Data Structures

Venom – 85 course points

In this assignment, you will be using various BST implementation techniques discussed during lecture to simulate the Marvel villain Venom's path in choosing a host body to take over. **You do not need to know anything about Marvel or Venom to complete this assignment.**

Start your assignment early! You need time to understand the assignment and to answer the many questions that will arise as you read the description and the code provided.

Refer to our Programming Assignments FAQ for instructions on how to install VSCode, how to use the command line and how to submit your assignments.

View Venom under Full Submission on Autolab to download the project files.

The assignment has two components:

1. Coding (85 points) submitted through Autolab.
2. Reflection (3 points) submitted through a form.
   - Submit the reflection **AFTER** you have completed the coding component.
   - Be sure to sign in with your RU credentials! (netid@scarletmail.rutgers.edu)
   - You cannot resubmit reflections but you can edit your responses before the deadline by clicking the Google Form link, signing in with your netid, and selecting "Edit your response"

Overview

Welcome to the Venom assignment!

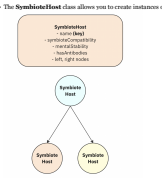
In this assignment, you play the role of Venom. Venom is an otherworldly organism, called a symbiote! These organisms need to latch on to and essentially merge with different hosts to survive. However, picking the right host is easier said than done.

In this assignment, we will organize and store hosts in a binary search tree, so as to easily organize Venom's options, and therefore deduce the best host to merge with.

Implementation

Overview of files provided

- The **SymbioteHost** class allows you to create instances of Symbiote Hosts. These hosts are the people that make up our BST and **serve as nodes**. These nodes (SymbioteHost) store information such as the host's name, compatibility with the symbiote (Venom), mental stability, whether or not they have symbiote (Venom) antibodies, and the host's left and right children. Do not edit or submit to Autolab.



- The **Venom** class contains methods that manipulate BSTs in order to have the symbiote conduct certain tasks. Edit the empty methods with your solution, but DO NOT edit any provided ones or the method signatures of any method. This is the file you submit.
- **StdIn** and **StdOut** are libraries that handle input and output. Do not edit these classes.
- An input file is provided; it stores data handling Symbiote Host profiles. You are welcome to create your own input files, as they will not be submitted to Autolab.

Unlike the last assignment, you are not provided with a driver. However, you are provided with two test classes:

- **Driver.java**, a terminal-based visual driver with main() methods you can fill in to run methods on the Venom class and print out the resulting tree.
- **VenomTest.java**, a JUnit test class with tests you can use to test specific edge cases through assertion statements.
- See the **Testing Your Code** section below for more information.

Both files contain comments and examples regarding how to make the test code work.

Venom.java

- DO NOT add new import statements.
- DO NOT change any of the method's signatures.

Methods to be implemented by you:

createSymbioteHosts(String filename): PROVIDED

DO NOT EDIT THIS METHOD.

As Venom, you have your eye on a group of people you feel could be potential hosts...now it's time to gather their information. Using the input file provided containing Venom's desired vessels, store the data of each person in an array.

This method will open a file to be read and create objects of type SymbioteHost based on the information within that file.

The input file provided will be of the format:

- One line containing the number of SymbioteHosts in the file, let's say p
- For each of p people:
  - One line containing the host's first and last name
  - One line (integer) containing the host's compatibility with the symbiote
  - One line (integer) containing the host's mental stability
  - One line (boolean) containing whether or not the host has antibodies, preventing them from being a victim of Venom

This method does the following:

1. Creates an array of size p, this value being the total number of people within the input file. The value, p, is also the first integer of the input file.
2. For each person in the input file:
   1. Reads the person's name, compatibility, mental stability, and whether or not they have antibodies.
   2. Uses this information to create a new SymbioteHost object. For the left and right children of the SymbioteHost object, put null as this data is not given.
   3. Inserts the new SymbioteHost into the **next open space in the array.** If index 1 is taken, insert into index 2.
   4. Then returns the resulting array.
3. The method returns an array of type SymbioteHost. Each index of the array contains a unique SymbioteHost.

insertSymbioteHost(SymbioteHost symbiote)

As Venom, we need an easy way to store and traverse the list of potential target hosts. Utilize the Binary Search Tree (BST) data structure discussed during the BST lectures to do so.

