**Assignment 2 - ITSC 2214**

Due October 18, 2019, midnight

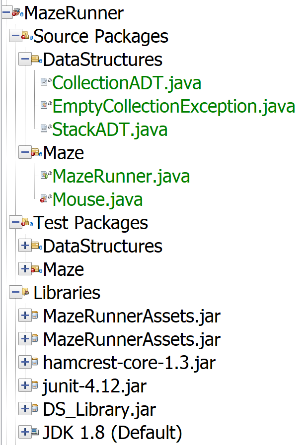
**Objectives**

At the completion of this assignment, you will have completed an ArrayList implementation of a Stack interface, and completed unit testing of the implementation. You will also complete a method in a maze application that makes use of the stack to keep track of paths to explore. You will get more experience writing JUnit test cases for methods, to make sure the methods work as expected. Your ArrayListStack will require you to program with exceptions, so you will get more experience with throwing and catching exceptions and how you work with them when writing test cases.

**Instructions**

**Step 1 - Setup:**

Download and import the MazeRunner.ZIP file into Netbeans and open that project. You should see this when you have imported the project:

Note that there are red error icons. The project won’t compile until you’ve completed a few tasks.

In the MazeRunner project, right-click on the ***DataStructures*** package and select New->Java Class… Create a new class called ***ArrayListStack*** (remember that capitalization matters, so ensure this uses CamelCase spelling). You need to have this class implement the interface StackADT in the DataStructures package, so type ‘***implements StackADT*** after the class declaration. You will see errors because you need to add an import statement in order to have access to the ADT. Click on the error and let Netbeans add the import statement for you (see next image).

You need the ArrayListStack to handle generics, so add a <T> after the class name. Then you want this class to implement the StackADT, so type implements StackADT<T> after the class name, like this:

Once you’ve added the implements stuff, you will see another error, and that’s because you haven’t yet added the methods that implement the StackADT. So, once again, click on the red error message and note that NetBeans offers to “*implement all abstract methods*” (see the next image). Go ahead and click that.

Once you have done this, you will have an ***ArrayListStack.java*** class that is ready for you to complete. All the correct method headers are there, and each method has just an ‘UnsupportedOperationException’ in it.

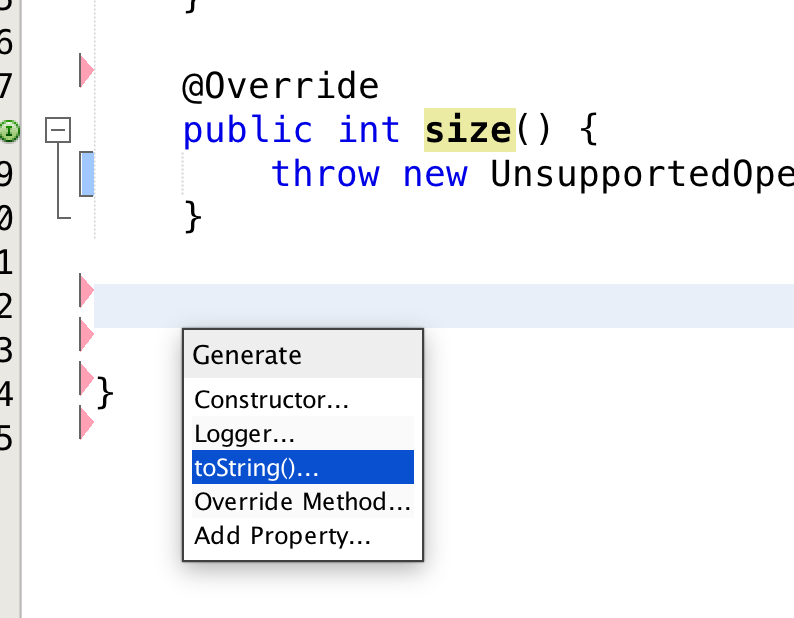
**Step 2 - Implement the *ArrayListStack*:**

You will need to create a (private) ArrayList<T> at the top of this class to serve as your stack. NOTE: you are NOT allowed to use the Java built-in stack. You are making your own stack, using an ArrayList.

