CS1027: Assignment 2

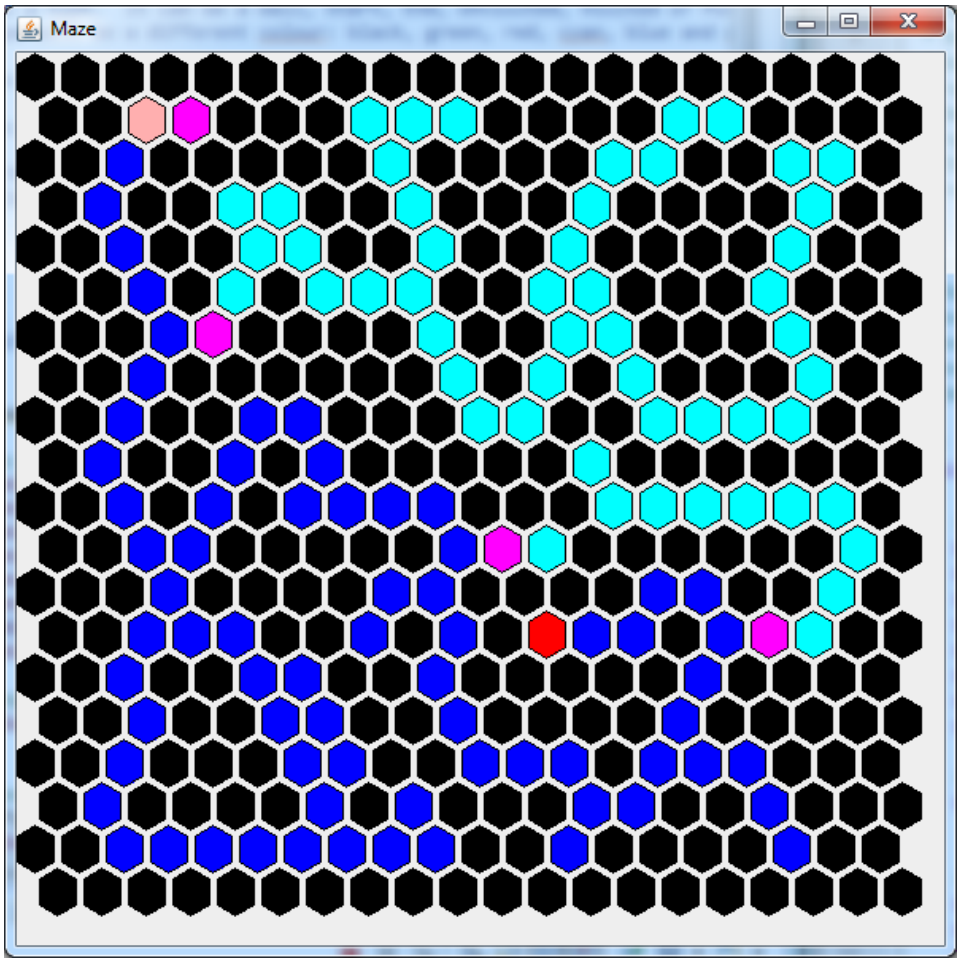
Due: Thursday, July 13th, 11:55pm.
Weight: 10%

Purpose:
To gain experience with

- Stacks
- Working with complex classes
- Exceptions

Task:

Mazes can be solved by computers using a variety of algorithms. A simple approach is to start at the beginning and search through all open spaces one at a time until the end is found. You will create a program that searches for the end of a maze made up of hexagonal tiles using a stack to keep track of tiles yet to be checked. Here is a link to an incomplete example: http://i.imgur.com/vNfPlOd.gif



Functional Specifications:

For this assignment you are given a number of classes to use, and you will also use some of the classes from Lecture. You will create a class `MazeSolver` that has a main method only, which uses these classes to implement the maze solving algorithm given below.

Classes from Lecture/Website you will need:

- `ArrayStack.java`
- `EmptyCollectionException.java`
- `StackADT.java`

Provided Classes for this Assignment:

- `Maze.java` - A class representing a Maze made up of Hexagon tiles. Opens in a graphical window.
- `Hexagon.java` - A class representing the Hexagon tiles in a Maze window.
- `HexComponent.java` - A superclass of Hexagon representing the graphical element
- `HexLayout.java` - A class allowing the Maze to lay out the Hexagon tiles correctly
- `UnknownMazeCharacterException.java` - An Exception to do with reading in the Maze from a file.
- `InvalidNeighborIndexException.java` - An Exception to do with accessing the neighbor of a Hexagon that cannot exist.