- The BST is a Symbol Table that stores SymbioteHost nodes.
  - This is a bit different from the code from lecture that had distinctive <key,value> pairs.
- The key is a SymbioteHost's name. Use the name as the key to compare and traverse through the BST.
- The value is also stored in SymbioteHosts in the form of remaining host's characteristics.

This method takes a SymbioteHost node a parameter, and adds the node to the BST rooted at **root**. Remember that the **root** is a reference to the root node of the BST.

- Follow the BST insertion algorithm as discussed in class to insert a SymbioteHost into the BST.
- The BST has unique SymbioteHosts' key, meaning no two SymbioteHosts can have the same name. If you encounter the same key again, **UPDATE** the existing node with its new stability, compatibility and antibodies values.
- SymbioteHosts with names greater than a node's name are stored to the right of the node, otherwise they are stored to the left. **Compare based on names** Use the compareTo() method to determine where to place SymbioteHosts lexicographically (ie. comparing strings).

Note that although only names are used as keys, each node represents an entire SymbioteHost.

How to test:

- You can complete the **Driver** class to finish the driver. We provide simple driver code in the main() method, you must fill in each case in the switch statement. **The class is not fully implemented -- use the comments in the code to make the code work.**
- You can also use JUnit test cases. We provide a sample test method for insertSymbioteHost. A smaller test input file (testInput.in) is provided for you to work with fewer nodes than the input file.

buildTree(String filename)

This method builds the BST from the input filename parameter.

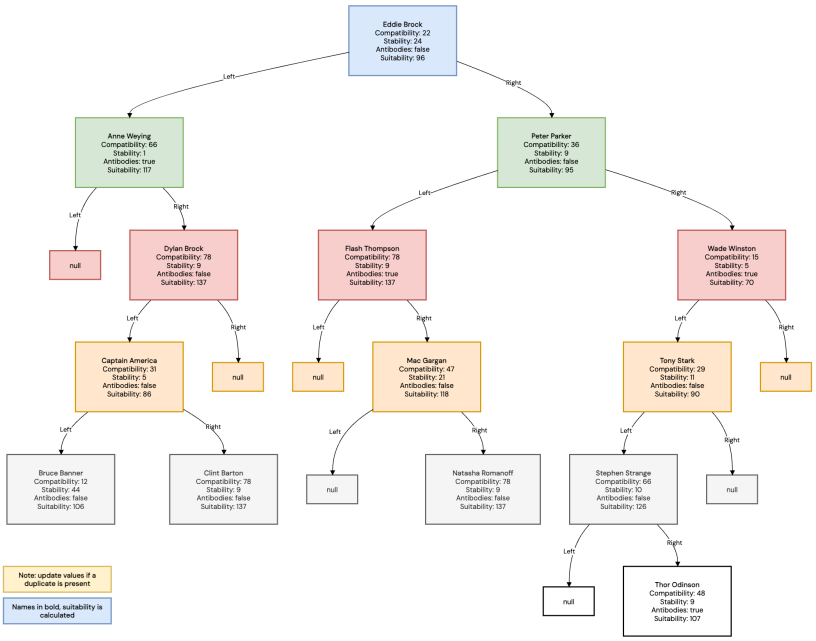
1. Call createSymbioteHosts using the filename given as parameter. createSymbioteHosts will return an array of SymbioteHost that were read from the file.
2. Then, call insertSymbioteHost for every element in the array returned by createSymbioteHosts to insert each into the BST.