You can call the ArrayList as any name.

You will then have to initialize your stack in a class constructor. Your constructor should not take any parameters.

In each of the five methods that were created for you, you will need to remove the UnsupportedOperationException and put in the appropriate code to make the stack work.

You also need to add a toString() method to the ArrayListStack class. To do this, let NetBeans help you. Right click at the bottom of the ArrayListStack file (just before the final brace that closes the class) and select *Insert Code…* then select *toString()…*

Click on the name of your stack in the dialog box that comes up, so that your stack is printed out.

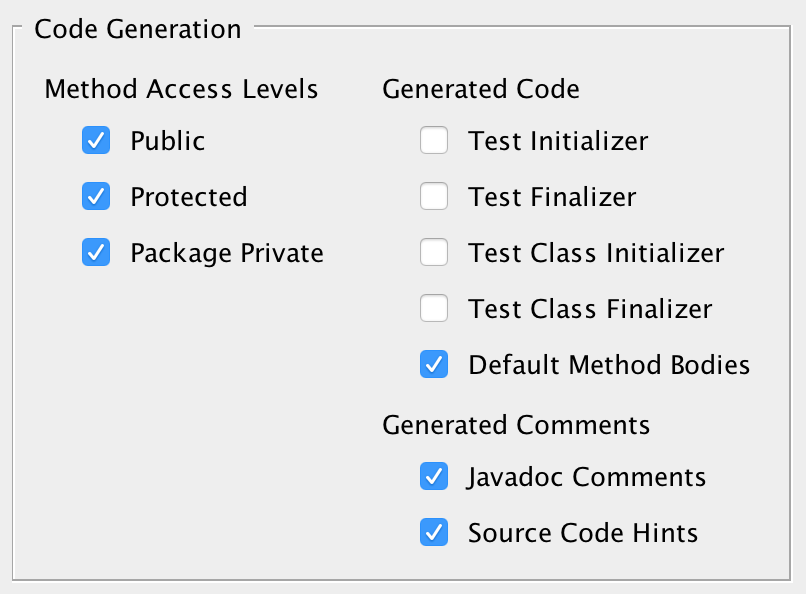
Make sure that you write JavaDoc comments for your ArrayListStack class and generate the JavaDoc files.

Once you have the ArrayListStack implemented, the red errors associated with your source code should all go away.

You aren’t done yet. But you can at least now compile and run the project and see the maze…

**Step 3 – Unit Test the ArrayListStack**

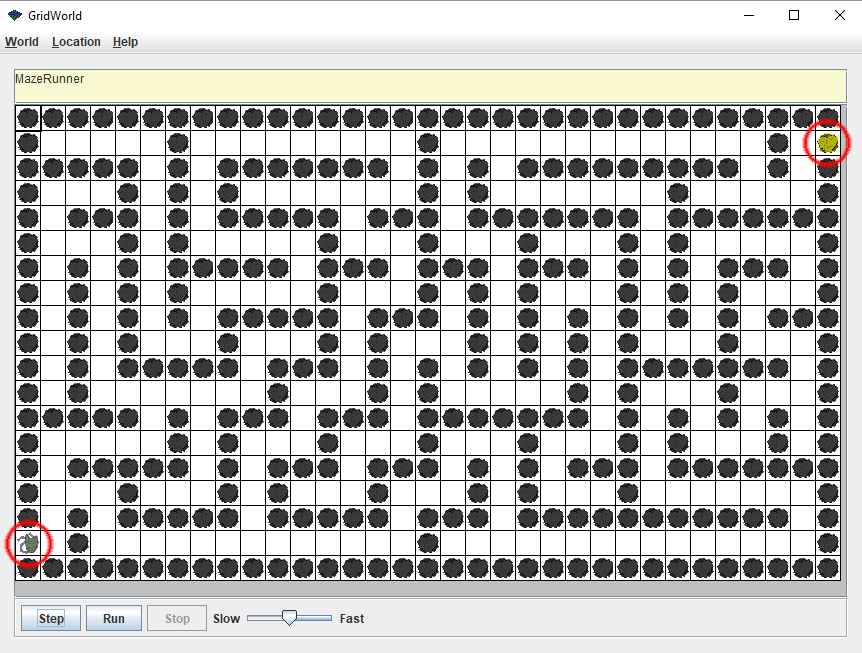
To create a unit test for the ***ArrayListStack***, right-click on the file and select Tools -> Create/Update Tests.

