What’s the Point?

Today, we’re going to play around with objects a bit. To see objects in motion, we’re going to implement Snake, a simple computer game. I’ve posted a quick overview of what we’ll be doing on Panopto; why don’t you go watch it now?

What We’ll Be Doing

Here are the key techniques we’re going to practice:

- Declaring a class, including data fields and methods
- Instantiating objects and arrays of objects
- Using boolean variables to control the properties of an object
- Infinite game loops with keyboard interactions

Pair up!

Remember, we’ll be doing pair programming this semester. So choose a partner (somebody you’ve never worked with in section), and find a computer.

Part 1: Working on Classes Together

Since this is the first time that we’ll be creating any objects, the section will work together to write the declaration. Once you are done, the section will work together on some pseudocode main() and one of the methods - and then you will start writing the real Java code at your computer.

The SnakeSegment Class

In this section, you will write a class named SnakeSegment. Each object of this class will represent a single segment in the snake. It will need variables to store the current \((x, y)\) location of the segment on the screen (integers will work here), as well as a variable to store the current direction that it is facing. (A segment can be facing up, down, left, or right.) In program 2, we’ll add additional fields.

In addition, the class should include three instance methods. The changeDirection() method will be called by main() when the user hits a key on the keyboard - it will change the direction of the segment (except that it will not allow the segment to reverse - those changes are banned). The move() method will move
the segment one step in the direction that it is currently facing. Finally, the draw() method will draw the segment on the screen; it should include some small mark to show which way the segment is moving.

In Section, discuss exactly what the parameters and return types for these methods should be. Together, write the entire class declaration - except, leave the method bodies blank.

Part 2: Pseudocode

Next, the section will work together to write some of the pseudocode you’ll need. First, we’ll work on main().

Pseudocode - main()

In Program 1, you will have an infinite loop - the game will run forever. You will only have a single object, and you’ll move it around on the screen. Each pass of your loop will need to do the following steps (I recommend you do them in exactly this order):

- Read the next key from the keyboard (if any). Call changeDirection() on the object if the user is trying to change the direction of the “snake”.
- Use the move() method to update the position of the snake on the board.
- Clear the screen, then use the draw() method to show the snake.
- Call StdDraw.show(int) to show the screen, and pause for a bit.

To check to see if the user is pressing a key, use the following two methods:

- StdDraw.hasNextKeyTyped() - returns a boolean, which indicates whether or not there is another key to read.
- StdDraw.nextKeyTyped() - returns a char, which is the next key typed by the user.

Pseudocode - changeDirection()

The changeDirection() method will be called by main() when the user presses a key to change direction. (You can choose which keys you want to use - but note that you should limit yourself to the letters. Arrow keys are a little more difficult to read.)

For the most part, this is a simple method: if the user presses the key to move left, the direction should be changed to point left. However, if the snake is already moving in the opposite direction, then changeDirection() should do nothing. (Do not throw an exception; just ignore the input. If you want to print out a message to System.out, you may do so - but it’s not required.)
This section ends when your Section Leader says so; you may finish all of the points above, or your Section Leader may end it early.

☑ CHECKPOINT 1

Everybody who has been taking part should get this checkpoint for free!

Program 1

Once you’ve written the pseudocode as a Section, it’s time to write the Java code. Open up DrJava, and create TWO different files: one to hold the SnakeSegment class, and another to hold your main() method.

NOTE: When you try to Run your program in DrJava, you must always select the class with the main() method first. If you have SnakeSegment.java selection, DrJava will not run your program.

Write your main() method; port the pseudocode that the class came up with. Make sure that your code calls StdDraw.setScale(int,int); that it creates a SnakeSegment object; that it includes an infinite loop which reads from the keyboard; and that it draws the screen on each pass of the loop.

In SnakeSegment.java, copy in the code that the class wrote together. Then work on filling in each method body: you can use the pseudocode that the class came up with to write changeDirection(), but you will have to write your own implementations for move() and draw().

move() Requirements

Your move() method must:

- Update the segment’s (x,y) location based on the direction that it is currently facing.
- If the snake hits any of the 4 edges of the screen, then wrap it around to the other side.

