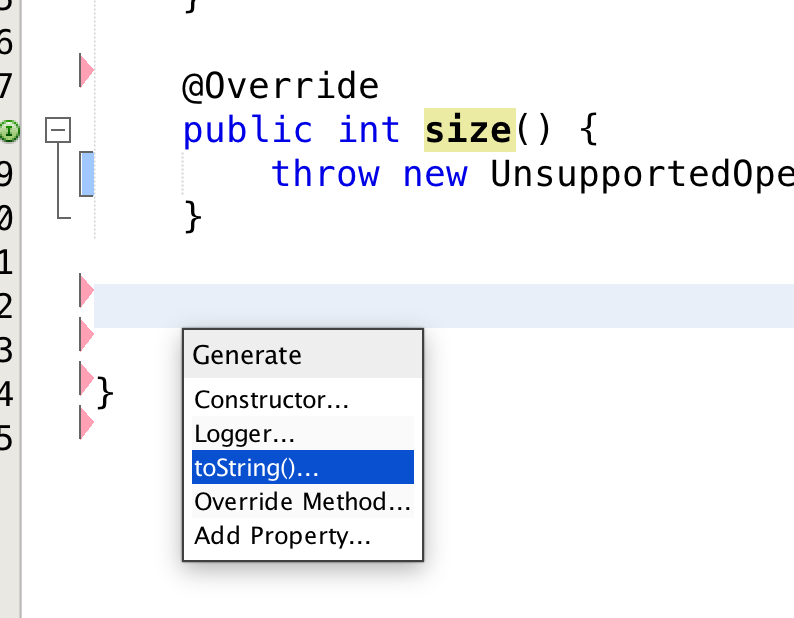
**Step 2 - Implement the ArrayList:**

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 your stack whatever you want.

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

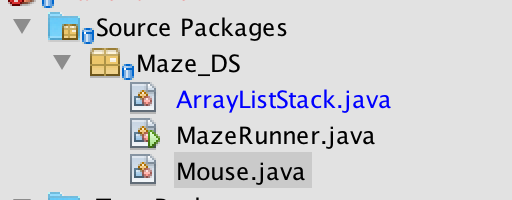
In each of the five methods that were created for you, you will need to remove the UnsupportedOperationException and put in the appropriate code to make the stack work.

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.

**Hint:** if you feel lost on what should go in the stack methods, look at the ArrayStackImplementation movie that Dr. Celine posted on YouTube (<https://youtu.be/qnTlxaAH4Rg>). It goes through the implementation using an Array. What you have to do is basically the same thing, except that you are using an ArrayList instead of an Array. Using an ArrayList is easier, because you can use the ArrayList’s built-in size method and the ArrayList takes care of capacity issues.

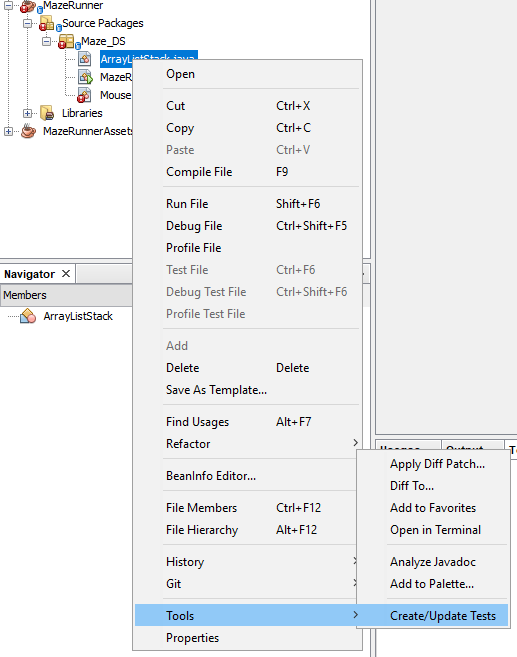
Make sure that you write JavaDoc comments for your ArrayListStack class and generate the JavaDoc files.

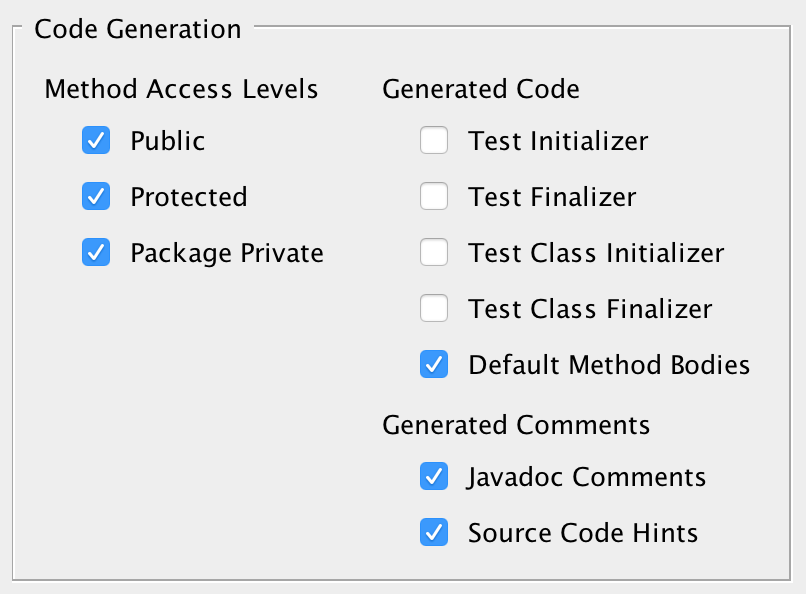
Once you have the ArrayListStack implemented, the red errors associated with your source code should all go away:

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.