In the dialog box that comes up, set the following:

A test file will be created in the Test Packages folder, with default method bodies for each of the stack methods. You will have to edit those to write your test cases. Remove the test case for toString() – you do not have to test that method. Make sure your tests pass before moving on.

**Step 4 – Run to see the maze:**

Run the program and note that a graphical maze world appears and looks something like this (The maze is randomly generated each time the program runs, so yours won’t look exactly the same).

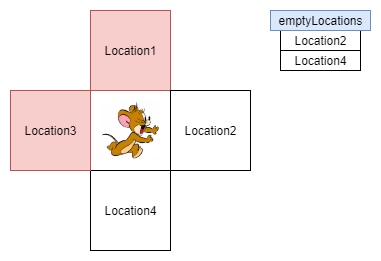
Note that there is a gray mouse on the left edge of the maze, and a piece of yellow cheese on the right-hand side of the maze. You need to help move the mouse to get to the piece of cheese.

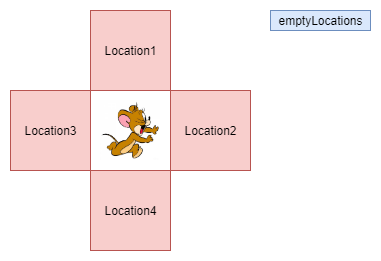
While you can do some weird things by clicking on the grid and using the menu items at the top, none of that will help you complete the assignment. When you have completed the coding for this assignment, you should be able to simply press the *Run* button at the bottom and see the mouse make its way to the cheese, sometimes taking wrong turns, but eventually getting there.

For now, pressing either the *Step* or the *Run* button will do nothing, because you haven’t implemented the ***move***() method in ***Mouse.java***.

**Step 5 – Make the mouse move**

You are at the final, exciting step. Now that you have a working stack, you can use it to store a series of breadcrumbs as the mouse moves throughout the maze. If you look at the top of the Mouse.java file, you will see that it is making use of your *ArrayListStack*, and passing it *Location* objects. Location objects store X, Y coordinates in the maze grid. Note that all of the maze world code is in a project that you don’t have access to – your Assignment2 project simply includes the jar file for the maze assets in order to get access to them.

You need to implement the *move*() method in Mouse.java, which involves figuring out whether there are any unvisited empty locations adjacent to where the mouse currently is. The move method is passed with a list of empty available locations as an argument, which are adjacent to where the mouse currently is (in other words, the hard work has been done for you). For example, if the mouse is sitting at a crossroad where it has visited the left and upper locations already, it would encounter this:

All you have to do is see if there are any empty locations in the list as in the argument of the move method. If there are empty locations, add the mouse’s current location to the crumbs list (to indicate the mouse has been there) and then randomly choose an empty location and make the mouse go there. If there aren’t any empty locations (your mouse is at a dead end!), then the mouse needs to back up to the last location visited (which should be at the top of the crumbs list).

How do you know what the mouse can do? You can find out by typing “this.” and seeing what methods NetBeans suggests. However, to complete the *move*() method, you only need to use the following two mouse methods: *getLocation*() and *moveTo*(Location).

