create a class for the penguin. You will define a constructor within the Penguin class to draw the penguin. The penguin constructor is declared as follows:

```
public Penguin (DrawingCanvas penguinCanvas)
{

}
```

Before the constructor, you should place the declarations of the constants you will use to draw the penguin. These constants should be defined based on the equations that you created on page 3 and 4 of this lab.

The statements within the Penguin constructor will be responsible for actually drawing a penguin by creating shapes. You will probably find FilledOvals to be most useful. Creating a filled oval is very similar to creating a filled rectangle. You must call the FilledOval constructor. The FilledOval constructor has five parameters. Therefore, you must provide five arguments in the following order:

- left coordinate
- top coordinate
- width
- height
- canvas

For the last argument you should use the constructor parameter named penguinCanvas. Remember also to set the colors of the penguin parts as you create them. You might start just drawing a couple of the penguin's parts instead of trying to do it all at once.

You need to call the penguin constructor from the begin method of the DrawAPenguin class. The constructor call should appear after the construction of the two rectangles that make up the background. The constructor call should look like this:

```
new Penguin (canvas);
```

This tells Java to call the Penguin constructor passing the predefined canvas as the argument. Compile and run your program. If all is well, you should see a penguin! Chances are, though, that things won't line up nicely or be the right size. If so, go back to BlueJ and improve the equations used to produce the sizes and locations of the parts and try again.

Once your program draws a penguin, go back and check that all the equations work properly. To do this, change the value of the literal you used for your first constant to be half the value or twice the value. Think about what you expect to happen. Do you expect the penguin to now be half its original size? Or closer to the left? Compile and run your program. If your equations are correct, the penguin should scale or should be drawn nicely to a different location. If instead, the penguin "comes apart", that is an indication that your equations are not right. Figure out which part(s) are in the wrong place or of the wrong size and fix the corresponding equations.

## Style Issues Comments

Be sure to include comments in your program. In particular, there should be:

- A comment at the beginning of each class, immediately before the “public class” line. This comment should explain the purpose of the class.
- A comment immediately before each constructor and method declaration. These comments should describe what the constructor or method does.
- A comment immediately before each constant declaration. These comments should explain the purpose of the constant.

## Naming

Choose good names for your constants. Names should be descriptive of the purpose of the constant. Also, follow the naming convention of using only UPPERCASE letters in constant names. If a name consists of multiple words, separate the words with an underscore character, as in HEAD\_SIZE.

## Grading

5	Step 1: Drawing the background
20	Step 2: Constructing a penguin
10	Comments
10	Naming
5	Indentation

## 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 DrawAPenguin.java file and Penguin.java file, not the other files created by BlueJ. Click Continue. Then click the Submit button.