Com S 227
Fall 2019
Assignment 3
250 points

Due Date: Thursday, Nov 7, 11:59 pm (midnight)

5% bonus for submitting 1 day early (by 11:59 pm Nov 6)

10% penalty for submitting 1 day late (by 11:59 pm Nov 8)

No submissions accepted after Nov 8, 11:59 pm

General information

This assignment is to be done on your own. See the Academic Dishonesty policy in the syllabus, http://www.cs.iastate.edu/~cs227/syllabus.html#ad **, for details.**

You will not be able to submit your work unless you have completed the *Academic Dishonesty policy acknowledgement* on the Homework page on Canvas. Please do this right away.

If you need help, see your instructor or one of the TAs. Lots of help is also available through the Piazza discussions.

Overview

In this homework, you need to write two classes (namely Maze, MazeCell), whose skeleton codes are given under package *hw3* in the zip file. The classes under packages *maze* and *strategy* you must NOT modify. You may modify RunSearcher under the *ui* package to run the search for different mazes or to test your code. This is the class that you use to run the whole program. You don't submit RunSearcher. The *forwardSearch* method in the RunSearcher class should be quite interesting. If you are curious, you may want to read it at your leisure to figure out how the search actually works.

The classes under *strategy* are different comparators that allow a cell's neighbors to be searched in different orders, and are used by the *forwardSearch* method. They are included here for completeness and for fun, but you don't have to read them. ***The main purpose of this assignment is to help you practice arrays (especially 2-D arrays), loops, and ArrayLists***. So you should focus only on package *hw3*.

The *maze* package contains classes mostly for graphical user interface. Check them out only if you are curious. Status in package *maze* represents an enum type, which is fairly easy to use. Here are some examples:

```
        private Status status; // use Status as type
```

or,

```
        status = Status.WALL; // set status to be a WALL
```

or,

```
        if(status==Status.Wall) // check if status is WALL
```

There are two classes in hw3 that you need to complete. Descriptions about these classes are given in the java doc. *Make sure to read the javadoc carefully so as to follow exactly the specifications*.

Sample usage

A good way to think about the specification is to try to write some simple test cases and think about what behavior you expect to see. Some basic tests on MazeCell and the Maze class are given in a class called **MazeAndMazeCellTest.java** in package *ui*. You should expand the test cases to thoroughly test your classes.

Suggestions for getting started

*Smart developers don't try to write all the code and then try to find dozens of errors all at once; they work **incrementally** and test every new feature as it's written. Here is a rough guide for how an experienced coder might go about creating a class such as this one:*

1. Import the given zip file to Eclipse. See section **Importing the sample code** below for instructions.

2. Start on the MazeCell class first. Look at each method. Mentally classify it as either an *accessor* (returns some information without modifying the object) or a *mutator* (modifies the object, usually returning `void`). The accessors will give you a lot of hints about what instance variables you need. Write your own test codes, or use and expand **MazeAndMazeCellTest.java**, to make sure your methods are implemented correctly.

3. Similarly, work on the Maze class.

Testing and the SpecChecker

As always, you should try to work incrementally and write tests for your code as you develop it.

We will provide a basic SpecChecker, but **it will not perform any functional tests of your code**. At this point in the course, you are expected to be able to read the specfications, ask questions when things require clarification, and write your own unit tests. Since the test code is not a required part of this assignment and does not need to be turned in, **you are welcome to post your test code on Piazza for others to check, use and discuss.**

The SpecChecker will verify the class names and packages, the public method names and return types, and the types of the parameters. If your class structure conforms to the spec, you should see a message similar to this in the console output:

```
x out of x tests pass.
```

This SpecChecker will also offer to create a zip file for you that will package up the two required classes. Remember that your instance variables should always be declared **`private`**, and if you want to add any additional "helper" methods that are not specified, they must be declared **`private`** as well.

> See the document "SpecChecker HOWTO", which can be found in the Piazza pinned messages under "Syllabus, office hours, useful links" if don't remember how to import and run a SpecChecker.

Importing the sample code

1. Download the zip file. You don't need to unzip it.
2. In Eclipse, go to File -> Import -> General -> Existing Projects into Workspace, click Next.
3. Click the radio button for "Select archive file".
4. Browse to the zip file you downloaded and click Finish.