Your move() method must NOT draw anything to screen. Instead, draw the segment in the draw() method.

draw() Requirements

Your draw() method must:

- Draw something on the screen (a circle, a square, or whatever you like) to represent the single segment.
- Draw something extra, which makes it possible to see which way the segment is “facing.”
Putting It All Together

When you’re done, your program should have a single segment of the snake moving around on screen; it should allow you to turn it in any of the four directions - but it should never allow you to turn it around 180 degrees.

Program 2

Program 2 will be an extension of Program 1. In Program 2, you will build a long snake, with many segments. You will write code to chain the segments together - and you will also write code to detect when the snake runs into itself.

An Array of Segments

In Program 1, you only had a single SnakeSegment object. In this program, allocate an array of length 10, and create 10 different SnakeSegment objects.

Add a boolean field to your SnakeSegment class, which indicates whether or not the segment is the head of the snake. Mark element [0] of your array as the head - the rest will be the body. Then modify the code in draw() so that the head will look different than the other segments. (If you want to simply modify the body segments so that they do not draw the extra part to show direction, that’s OK.)

Add checks in changeDirection() and move() - if either method is called on any node other than the head, throw an exception.

Add a new method, follow(SnakeSegment). It should take a single SnakeSegment reference, and it should move the current segment to the location of the parameter. Throw an exception if this method is ever called on the head.

Now, modify main(). When the user hits a key, make sure that you call changeDirection() only on the head. Likewise, call move() only on the head. But add a loop which calls follow() on all of the body segments (have each one follow the segment in front of it). Finally, update the draw() code so that it’s a loop which draws all of the objects.

Finally, write a static method (in the same class as main()) which takes an array of SnakeSegment objects as a parameter. It should use a simple loop (not nested) to compare the first segment to all of the rest. If any of them are at the same location, then return true. Call this method from main(), and report an error to the user (and then end the program) if the snake ever runs into itself.

✓ CHECKPOINT 2

Raise your hand. Your SL will come over and verify that you completed this step correctly.
Additional Features

Some people finish their code early. If you finish early, here are additional features to add to your snake program. You can choose which feature(s) to implement - do them in any order, and any combination.

However, if you leave early and you have not implemented all of these features, you will not get the fourth checkpoint.

If you’re struggling, don’t worry! None of this is required. Just keep working on the main programs (above), and do the best you can.

Extra Feature: Add a Tail

Your snake has a boolean which indicates whether or not a segment is the head of the snake - and when you draw, you make the head look different than the rest.

Now add another boolean for the tail. It should work just like any other body segment - but it should be drawn differently. (Note: In order to make this work, you’ll have to update the follow() method to record what direction the segment moved - so that the tail can be drawn properly.)

Extra Feature: Killer Walls

If the snake runs into any of the walls, report to the user and terminate the game.

Take this further, and add at least one pair of “doorways” - which are special locations which allow the snake to get from one side to the other. If you add this, make sure you that you draw the walls on the screen, so that the user can tell where the doors are!

Extra Feature: Food

Add randomly-generated “food” to the screen. If the snake “eats” the food (that is, the head moves over the food), then extend the length of the snake by one (this will require resizing the array). When the snake eats a piece of food, replace it with another, at a random location on the screen.

Draw the food on the screen, so that the player can find it.

For the first version, just have one piece of food on the screen at once. If you finish that, extend it to allow multiple pieces of food. (You will probably want to create another class to hold all of the FoodItem objects you will be creating.)

Extra Feature: Multiplayer

Your keyboard has space for two people to play at once. A common way to do this is to use the keys WASD to control Player 1, and IJKL to control Player 2. Add a second snake, and have two people play at the same time. End the game if either snake touches itself or the other snake.
The End: Clean Up

The G/S 930 lab is host to sections from several of our programming classes, and is available the rest of the time for any CS student to use to work on their programming assignments. To help you get in the habit of leaving your work space neat and clean, we’ll end each section activity with this checkpoint, which also serves as confirmation that you attended section - even if you get nothing else done, you can clean up before you go, thereby earning one checkpoint and proving you attended.

- Log out of your computer.
- Pick up your papers, writing implements, cell phones, etc.
- Push in your chairs.

✓ CHECKPOINT 4

Raise your hand. Your SL will come over and verify that you completed this step correctly.

As with so much in this course, thanks to Dr. McCann for some of the ideas - and even some of the exact text - for this Section Activity!