COLBY COLLEGE
Department of Mathematics
and Computer Science

CS115
Spring 1999

Allen B. Downey

Homework 4: Snowpeople!

Due: Wednesday 3 March

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.

Make a new project

1. Use the directions I handed out last week to create a new project, except this time use the project stationery named "cs115 – with graphics". Name the new project `Snowperson.prj` and run it. Notice that three windows appear:

   Application window: Prints messages about the running program, including run time errors, if any.

   Console: Contains any output generated with `print` or `println`. For graphical applications, the console is useful for debugging.

   The Frame: The window that contains all the graphics. I found that I could not close this 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. Go through the usual process of renaming the project, making sure the source code is named `Snowperson.java` and the class is named `Snowperson`, and the Java target is named `Snowperson`.

3. Look over the source code as it currently exists and make sure you understand all the code in `paintBox`. Fiddle with it 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`

Make a snowperson

1. Create a new method named `paintSnowperson` that takes the same parameters as `paintBox`. It should draw 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. Remove the existing code from the body of `paintBox` and add code that invokes `paintSnowperson`, passing as arguments a bounding box that is the same height as the window but only one third the width, and centered.
3. You can resize the graphics window by dragging the lower-right corner of the Frame. Each time you resize, Java clears the window and invokes `paint` again, which invokes `paintBox`, which makes your program redraw the snowperson according to the new window size.

   As you resize the screen, the snowperson should always touch the top and bottom of the screen, and the three ovals should touch, and the proportion of the three ovals should stay the same.

Draw the snowperson *American Gothic*

1. Modify `paintBox` 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 `paintFace` that takes the same parameters as `paintSnowperson`, 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 `paintSnowperson` so that it invokes `paintFace` in order to fill in the top oval (the face).

Give the Snowpeople T-shirts

1. Add a new method called `paintShirt` 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 `paintSnowperson` so that it invokes `paintShirt` 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 `paintBox` so that after drawing Mrs. Snowperson, it draws a baby snowperson inside her abdomen oval.

   Notice that adding something to `paintBox` affects just one snowperson. Adding something to `paintSnowperson` affects all snowpeople.

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 `paintBox`, we would add a line to `paintSnowperson`. What would that line do? It would invoke paintSnowperson! 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 paintSnowperson 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 paintSnowperson 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!