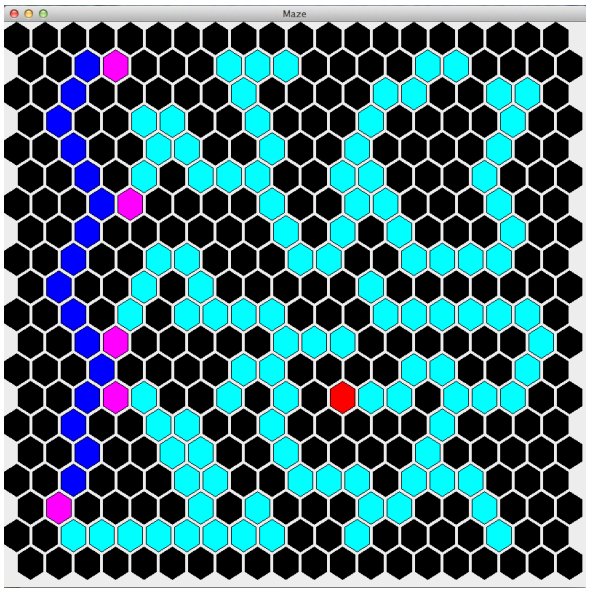
CS1027b Computer Science Fundamentals II

Assignment 2

Description

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.



Due Sunday ~~February 8th~~ February 15th via Owl.

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 Lectures 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 neighbour of a Hexagon that cannot exist.

These can be found in: 1027Asn2ProvidedCode.zip

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: maze1.txt, maze2.txt.

The API for the provided classes will orient you to their methods: API Reference

Each Hexagon tile can be one of a Wall (black), Start (green), End (red) 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`'s progress. The end tile will turn gold when it is found (processed).

Classes to Submit

- `MazeSolver.java`

Algorithm for `MazeSolver`

- Create a maze object reference
- Try to open a new maze from one of the maze files provided
- (The maze should now be initialized)
- Create a Hexagon reference and get the start Hexagon tile from the maze
- (Using an array stack of Hexagons, we will solve the maze.)
- Create the stack and push the starting tile.
- Create a boolean variable that we will use to keep track of whether we have found the end
- Create a reference for the current Hexagon that we will process
- Create a counter to keep track of how many steps it takes us to solve the maze
- Now, while our stack is not empty, and while we have not found the end
    - pop the top of the stack to get the current hexagon tile
    - increment the step counter
    - if the current hexagon tile is an end tile, make the found end boolean variable true
    - otherwise, for each of the possible six neighbours of the current tile
        - if the neighbour exists and is unvisited
            - push the neighbouring hexagon tile to the maze solving stack and set the neighbouring hexagon tile to be a pushed tile type
    - set the current hexagon tile to "processed", now that we are done with it
    - Note: each time through we need to update the maze window with a call to repaint()
- Once we have searched the maze using the above algorithm, print out the following using a System.out.println or .format:
    - If the end was found, or if it was not found
    - The number of steps that it took to finish
    - How many tiles were still on the stack

Implementation notes

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), and you should use the finally block to close the file resource if open.

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 application on Windows). The user could run the MazeSolver program with the following command from a command line (for example): `java MazeSolver maze1.txt`