IMPLEMENT the insertSymbioteHost method and update the method's code to make this method work. Once the insertSymbioteHost and buildTree methods are implemented, submit Venom.java with these two methods implemented under Early Submission for extra credit.

How to test:

- You have two options for writing tests. Filling in the Driver class, or writing JUnit tests. The Driver class may be easier, but less helpful.
- Use BST ordering to conclude whether or not your code works as intended.

Here is the expected tree structure when using hosts0.in. Note that names are emphasized with bolds.



If you use the printTree() method, the structure will look like this:

```
+---- Host{name='Eddie Brock', symbioteCompatibility=22, mentalStability=24, hasAntibodies=false, suitability=96}
      |--L-- Host{name='Anne Weying', symbioteCompatibility=66, mentalStability=1, hasAntibodies=true, suitability=117}
      |      |--L-- null
      |      |--R-- Host{name='Dylan Brock', symbioteCompatibility=78, mentalStability=9, hasAntibodies=false, suitability=137}
      |      |      |--L-- Host{name='Captain America', symbioteCompatibility=31, mentalStability=5, hasAntibodies=false, suitability=86}
      |      |      |      |--L-- Host{name='Bruce Banner', symbioteCompatibility=12, mentalStability=44, hasAntibodies=false, suitability=106}
      |      |      |      |--R-- Host{name='Clint Barton', symbioteCompatibility=78, mentalStability=9, hasAntibodies=false, suitability=137}
      |      |      |--R-- null
      |--R-- Host{name='Peter Parker', symbioteCompatibility=36, mentalStability=9, hasAntibodies=false, suitability=95}
             |--L-- Host{name='Flash Thompson', symbioteCompatibility=78, mentalStability=9, hasAntibodies=true, suitability=137}
             |      |--L-- null
             |      |--R-- Host{name='Mac Gargan', symbioteCompatibility=47, mentalStability=21, hasAntibodies=false, suitability=118}
             |      |      |--L-- null
             |      |      |--R-- Host{name='Natasha Romanoff', symbioteCompatibility=78, mentalStability=9, hasAntibodies=false, suitability=137}
             |--R-- Host{name='Wade Winston', symbioteCompatibility=15, mentalStability=5, hasAntibodies=true, suitability=70}
                    |--L-- Host{name='Tony Stark', symbioteCompatibility=29, mentalStability=11, hasAntibodies=false, suitability=90}
```

```
        |--L-- Host{name='Stephen Strange', symbioteCompatibility=66, mentalStability=10, hasAntibodies=false, suitability=126}
        |      |--L-- null
        |      |--R-- Host{name='Thor Odinson', symbioteCompatibility=48, mentalStability=9, hasAntibodies=true, suitability=107}
        |--R-- null
|--R-- null
```

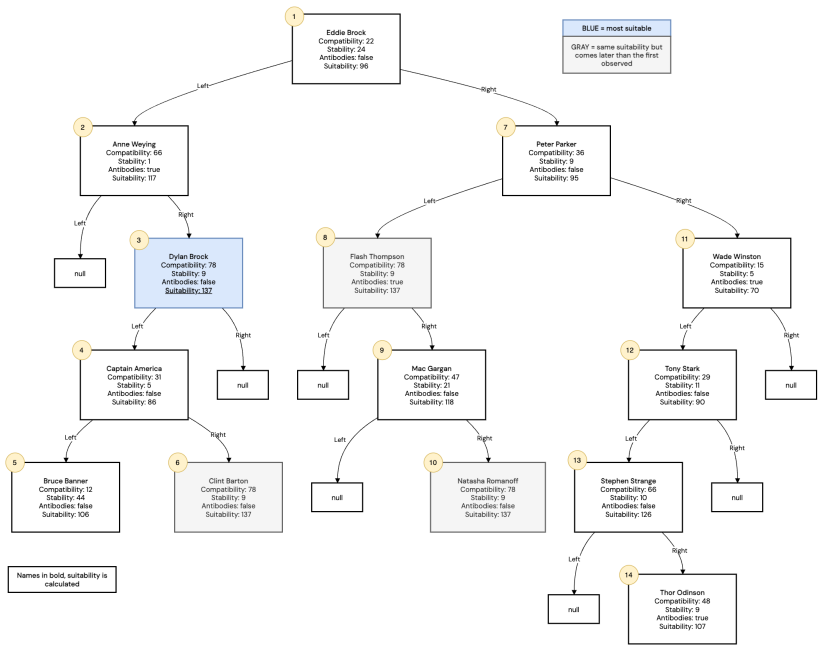
findMostSuitable()