The main classes you will be working with are Maze and Hexagon. The Maze class is initialized with a file representing the layout of the maze: `maze0.txt, maze1.txt, maze2.txt, maze3.txt, maze4.txt`

The API for the provided classes will orient you to their methods. Look into ***"doc/index.html"*** in the zip you downloaded for the javadocs. In other words, **BEFORE YOU START**, familiarize yourself with the classes, specifically the Maze and Hexagon classes. Be sure to read the description of the class near the top of the page and the available methods provided by the class.

Each Hexagon tile can be one of a Wall (black), Start (green), End (yellow) or Unvisited (cyan) when we build the maze at the beginning, and as we visit tiles during they solving process, they can become pushed (magenta) or processed (blue). The tiles will display in different colours so you can track your MazeSolver progress. The start tile will turn pink when it is processed and the end tile will turn red when it is found (processed).

High Level Algorithm for `MazeSolver`:

- Create maze object
- *Try* to open maze file
- Reference starting Hexagon from the maze
- Create Stack and *push* starting Hexagon
- Make sure bookkeeping variables are created (stepCounter and hexagonOnStack counter)
- While stack is not empty
    - Pop
    - Increase step counter, decrease hex counter
    - If Hexagon is end tile, we need to do special stuff
    - Otherwise
        - For each of the six possible neighbours of current Hexagon
            - If neighbour exists, is not a wall, was not visited
                - Push neighbor on the stack and do bookkeeping (counters)
    - Set current Hexagon as *processed*
    - Update maze window with *repaint()*
- Once we are done
    - Say if we found the end
    - Number of steps that it took to finish
    - Number of tiles still on the stack

Analysis

You must run your code on all 5, however, you will notice that maze1 and maze3 have interesting things happening. Write a brief description in a separate .txt file describing the peculiarities in these three mazes. This can be in point form, and very brief, however it does need to be readable/understandable.

Exceptions

Your code must correctly handle thrown exceptions. To handle the exceptions you need only to inform the user through a console print statement what specifically has happened. These handlers must be specific (rather than one catch block for Exception itself).

Command Line Argument

You must read the Maze file from the command line. The command line is what computers used to use to run programs rather than having graphical windows, and all systems still have the functionality available (like through the Terminal application on Macs, or the Command Prompt (cmd) application on Windows). The user could run the MazeSolver program with the following command from a command line (for example): `java MazeSolver maze1.txt`

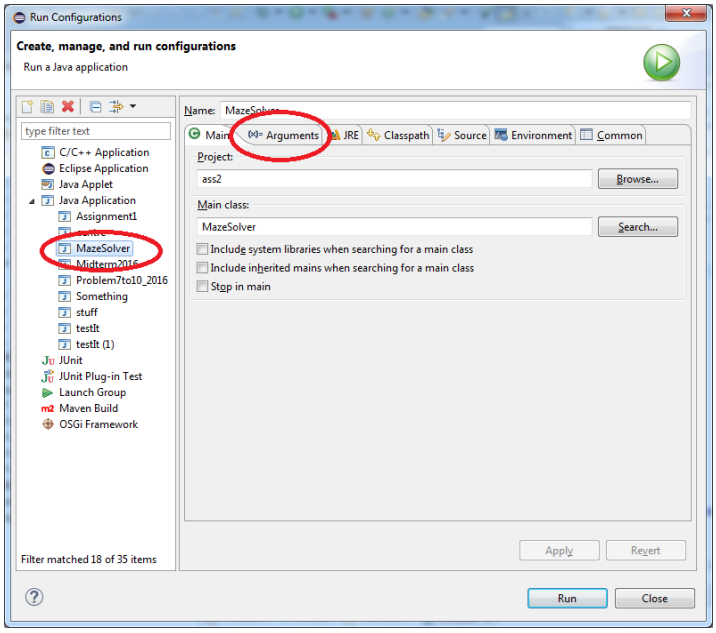
If you have ever wondered what the "`String[] args`" thing meant in the main method header, this is what it is for; It allows us to read in the text supplied on the command line. The args array of Strings will store any of the text supplied by the user, that is, any tokens following the application name (MazeSolver in the above example).

To read a single argument, we look in args[0], but first we want to check that the user has entered something. The following code example could be the beginning of your MazeSolver.java file. It will check the length of the args array and thrown an exception if there is not an argument provided. Then it will store a reference to the String using the reference variable mazeFileName.

```
public class MazeSolver {
      public static void main (String[] args) {
            try{
                  if(args.length<1){
                        throw new IllegalArgumentException("No Maze Provided");
                  }
                  String mazeFileName = args[0];
                  //...
```