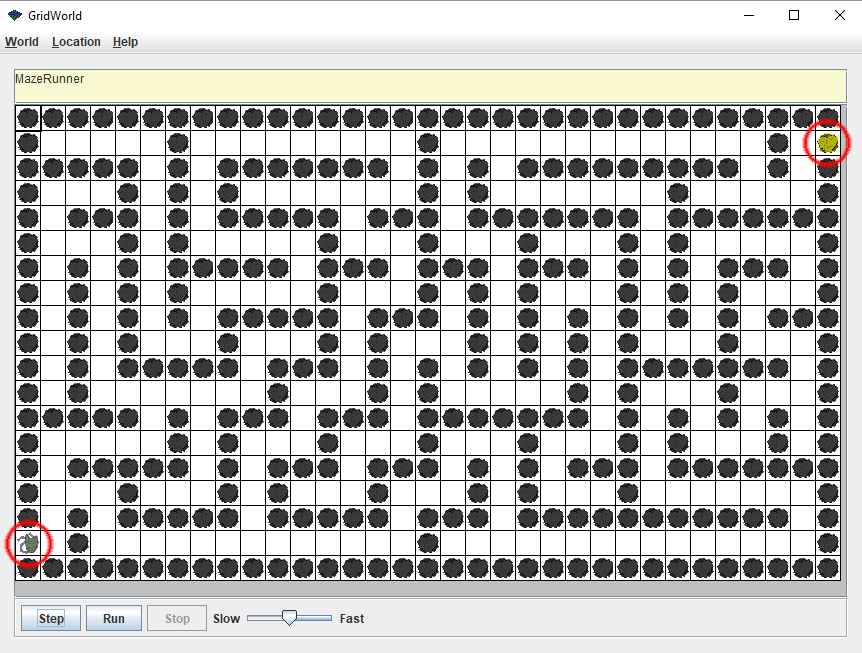


In the dialog box that comes up, set the following:

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).

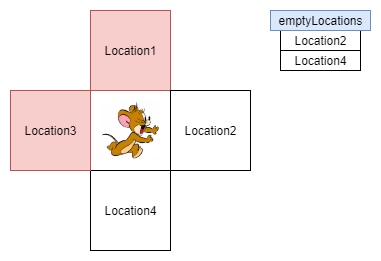
Note that there is a gray mouse on the left edge of the maze, and a piece of yellow cheese on the right-hand side of the maze. You need to help move the mouse to get to the piece of cheese.

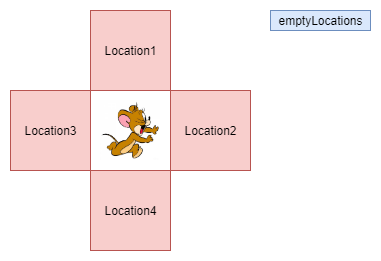
While you can do some weird things by clicking on the grid and using the menu items at the top, none of that will help you complete the assignment. When you have completed the coding for this assignment, you should be able to simply press the *Run* button at the bottom and see the mouse make its way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

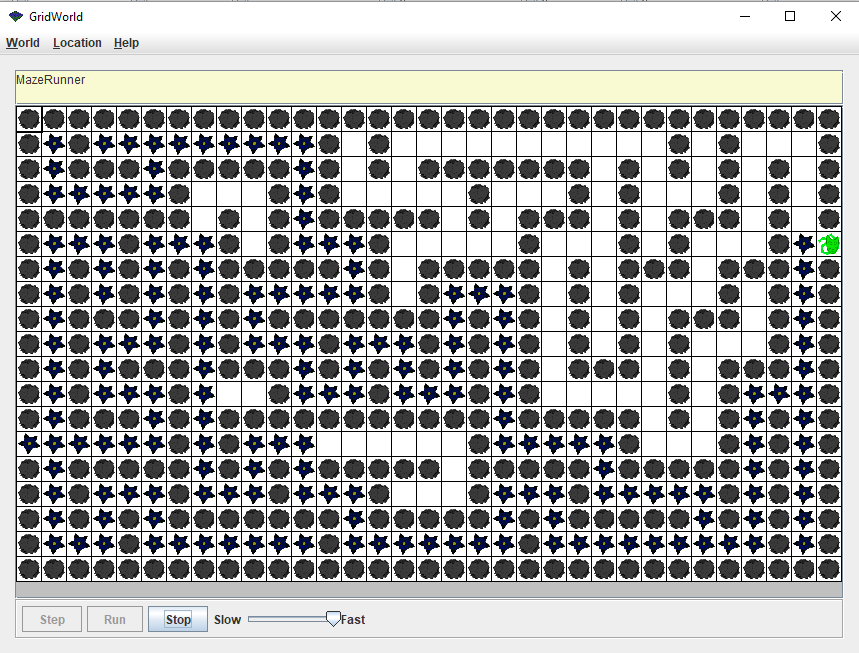
You need to implement the move() method in Mouse.java, which involves figuring out whether there are any unvisited empty locations adjacent to where the mouse currently is. The move method is passed a list of empty locations that are adjacent to where the mouse currently is (in other words, the hard 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 you’ve been passed. 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). You’ll be able to tell when this is the case because the emptyLocations list will be empty.

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).

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)

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).