Venom does not have time to waste on hosts with low suitability... Venom needs to find the host that is most suitable for a takeover. Traverse through BST of potential hosts created in the previous method and find Venom's target.

This method will traverse through the BST created using **preorder traversal.** It returns the SymbioteHost that is deemed the MOST suitable.

- The SymbioteHost class has a built-in method to calculate the suitability of a SymbioteHost, and you may call it for some SymbioteHost **sh** by calling **sh**.calculateSuitability(). This method uses the compatibility and mental stability factors to compute the suitability.
- Implement this method recursively -- you may use private helper methods to do so (refer to lecture's preorder code).
- We define a host match as a host whose suitability is GREATER than the current host's best value.
    - **If two hosts have the same (best) suitability, the most suitable host is the first one (meaning the first one seeing during the traversal).**

RUN buildTree in your test code before testing this method. The expected output for this method on hosts0.in is Dylan Brock. In the image below, the blue node represents the most suitable host, the yellow numbers represent the order in which that node is visited, and gray nodes have the same suitability as the best one but are visited later.



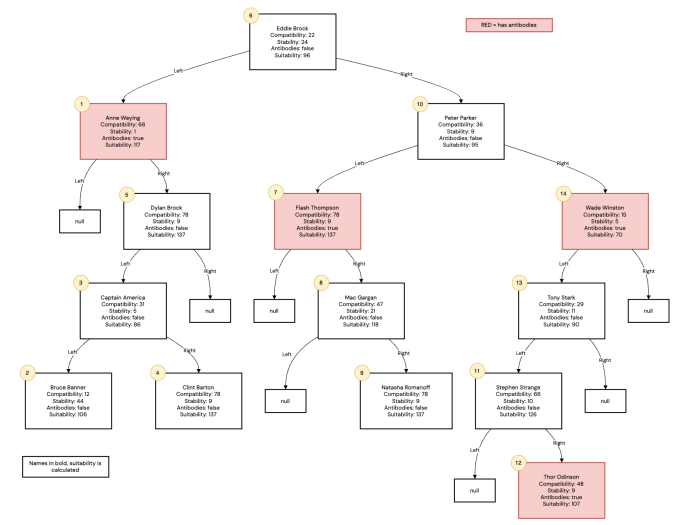
findHostsWithAntibodies()

Venom just discovered that some targets possess a strain of antiviral antibodies in their bodies. While merging with these targets is still possible, it is risky and may weaken Venom. So, Venom must traverse through the tree yet again to find all of the people containing antibodies, so it knows who to be wary of when selecting hosts.

This method traverses the BST using **inorder traversal**, and returns an ArrayList of SymbioteHosts that are known to have Venom-resisting antibodies. This method will be called again in cleanUpTree().

- Implement this method recursively. Refer to lecture's inorder code.

RUN buildTree in your test code before testing this method. The expected array of SymbioteHosts returned is (in this order for hosts0.in):



Resulting ArrayList

0	1	2	3
Anne Weying Compatibility: 66 Stability: 1 Antibodies: true Suitability: 117	Flash Thompson Compatibility: 78 Stability: 9 Antibodies: true Suitability: 137	Thor Odinson Compatibility: 48 Stability: 9 Antibodies: true Suitability: 107	Wade Winston Compatibility: 15 Stability: 5 Antibodies: true Suitability: 70

findHostsWithinSuitabilityRange(int min, int max)

To optimize its list of targets, Venom has decided to gather a list of potential hosts with suitability not-of-interest. This includes potential hosts with suitability ranging from 0-100, inclusively. Traverse through the BST one final time and provide Venom with all of the people falling into the undesired range.

