Computer Science — Project—SnowflakeCatcher

ASSIGNMENT OVERVIEW

In this assignment you'll be developing an animated, interactive game using the Processing environment. This game will involve an animated "snowflake" that falls from the top of the screen, and a paddle-style "catcher" at the bottom of the screen which you can move left and right to catch the snowflake. The more snowflakes you catch, the higher your score!

This assignment is worth 50 points and should be uploaded by 23:59:59 on the date given in class.

BACKGROUND

You've probably already looked at graphical representations of objects, animated so that they appear to move on the screen. In this assignment, you'll be animating a graphical "snowflake." We also want to be able to accept input from the mouse or keyboard at the same time that we're animating the object, and that turns out to be a little tricky: how do you get an animation loop to repeat at the same time that you're waiting for input that may or may not be entered?

This kind of input—input that the loop is expecting but not waiting for—is called an *event*, and the code that checks for the event is called an *event listener*. Writing event listeners is an important part of learning to code any kind of program that needs to keep running while simultaneously managing user input as it occurs. The Processing environment has event listeners built in from the start, so writing programs that monitor the possibility of events is quite easy.

We'll be working through the writing of the `SnowflakeCatcher` game one step at a time.

ASSIGNMENT SPECIFICATION

Write three classes that can be used to play the game `SnowflakeCatcher`:

1. `Snowflake`
   This class tracks the characteristics of a graphical snowflake that will "fall" down the screen. We'll need to consider the `x`- and `y`-coordinates of the snowflake, its `velocity` as it falls, its `radius`, and perhaps its `color`. You'll want to write "getters" and "setters" for each of these. Additionally, there should be a `move` method that takes a `deltaTime` value as a parameter, and that calculates the new `x`- and `y`-coordinates of the snowflake each time it is called.
2. `Catcher`
   The catcher is a paddle that moves left or right along the bottom of the screen, and it will ultimately be controlled by the user via pressing the left- and right-arrow keys on the keyboard. Instance variables for the catcher should monitor the state of the `x`- and `y`-coordinates of the catcher, its `width` and `height`, along with "getters" and "setters" for each of these.
3. `SnowflakeCatcher`
   This class will create an instance of a `Snowflake` object and a `Catcher` object, and display the snowflake as it falls and the catcher as it is controlled by the user.

DELIVERABLES

`SnowflakeCatcher.zip`, a zipped directory containing the `Snowflake.pde`, `Catcher.pde`, and `SnowflakeCatcher.pde` files that can be used to play the game.

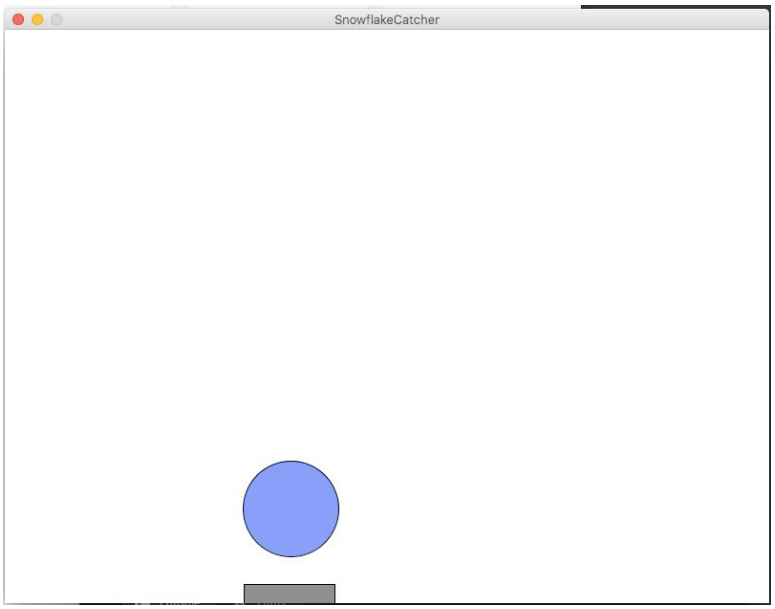
These three files will be developed by you based on details in this document. To submit your assignment

for grading, copy your file to your directory in `/home/studentID/forInstructor/` at *crashwhite.polytechnic.org* before 23:59:59 on the deadline given in class.