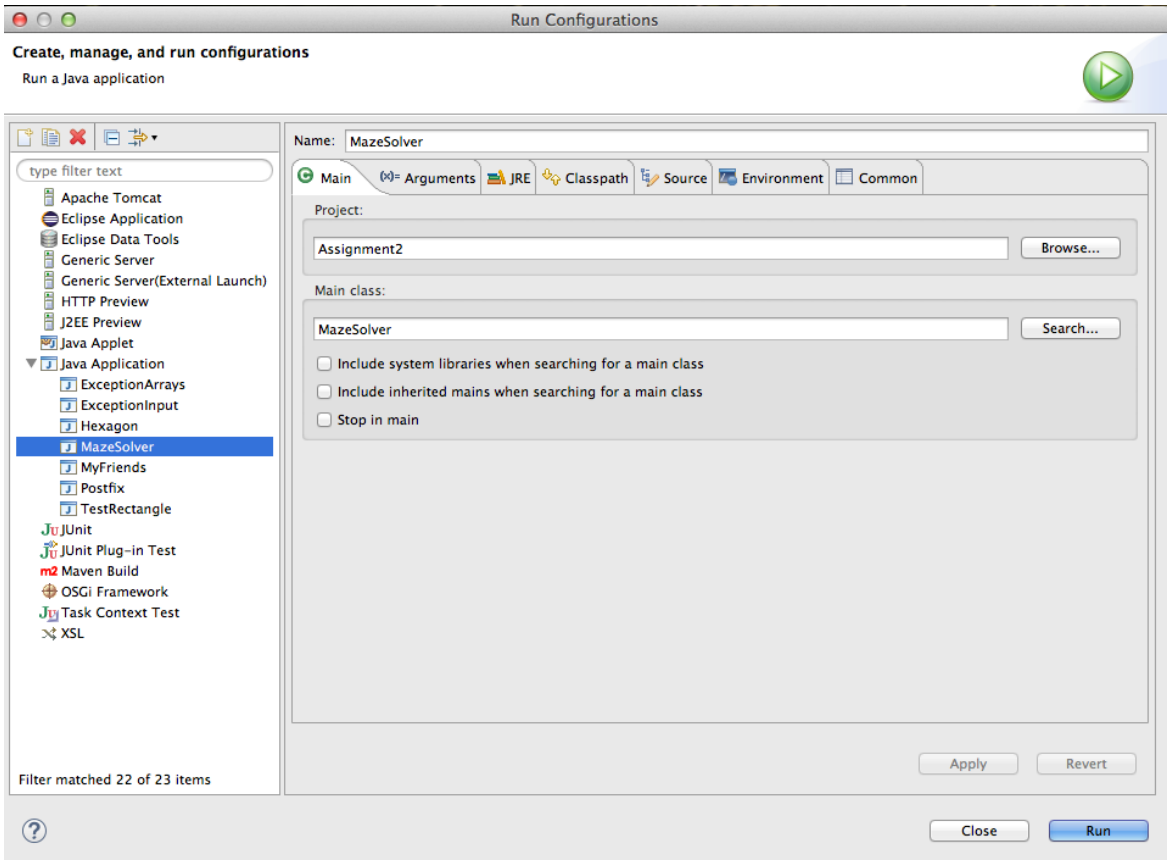
If you have ever wondered what the "`String[] args`" thing meant in the main method header, this is what it is for; It allows us to read in the text supplied on the command line. The `args` array of Strings will store any of the text supplied by the user, that is, any tokens following the application 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("Please provide a Maze file as a command line argument");
                        }
                        String mazeFileName = args[0];
                        //...
```

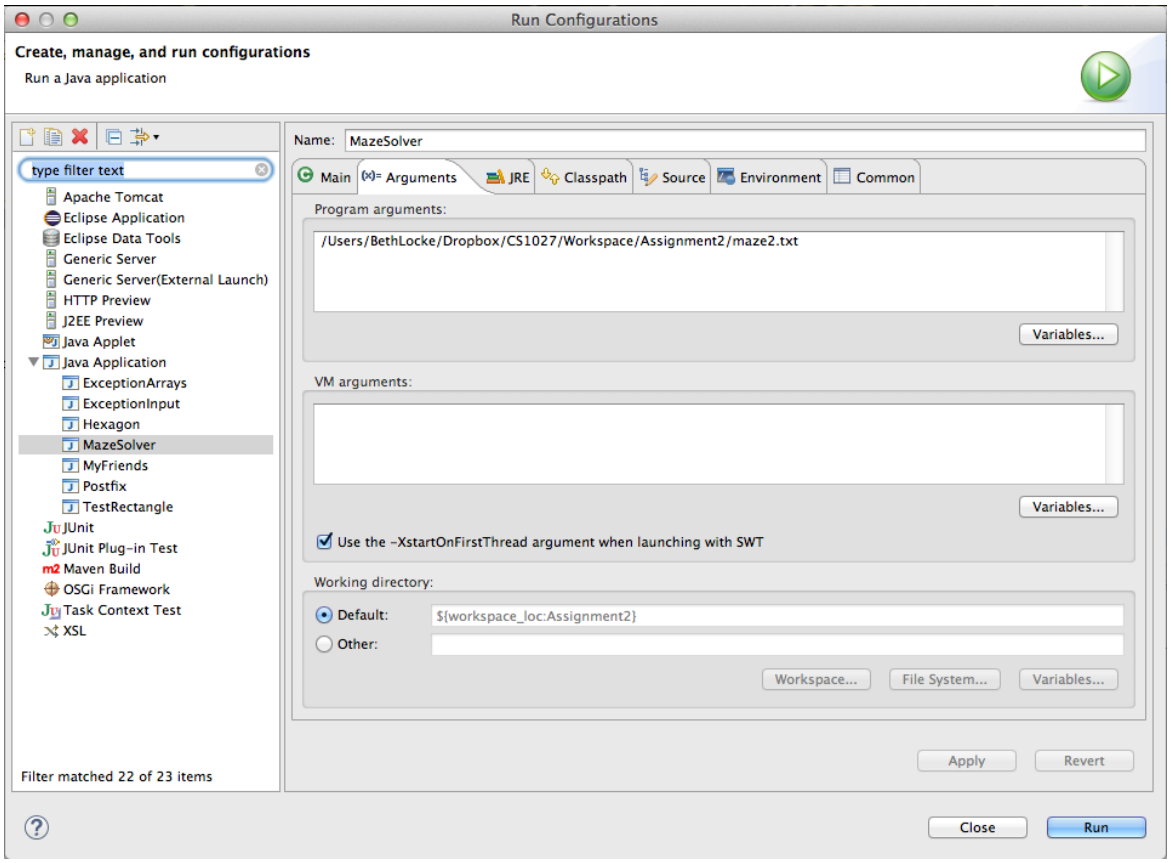
Setting up a Command Line Argument when running your program in Eclipse

To get Eclipse to supply a command line argument to your program when it runs, you will need to modify the "Run Configurations". Open the "Project"->"Run Configurations" menu item. 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 in the "Program arguments" text box.

Non-functional Specifications

- Your program has to be compilable under Eclipse.
- Use Javadoc comments for each class and method. All significant variables must be commented.
- Use Java conventions and good Java programming techniques (meaningful variable names, conventions for variable and constant names, etc). Indent your code properly.
- Remember that assignments are to be done individually and must be your own work.