More about grading

This is a "regular" assignment so we are going to read your code. Your score will be based partly (about a third) on functional tests that we run and partly on the grader's assessment of the quality of your code. Are you doing things in a simple and direct way that makes sense? Are you

defining redundant instance variables? Some specific criteria that are important for this assignment are:

- Use instance variables only for the "permanent" state of the object, use local variables for temporary calculations within methods.
    - You will lose points for having lots of unnecessary instance variables
    - All instance variables should be **`private`**.
- **Accessor methods should not modify instance variables**.
- Avoid code duplication.
- Internal (//-style) comments are normally used inside of method bodies to explain *how* something works, while the Javadoc comments explain *what* a method does. Use internal comments where appropriate to explain how your code works. (A good rule of thumb is: if you had to think for a few minutes to figure out how something works, you should probably include a comment explaining how it works.)

See the "Style and documentation" section below for additional guidelines.

Style and documentation

Roughly 15% of the points will be for documentation and code style. Here are some general requirements and guidelines:

- Each class, method, constructor and instance variable, whether public or private, must have a meaningful and complete Javadoc comment. Class javadoc must include the **`@author`** tag, and method javadoc must include **`@param`** and **`@return`** tags as appropriate.
    - Try to state what each method does in your own words, but there is no rule against copying and pasting the descriptions from this document.
    - Run the javadoc tool and see what your documentation looks like! You do not have to turn in the generated html, but at least it provides some satisfaction :)

- All variable names must be meaningful (i.e., named for the value they store).
- Your code should not be producing console output. You may add **`println`** statements when debugging, but you need to remove them before submitting the code.
- Try not to embed numeric literals in your code. Use the defined constants wherever appropriate.
- Use a consistent style for indentation and formatting.
    - Note that you can set up Eclipse with the formatting style you prefer and then use Ctrl-Shift-F to format your code. To play with the formatting preferences, go to Window->Preferences->Java->Code Style->Formatter and click the New button to create your own "profile" for formatting.

If you have questions

For questions, please see the Piazza Q & A pages and click on the folder **`assignment3`**. If you don't find your question answered, then create a new post with your question. Try to state the question or topic clearly in the title of your post, and attach the tag **`assignment3`**. *But remember, do not post any source code for the classes that are to be turned in.* It is fine to post source code for general Java examples that are not being turned in, and **for this assignment you are welcome to post and discuss test code**. (In the Piazza editor, use the button labeled "code" to have the editor keep your code formatting. You can also use "pre" for short code snippets.)

If you have a question that absolutely cannot be asked without showing part of your source code, change the visibility of the post to "private" so that only the instructors and TAs can see it. Be sure you have stated a specific question; vague requests of the form "read all my code and tell me what's wrong with it" will generally be ignored.

Of course, the instructors and TAs are always available to help you. See the Office Hours section of the syllabus to find a time that is convenient for you. We do our best to answer every question carefully, short of actually writing your code for you, but it would be unfair for the staff to fully review your assignment in detail before it is turned in.

Any posts from the instructors on Piazza that are labeled "Official Clarification" are considered to be part of the spec, and you may lose points if you ignore them. Such posts will always be placed in the Announcements section of the course page in addition to the Q&A page. (We promise that no official clarifications will be posted within 24 hours of the due date.)

What to turn in

Note: You will need to complete the "Academic Dishonesty policy questionnaire," found on the Homework page on Canvas, before the submission link will be visible to you.

Please submit, on Canvas, the zip file that is created by the SpecChecker. The file will be named **`SUBMIT_THIS_hw3.zip`**. and it will be located in the directory you selected when you ran the SpecChecker. It should contain one directory, **`hw3`**, which in turn contains two files, **`Maze.java`**, **`MazeCell.java`**. Please LOOK at the file you upload and make sure it is the right one!

Submit the zip file to Canvas using the Assignment 1 submission link and verify that your submission was successful. If you are not sure how to do this, see the document "Assignment Submission HOWTO" which can be found in the Piazza pinned messages.

We recommend that you submit the zip file as created by the specchecker. If necessary for some reason, you can create a zip file yourself. The zip file must contain the directory **hw3**, which in turn should contain two java files. You can accomplish this by zipping up the **src** directory of your project. The file must be a zip file, so be sure you are using the Windows or Mac zip utility, and NOT a third-party installation of WinRAR, 7-zip, or Winzip.