COP 3330
Homework 2

Out: 1/29/12 (Sunday)
Due: 2/10/12 (Friday) at 11:55 PM WebCourses time
Late submissions accepted until 2/12/12 (Sunday) at 11:55 PM (with penalty)

Objective

1. To help you become comfortable with using the basic control structures in Java.
2. To familiarize you with generating and using random numbers.
3. To familiarize you with using multi-dimensional arrays in Java.

Code

You may discuss high-level ideas of how to solve these problems with your classmates, but do not discuss actual source code (except for boiler-plate code such as opening a file for reading, reading user input, etc.). All source code should be yours and yours alone. **Do not:**

- Share your source code with anyone
- Ask anyone to share his/her source code with you
- Ask/pay a programming forum/community to write your homework for you
- Pair program
- Do anything else that you know is deceptive, dishonest, or misleading in any way

Problem: Tallahassee Smith and the Cave of Ruin

In this problem, you will implement a class *Adventure*, which will simulate a simple adventure game set in a cave. Your program will be responsible for generating the cave, and taking input from the user to move a player character around it. Depending on the path taken, players can win or lose the game, and your program must indicate this.

Your program should create a 10x10 grid, representing the cave. The player character is an extremely overeducated but intrepid two-dimensional adventurer named Major Charles Stafford "Tallahassee" Smith III, BA, LLB, PhD, COD, RSVP, ASAP.

Tallahassee Smith is represented by a 'P' in the game world, and will start in the upper left cell of the grid. He can move into adjacent cells above, below, left, or right of the current cell. The goal is to move him to the treasure in the lower right corner.

For dramatic reasons, our hero must brave numerous obstacles and dangers as he makes his way to the treasure (represented by a 'T'), which is the only possible way to pay off his massive student loans. More concretely, each cell of the grid can be of three types:

1. **Empty, indicated by a . (period)**: This means the player can move into this cell without any problems. The start square will always be an empty cell.
2. **Blocked, indicated by an X**: This indicates an area where there has been a cave-in. The player cannot move into such a cell.
3. **A pit, indicated by a ***: This represents a fatal trap. The player can move into such a cell, but on doing so he is instantly killed, and the game ends.

Your program must do the following:

1. **Generate a random cave:** You must create a 10x10 grid representing the cave. Initially, the upper left and lower right cells will always contain 'P' (the player) and 'T' (the treasure) respectively. Each of the **remaining cells** should be chosen to be a pit (with 5% probability), an obstacle (10% probability), or empty (85% probability).
2. **Print the game state after each turn:** Display the current state of the cave after each turn. This shows the current position of the player (by placing the 'P' in his current cell). Everything else is stationary, so the rest of the board remains the same across turns.
3. **Update the game board after each turn:** Your program must prompt the user for a choice (Up, Down, Left, Right) and attempt to move the player in that direction. It must account for the following possibilities:
    - **Moving into an empty cell**: The player moves into the cell and out of his current cell.
    - **Moving into a blocked cell**: This is not possible, so the player should stay where he is. Your program should print a message saying that Tallahassee Smith cannot move in that direction.
    - **Moving into a pit**: This should result in a Game Over message, and the game should end.
    - **Trying to move into a wall**: If the player is at one of the edges, and tries to move towards the edge, he will run into the wall of the cave. This should be treated the same way as blocking – the player stays where he is, and a message is printed saying that he cannot move in that direction.

Input (Standard input)

Your program doesn't use input until the game board has been generated. Once that is done, the game prompts the user to move the character on each turn. You can handle this by prompting the user to select the options by number: 1 – Up, 2 – Down, …, etc, or having them type in letters for input. Make sure to have your move prompt tell the user how you are expecting them to interact with your program.

Output (Standard output)

The game board and any messages should be printed to standard output.

Sample I/O

Watch the video for an example of a working game. Since the graders will essentially play the game to grade it, you are free to be somewhat creative, but make sure you include all the necessary functionality. You can find the video here:

https://www.youtube.com/watch?v=9KKFCNR24mg

Deliverables

Submit the source code file (**Adventure.java**) over Webcourses as an attachment. **In particular, *do not* submit any .class files!!! This will result in 0 credit for the assignment.** You must send your source files as an attachment using the "Add Attachments" button. Assignments that are typed into the submission box will **not** be accepted.

Restrictions

Your program must compile using Java 6.0 or later on the command prompt. It's okay to develop your program using the IDE of your choice, but the TAs will use the command prompt to compile your code and run the programs. Your code should include a header comment with the following information:

Your name
Course number, section number
Description of the code

You **WILL** lose credit if this information is not found in the beginning of each of your programs. You should also include **inline comments** to describe different parts of your program.

Execution and Grading Instructions

1. Download the source *.java* file and place in a folder.
2. Check the source code of each program to make sure it contains header comments, inline comments and reasonable use of variable names.
3. Run the game and test to see if all the required functionality is present.
4. If all required functionality is present and works correctly, give full credit for execution. Otherwise, give partial credit based on the situation.

Points Breakdown

Total Points: 100

1. Adventure.java
    - Execution: 85
    - Documentation: 15