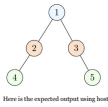
This method traverse the BST using **level order traversal**. The method takes in minSuitability and maxSuitability as parameters, and returns an ArrayList of all SymbioteHosts with suitabilities falling into the provided range. Like the previous method, this method will also be called in cleanupTree.

Algorithm for level-order traversal of the BST:

- Use the provided Queue class to initialize a queue of SymbioteHost.
    - Queue **queue** = new Queue<>(); initializes the queue.
    - Then, you may use:
        - **queue.enqueue(item)** enqueues the parameter item into a queue.
        - **queue.dequeue()** dequeues an item as discussed in class.
        - **queue.isEmpty()** checks if the queue is empty.
- Add the BST's root to the Queue.
- While the queue is not empty:
    - Dequeue a node.
    - Investigate whether the dequeued SymbioteHost's suitability is within the range. If so, add to the ArrayList to be returned.
        - the range is from min inclusive to max inclusive.
    - Enqueue the node's children.

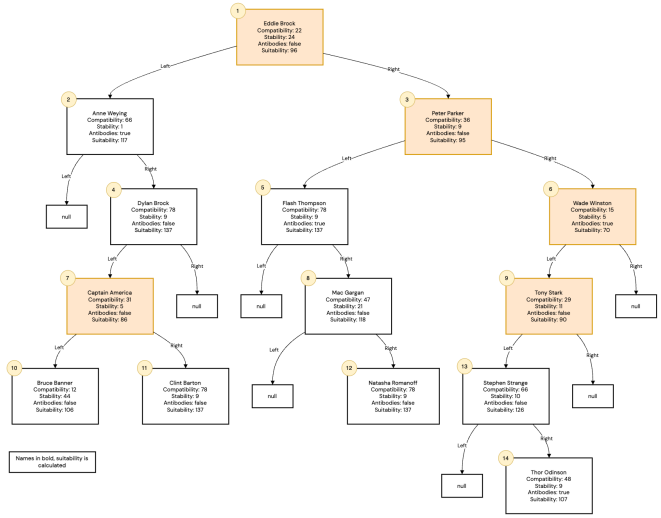
Here is an example for you to visualize:

- The queue starts as empty. Start by enqueuing the root (at position 1).
- Dequeue the node at position 1, visit it, and enqueue its left and right children (positions 2 and 3).
- Repeat the same procedure on all other nodes until the queue is empty:
    - Dequeue the node at position 2, visit it, and enqueue its children.
    - Dequeue the node at position 3, visit it, and enqueue its children.
    - Lastly, dequeue and visit positions 4 and 5. They are leaves.



Here is the expected output using hosts0.in -- run buildTree in your test code before running this method:

%3CmxGraphModel%3E%3Croot%3E%3CmxCell%20id%3D%220%22%2F%3E%3CmxCell%20id%3D%221%22%20parent%3D%220%22%2F%3E%3CmxCell%20id%3D%222%22%20value%3D%22Resulting%20Array%22%20style%3D%22text%3Bhtml%3D1%3Balign%3Dcenter%3BverticalAlign%3Dmiddle%3Bresizable%3D0%3Bpoints%3D%3B%3D%3Bautosize%3D1%3BstrokeColor%3Dnone%3BfillColor%3Dnone%3BfontFamily%3DDM%20Sans%3BfontSource%3Dhttps%253A%252F%252Ffonts.googleapis.co



Resulting ArrayList (in this order)

deleteSymbioteHost(String name)