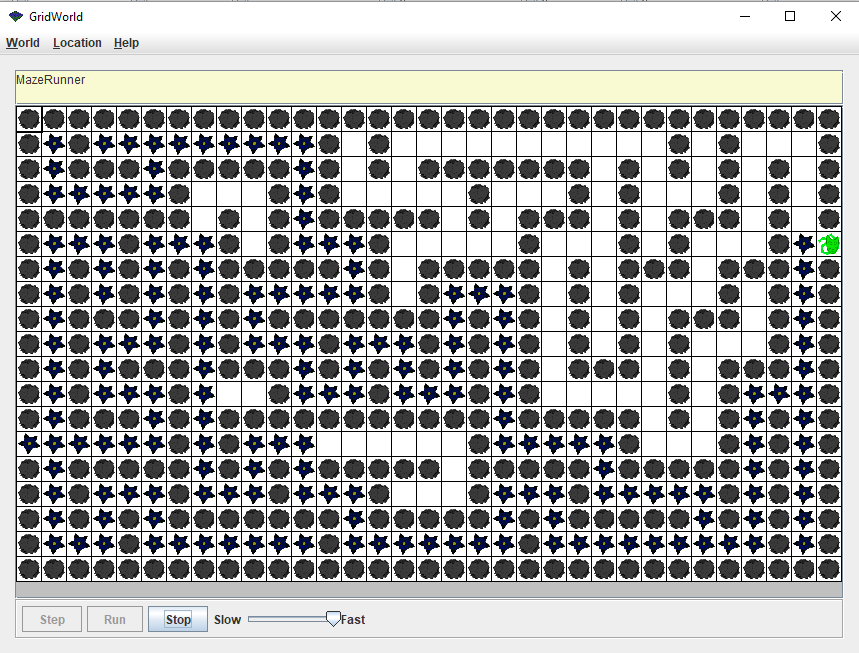
To randomly choose a location from the input *emptyLocations* argument, we can invoke:

Random rand = new Random();

int next = rand.nextInt(*emptyLocations*.size());

You don’t have to write an algorithm to explore the whole maze. The *move*() method gets called automatically for every step of the program (every time the user presses the *Step* button on the maze GUI, or it automatically gets called repeatedly until the cheese is found if the user presses the *Run* button. So, all you have to do is think about, for where the mouse is right now, knowing what empty locations are adjacent, what should the mouse do next.

Once the *move*() method is working correctly the mouse will navigate to the end of the maze:



**Requirements**

You must fulfill the following requirements in your implementation.

1. Create an ***ArrayListStack.java*** class that ***implements the StackADT*** given to you.
2. Generate the JavaDoc html files for the ArrayListStack class.
3. Create an ***ArrayListStackTest.java*** file that tests the class.
4. Complete the ***move***(…) method in ***Mouse.java***. (You do not have to write test cases for this method, but you can watch the mouse in the maze to see if your move method is working as expected)
5. Submit your project to Web-Cat.

**Files needed**

The files needed to get started are available on Canvas inside of a zipped project called ***MazeRunner.ZIP***. Download this, **but don’t unzip it**. Import the project into NetBeans. When you first import the project, it won’t run because you have not yet implemented the ***ArrayListStack.java*** and the ***move()*** method in Mouse.java.

**What you must test**

You must create and complete a test class for your ArrayListStack.java. You don’t need to write a test case for the constructor or for the toString method.

**Code coverage note**: Web-cat will run the JUnit tests that you submit and will basically count the percentage of lines of code in your classes that get executed as a result of your test cases. So, you want to try to write test cases that cause all lines of code in all branches of your code, to be executed. (So, if you have an ***if-else*** statement, write code in your test case that runs the branch of the ***if*** statement when the expression is true, and write other lines of code in your test case that will cause the else clause part of the if statement to be run).

**Submission**

Submit your assignment to Web-Cat (more details to follow).

**Grading**

1. TA Grade on quality of your JavaDoc and algorithmic clarity (20%)

2. Web-cat style testing\* (20%)

3. Web-cat unit testing (40%)

4. Web-cat coverage testing of your unit tests (20%)

*\*See the supplemental document on web-cat style testing for hints on how web-cat expects your code to be formatted.*

**Honor Code:**

Sign and submit the following declaration on Canvas. (Your code is submitted on web-cat, but this signed document must be submitted on Canvas – printing it, signing it and uploading a picture of it is sufficient).

Without this declaration, you will not receive a grade for Assignment 2

**I promise that I completed this assignment on my own and did not look at, borrow from or copy assignment code from a classmate or anyone who took the course before. All the code in this assignment (except what was given by the instructors) was written by me.**

**Describe any help you received on this assignment (from a professor or TA or the tutoring center):**

**Signature: Date:**