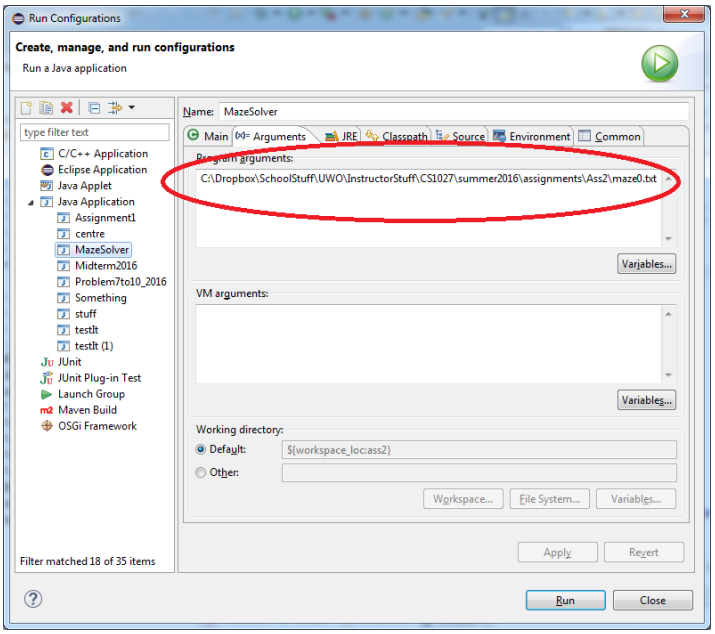
Setting up a Command Line Argument when running your program in Eclipse

To get Eclipse to supply a command line argument to your program when it runs, you will need to modify the "Run Configurations". On my computer it's located in "Run"->"Run Configurations". Something like the following should pop up.



Be sure the "Java Application->MazeSolver" is the active selection on the left-hand side.

Select the "Arguments" tab.



Enter the filename and location in the "Program arguments" text box. In my case I have it in "C:\Users\James\Dropbox\SchoolStuff\UWO\InstructorStuff\CS1027\summer2016\assignments\Ass2\maze0.txt", however you may have it somewhere completely different and that is to be expected.

If you are on Mac/Linux you will likely have something like "/Users/SomeName/Desktop/Ass2/maze2.txt".

TO SUBMIT:

- MazeSolver.java (IT MUST BE THE .java; DO NOT SUBMIT THE .class FILE)
- A **.**txt file with your brief analysis for maze1 and maze3.

FAQ:

- **Q:** I don't know where to start.
  - **A:** Did you read the whole assignment details?
- **Q:** I still don't know where to start.
  - **A:** Go over the high level description of the algorithm. It really does into the details of solving the problem**.** You should also make sure you're comfortable with the provided classes – read the documentation provided.
- **Q:** Does my .txt give enough details?
  - **A:** Probably, it's supposed to be brief.
- **Q:** Should I comment X?
  - **A:** Probably.
- **Q:** There is a bug in my code!
  - **A:** Look at the debugging lecture slides. There are a lot of tips in there.
- **Q:** Did I catch *enough* exceptions?
  - **A:** No idea. Keep trying to figure out how to break your code. Make sure you have *at least* 4 being dealt with.
- **Q:** Can I change the specifications, even if it's to make it 'better'?
  - **A:** NO! Not even if you invent the best maze solver in the world.
- **Q:** When catching exceptions, does order matter?
  - **A:** Yes, it does actually. Remember subclasses are also the same type of parent classes.
- **Q:** I swear I did everything right, but for some reason my files won't open!
  - **A:** This isn't uncommon.
    - If you're using eclipse, try putting the .txt files in the project directory (the same folder containing the *bin* folder, *src* folder.
    - If you're running from command line, try putting the files in the *src* folder.
- **Q:** I don't know how to do X?
  - **A:** Try going to my favorite website: www.google.ca and then typing X into the big text box.
- **Q:** Can I email the TA or professor with questions?
  - **A:** Yes, but you should really check out my favorite website (above).

Non-functional Specifications:

1. Include brief comments in your code identifying yourself, describing the program, and describing key portions of the code.
2. Assignments are to be done individually and must be your own work. Software may be used to detect cheating.
3. Use Java coding conventions and good programming techniques, for example:
   i. Meaningful variable names
   ii. Conventions for naming variables and constants
   iii. Use of constants where appropriate
   iv. Readability: indentation, white space, consistency
   v. private vs public

Make sure you attach your files to your assignment; **DO NOT** put the code inline in the textbox. **DO NOT SUBMIT YOUR *.class* FILES. IF YOU DO THIS, AND DO NOT ATTACH YOUR *.java* FILES, YOU WILL RECEIVE A MARK OF ZERO!**

What You Will Be Marked On:

1. Functional specifications:
   - Does the program behave according to specifications?
   - Does it run with the main program provided?
   - Are your classes created properly?
   - Are you using appropriate data structures?
   - Is the output according to specifications?
2. Non-functional specifications: as described above
3. Assignment submission: via OWL assignment submission