COLBY COLLEGE
Computer Science Department

CS151
Fall 1999

Allen B. Downey

Homework 4: Snowpeople!

The purpose of this assignment is to practice using methods as a way of organizing and encapsulating complex tasks.

WARNING: It is very important that you take this assignment one step at a time, and get each step working correctly before you proceed. More importantly, make sure you understand each step before you proceed.

Due: Wednesday 6 October

New stationery!

1. Create a new project named `Snowperson.prj`, using the project stationery named `cs115SlateStationery`. Run it. A new window should appear with a bunch of blue circles and a bunch of red circles. This window is the Slate.

   Unlike the other programs we ran, there is no console window. However, if you add a `print` or `println` statement, the console window will appear. For graphical applications, the console is useful for debugging.

   I found that I could not close the Slate window by clicking on it, which is probably good, because it will remind you to quit from the interpreter every time you run the program.

2. Look over the source code as it currently exists and make sure you understand all the code in `draw`. The method named `anim` is there for your entertainment, but we will not be using it for this assignment. You should remove it.

3. The `Slate` class appears immediately after the `cs151` class. You might want to check it out, although a lot of it will not make sense at this point.

4. Fiddle with the statements in `draw` and see what effect your changes have. Try out the various drawing commands. For more information about them, see

   `http://java.sun.com/products/jdk/1.1/docs/api/java.awt.Graphics.html`

5. Change the width or height of the `Slate` and run the program again. You should see that the image adjusts its size and proportions to fit the size of the window. You are going to write drawing programs that do the same thing. The idea is to use variables and parameters to make programs more general; in this case the generality is that we can draw images that are any size or location.

Use a bounding box

The arguments that `draw` receives (not including the graphic object) make up a **bounding box**. The bounding box specifies the invisible rectangle in which `draw` should draw.

1. Inside draw, create a new bounding box that is the same height as the `Slate` but only one-third of the width, and centered. When I say "create a bounding box" I mean define four local variables that will contain the location and size. You are going to pass this bounding box as an argument to `drawSnowperson`.
2. Create a new method named `drawSnowperson` that takes the same parameters as `draw`. To start, it should draw a single oval that fills the entire bounding box (in other words, it should have the same position and location as the bounding box).
3. Change the size of the `Slate` and run your program again. The size and proportion of the oval should adjust so that, no matter what the size the `Slate` is, the oval is the same height, one-third of the width, and centered.

Make a snowperson

1. Modify `drawSnowperson` so that it draws three ovals stacked on top of each other like a snowperson. The height and width of the snowperson should fill the bounding box, as in the figure on the quiz.
2. Change the size of the `Slate` and run your program again. Again, the snowperson should adjust so that the snowperson always touches the top and bottom of the screen, and the three ovals touch, and the proportion of the three ovals stays the same.
3. At this point, show your program to me so that I can make sure you understand the basic ideas behind this assignment. You should definitely not continue work on this assignment until I have seen your program.

Draw the snowperson *American Gothic*

1. Modify `draw` so that it draws two snowpeople side by side. One of them should be the same height as the window; the other should be 90% of the window height. (Snowpeople exhibit a sexual dimorphism in which females are roughly 10% smaller than males.)

   Their widths should be proportional to their heights, and their bounding boxes should be adjacent (which means that the drawn ovals probably will not quite touch). The pair should be centered, meaning that there is the same amount of space on each side.

With a corn-cob pipe...

1. Write a method called `drawFace` that takes the same number and type of parameters as `drawSnowperson`, and that draws a simple face within the given bounding box. Two eyes will suffice, but you can get as elaborate as you like.

   Again, as you resize the window, the proportions of the face should not change, and all the features should fit within the oval (with the possible exception of ears).
2. Modify `drawSnowperson` so that it invokes `drawFace` in order to fill in the top oval (the face). The bounding box you pass to `drawFace` should be the same as the bounding box that created the face oval.

Give the Snowpeople T-shirts

1. Add a new method called `drawShirt` that takes the usual parameters and that draws some sort of T-shirt logo within the bounding box. You can use any of the drawing commands, although you might want to avoid `drawString` because it is not easy to guarantee that the string will fit within the bounding box.

2. Modify `drawSnowperson` so that it invokes `drawShirt` in order to emblazon something on the chest (middle oval) of each snowperson.

Make Mrs. Snowperson pregnant

1. I realize that this is a little risqué, but there is a point. Modify `draw` so that after drawing Mrs. Snowperson, it draws a baby snowperson inside her abdomen oval.

   Notice that adding something to `draw` affects just one snowperson. Adding something to `drawSnowperson` affects all snowpeople.

Note: The next part of the assignment will not make sense until after class on Friday, so I recommend stopping here (if you get this far in lab).

Make all snowpeople pregnant

Ok, now let's imagine that instead of making a particular snowperson pregnant, we want to make all snowpeople pregnant. Instead of adding a line to `draw`, we would add a line to `drawSnowperson`. What would that line do? It would invoke drawSnowperson! Can you do that? Can you invoke a method from within itself? Well, yes, you can, but you have to be careful.

SO DON'T DO THIS YET!!!

Think for a minute. What if we draw a big snowperson, and then put a small snowperson in the abdomen oval. Then we have to put an even smaller snowperson in the small abdomen oval, and so on and so on. The problem is, it would never stop. We would be drawing smaller and smaller snowpeople until doomsday!

One solution is to pick a minimum snowperson size, and say that below that size, we refuse to draw any more snowpeople.

1. Add a line at the beginning of drawSnowperson that checks the height of the bounding box and returns immediately if it is less than 10.

   ```
   if (height < 10) return;
   ```

2. Now that you have that line, it is safe to add code at the end of the method so that after drawing the ovals, and the face, and the t-shirt, it invokes drawSnowperson to put a small snowperson in the abdomen.

What to turn in

As usual, print out the final version of your program, write your name in the upper right corner, and turn it in!