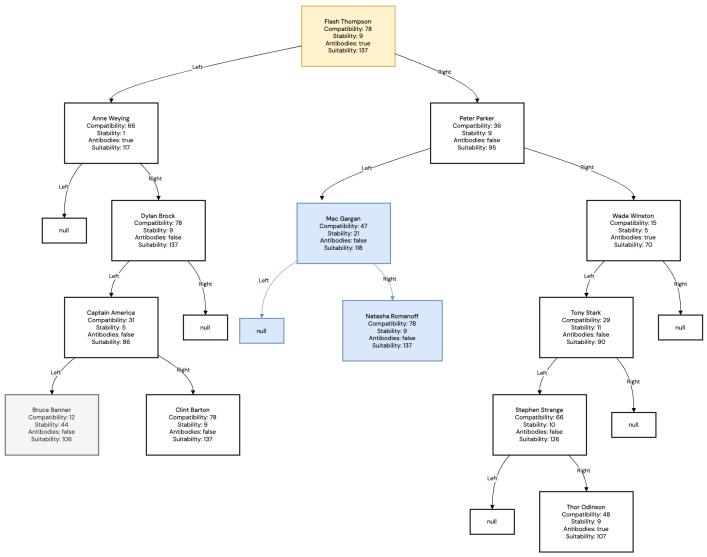
Time is of the essence to Venom. Difficulties with a host of choice WILL NOT BE TOLERATED... even if that means expending a current host for a new host of choice. This method implements the **BST Hibbard deletion** seen in lecture--follow the steps below when coding.

This method removes a node whose name matches the input parameter **name.**

- Search for the node containing **name** using the BST search algorithm seen in lecture. Use the compareTo() method.
- When the node is found:
  - If it has no children, delete the node by setting the parent link to null.
  - If it has only one child, replace the parent link with the child.
  - If it has two children, find the inorder successor of the node, delete the minimum in its right subtree, and put the inorder successor in the node's spot.
- Note: if the root of the BST changes, the root should be updated accordingly.

To test your code using hostsio.in, you may delete Eddie Brock. **ALSO** test deleting other nodes to make sure your code works as intended. The tree result when deleting Eddie Brock is expected to match the image below. **Run buildTree before running this method.**

-- Flash Thompson is the new root of the BST, since we deleted the root.
-- We replaced the node at Flash's old position with Mac Gargan.



cleanupTree() – challenge for the bored (0 points)

With so-called "heroes" on Venom's trail, it must optimize the tree. Venom has decided to remove anybody with antibodies, as well as anybody with suitability not of interest. To complete this final method, utilize the previous methods you've created.

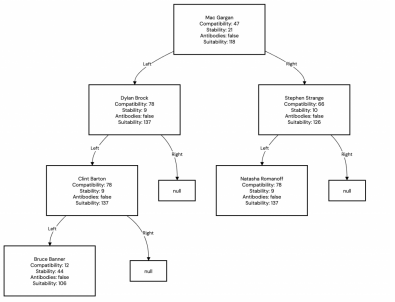
This method removes all nodes that have antibodies AND/OR are within the suitability range of 0-100.

To complete this method:

1. Begin by calling both findHostsWithinSuitabilityRange(0, 100) and findHostsWithAntibodies() and storing their respective ArrayLists.
2. Then, call deleteSymbioteHost(name) on each item that falls into the criteria for removal. Be sure to check that the SymbioteHost currently being removed has not already been removed.

This method requires TWO passes to delete nodes: one pass using the nodes given in findHostsWithinSuitabilityRange, and another pass using the nodes that have antibodies.

After calling this method, your tree will resemble the one below:



Make sure before running that Eddie Brock is NOT still removed from the method prior.

Testing Your Code

Two **incomplete** classes are included which you can finish and then use to test your code.

- **DriverJava**, a terminal-based visual driver with main() methods you can fill in to run methods on the Venom class and print out the resulting tree.
- **VenomTestJava**, a JUnit test class with tests you can use to test specific edge cases through assertion statements.