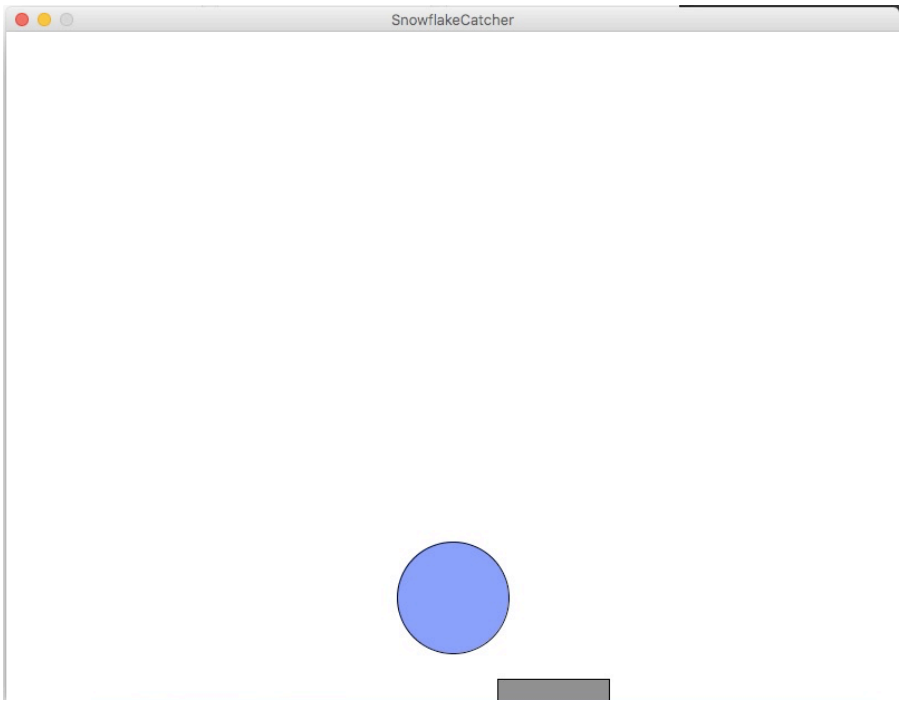
ASSIGNMENT NOTES

- `Snowflake` is the first class you should write based on the specifications above. The specifications given above are relatively straightforward, but you may want to read the next bullet point before you get too far...
- In the Processing environment, there's not really a convenient way of testing the Snowflake class without writing a main program that you can use as a tester, so let's get that set up as well. Write the `SnowflakeCatcher` class so that you can test your classes as you go.
- A Processing main program usually consists of three parts:
  1. a declaration of the variables that will be used in the program
  2. a `void setup()` method that is run once at the beginning of the program
  3. a `void draw()` method that is run repeatedly in an infinite loop. As instructions are executed, as variables change, and as input from the user is delivered, the `draw()` method continually alters the state of the program.
- Note that the Processing environment uses the `float` type rather than the `double`.
- There are three primary challenges in this assignment: creating a snowflake that falls, creating a catcher that responds to user input, and determining when a snowflake has been successfully caught by the catcher. It makes sense to tackle these challenges in that order.
- Questions you may consider as you're working on the game: Should the snowflake always fall from the same location? Should the paddle always start at the same location? Should the snowflake always have the same color and radius? Do we want to keep score? Can the game be made to get more challenging as it progresses? How could we make more than a single snowflake fall at the same time? Can I use a user's trackpad to move the catcher rather than the left- and right-keys? *Don't be waylaid by these distractions!* These are all perfectly fine features to build into the game, but we're not at a point where we can anticipate all of them. Build the basic game, and *then* introduce new features, one at a time.
- We're going to know that the catcher has caught the snowflake when the two objects collide with each other... but how exactly will we determine that?

Has this snowflake collided with the catcher yet? Is it about to?



Is *this* snowflake about to collide with the catcher? Is it about to? How will we know?



- Avoid the use of "magic numbers" in your program. If your snowflake is going to have a radius of 20 pixels, and therefore a width and height of 40 pixels. don't write this:

```
ellipse(x, y, 40, 40);
```

To anyone reading the code, the meaning of 40 isn't immediately clear. Although it uses an extra

line at some point in the program, this is superior:

```
float snowflakeRadius = 20;
ellipse(x, y, snowflakeRadius * 2, snowflakeRadius * 2);
```

And now, anywhere we need to identify the radius of the snowflake we can use that variable. This is a good example of *self-documenting code*.

GETTING STARTED

- You'll be using the Processing environment to write, develop, and run this assignment. Make sure you have the Processing environment downloaded and set up on your computer.
- Begin by writing the `Snowflake` class, but then jump over to writing the `SnowflakeCatcher` class so that you can use it to test your work as your go.
- If you run into any difficulties, you have two common strategies that you can use in debugging your program: use Processing's built-in debugger, or use a copious amount of println statements sprinkled through your code to identify the state of various variables. Those values will be printed out in the console as your program runs, and you can use that information to help you figure out what's going on.

EXTENSIONS

- Add in as many features as you wish selected from those listed in the fifth bullet point under "Assignment Notes" above.
- If you know about `git`, consider using it to track your progress as you work on this program. You don't want versions of your program that work successfully to be lost as you attempt to make improvements.

QUESTIONS FOR YOU TO CONSIDER (NOT HAND IN)

1. What is "feature creep," and how does it relate to the fifth bullet point under "Assignment Notes?"
2. What is refactoring? What are the benefits of refactoring? Are there any drawbacks to refactoring?
3. Why do you think the Processing environment uses the `float` type to represent real numbers rather than the more precise `double` type that we typically use in this course?