To implement the Driver:

Simply finish the main() method accorded to the included comments. This driver continually prints command options and reads the user input. In each of the cases for 1-5, you should **call the corresponding Venom method on the "test" object, and then call printTree() to print the output.**

For methods which require parameters to call, the code to read them as command line input has already been provided to you.

You will need to generate expected output yourself, by hand tracing BST insert and traversal with the given input. To test insertSymbioteHost using the driver, implement buildTree and the case for that method in the driver.

To implement VenomTest:

This is a JUnit test class similar to what you have seen before. The first test is given to you, but you must implement the rest.

You can create custom input files or use the ones provided. You should use these to build the tree, then use the assert methods to verify the tree structure is as you expect (for methods where the expected output is a modified tree).

i.e. Create a custom input file corresponding to a tree with "Peter Parker" at the root, then assertEquals("Peter Parker", treeRoot.getName()) to test it was inserted correctly.

- If that root should have a right child "Anne Weying"), check with assertEquals("Anne Weying", treeRoot.getRight().getName()).
- You can continue this pattern using more .getLeft() and .getRight() calls in repeated assertEquals() calls, to check the entirety of the given tree in the small test input file.

To test later methods that involve returning a single object or ArrayList, you can compare each value against the expected ones using assertEquals() statements for each value in each position.

Note: testInput.in is a SMALLER input file for you to run test cases with. If you are trying to replicate the outputs in the description, use hosts0.in.

Implementation Notes

- YOU MAY only update the methods with the WRITE YOUR CODE HERE line.
- **COMMENT all print statements you have written from Venom.java**
- DO NOT add any instance variables to the **Venom** class.
- DO NOT add any public methods to the **Venom** class.
- DO NOT add/rename the project or package statements.
- DO NOT change the class **Venom** name.
- YOU MAY add private methods to the **Venom** class.
- YOU MAY use any of the libraries provided in the zip file.
- **DO NOT** use System.exit()

VSCode Extensions

You can install VSCode extension packs for Java. Take a look at this tutorial. We suggest:

- Extension Pack for Java
- Project Manager for Java
- Debugger for Java
- Test Runner for Java

Importing VSCode Project

1. Download Venom.zip from Autolab Attachments.
2. Unzip the file by double clicking.
3. Open VSCode
    - Import the folder to a workspace through **File** > **Open**

Executing and Debugging

- You can run your program through VSCode or you can use the Terminal to compile and execute. We suggest running through VSCode because it will give you the option to debug.
- How to debug your code
- If you choose the Terminal. **YOU MUST COMPLETE THE DRIVER CODE FIRST**:
    - first navigate to **Venom** directory/folder (the one that directly contains the src, lib and bin folders).
        - to compile: **javac -d bin src/venom/*.java**
        - to execute: **java -cp bin venom.Driver (follow the comments to complete class)**
- Alternatively, you may run JUnit tests by:
    - right clicking "VenomTest.java" on the VS Code sidebar
    - clicking Run Tests
    - **YOU MUST IMPLEMENT JUNIT TESTS, not all are provided.**

Before submission

COMMENT all printing statements you have written from Venom.java

Collaboration policy. Read our collaboration policy here.

Submitting the assignment. Submit *Venom.java* separately via the web submission system called Autolab. To do this, click the *Assignments* link from the course website; click the *Submit* link for that assignment.

Getting help

If anything is unclear, don't hesitate to drop by office hours or post a question on Piazza.

- Find instructors office hours here
- Find tutors office hours on Canvas -> Tutoring
- Find head TAs office hours here
- In addition to office hours we have the Coding and Social Lounge at Hill 250, a community space staffed with lab assistants which are undergraduate students further along the CS major to answer questions.

By Shane Haughton, Elian Deogracia-Brito and Ayla Muminovic