Description

For this Homework, you will be implementing two classes: Maze and MazeNavigator. Your solution will simulate a maze navigation game where users must find a treasure chest.

Note: 5 points of your Challenge grade is based on Coding Style. You will need to follow the standards described on Brightspace . Use the "Run" button to check your Coding Style without using a submission.

Instructions

Note: As has been the case in every CS 18000 assignment, Scanner declaration and instantiation is only permitted once in the main method. The starter code does this for you. Additionally, **you are not permitted to use modifiable static fields in your implementation**. Failure to follow these rules will lead to failed test cases.

This program will navigate a player through a maze to find a treasure chest. The player can move one of four directions: Up, Down, Left, or Right. We will model the maze as a two dimensional boolean array. Here's one maze that may be used in a game:

```
[ true | true ]

[ true | true ]
```

```
[ true | true | true | true]

[ true | true | true | true]

[ true | true | true | true]

[ true | true | true | true]
```

A 2x2 array with four possible positions: **[0,0]**, [0,1], [1,0], [1,1]. A player starts in the [0,0] position, or top left of the maze. They can choose to move in any direction. **Moves that step outside the bounds of the array will wrap around to the other side (Exception Handling, ArrayIndexOutOfBoundsException**). For example, a player that decides to move up from the starting position in our sample maze would then move to **[1,0]**, the lowest possible position in that column.

Note: The previous paragraph mentioned a very important idea. If you do not remember to handle moves that go outside of the bounds of the array, your solution will have hard-to-find bugs. Be careful!

Of course, **boolean maze positions can also be false**. If a particular position is false, it cannot be occupied by the player. If a player tried to move into one, it would be an invalid move. The player would not move and would be prompted to try again. This is the maze part of the game. Here's another maze:

```
[ true | false | true | true ]

[ true | false | false | false ]

[ true | true | true | true ]

[ false | false | false | false ]
```

Here we have a new maze. The player can only move into spaces/positions that are true. Assume that the player is just starting out at [0,0]. If the treasure is at [2,2], **what moves would the player need to make?** Here's a series of moves:

- Down - Player moves to [1,0]
- Down - Player moves to [2,0]
- Right - Player moves to [2,1]
- Right - Player moves to [2,2]

The treasure was found! Now, the player will not know the configuration of the maze or the location of the treasure while they are playing the game. They will need to guess the correct path as they attempt different moves. The two classes you create will facilitate this process.

Maze

This class contains attributes associated with a specific maze.

Fields

Name	Type	Modifiers	Description
Maze	boolean[][]	Private	The two-dimensional maze.
playerRow	int	Private	The integer value corresponding to the player's current row.

playerColumn	int	Private	The integer value corresponding to the player's current column.
treasureRow	int	Private	The integer value corresponding to the treasure's current row.
treasureColumn	int	Private	The integer value corresponding to the treasure's current column.

Constructor

Parameters	**Modifier**	**Description**
boolean[][] maze, int treasureRow, int treasureColumn	Public	Instantiate the class fields to their associated parameters. **Initialize playerRow to 0.** **Initialize playerColumn to 0.**

Note: Be sure that you accept the parameters in the correct order!

Methods

Name	**Return Type**	**Parameters**	**Modifier**	**Description**
getMaze	boolean[][]	None	Public	Returns the maze.
getPlayerRow	int	None	Public	Returns the player's row.

getPlayerColumn	int	None	Public	Returns the player's column.
getTreasureRow	int	None	Public	Returns the treasure's row.
getTreasureColumn	int	None	Public	Returns the treasure's column.
setMaze	void	boolean[][] maze	Public	Sets the maze.
setPlayerRow	void	int playerRow	Public	Sets the player's row.
setPlayerColumn	void	int playerColumn	Public	Sets the player's column.
setTreasureRow	void	int treasureRow	Public	Sets the treasure's row.
setTreasureColumn	void	int treasureColumn	Public	Sets the treasure's column.

checkWin	boolean	None	Public	Return true if both the player row and player column are the same as the treasure row and treasure column, respectively, false otherwise.

MazeNavigator

This class facilitates the gameplay for the player. The process will be divided into two stages, initializing the game data and playing the game. During the first stage, the maze dimensions, position values, and treasure location are entered. Once the maze is configured, the player can begin the second stage and play the game. The game continues until the player finds the treasure (just as with a real maze, you cannot leave mid-game).

Notes:

- Maze configuration input values will be entered as comma-separated lists. Dimensions and positions within the maze will be entered as [row,column] and the values for each position will be a list the same length as the row.
- When ready to start, the player will enter "Yes" or "yes". All other input values should be interpretted as a negative response. Review the sample outputs for an example.
- When choosing a direction, the player will type the selected option out as an input.
- **Moves that step outside the bounds of the array will wrap around to the other side.**

Playing the Game

The player always starts in row 0 and column 0. Each turn, they will select a direction to move. If that move is valid, they will move one space in the selected direction. **If it is not valid, an error message is printed and the player does not move.** The player takes turns until they find the treasure.

Output

The starter code includes all of the output prompts. Several sample games are included below.

Additional Note

In Sample 1 below the first input is represented by [2,2].

The '[' and ']' simply indicate inputs. You would type 2,2.

==========

Sample 1

Welcome to the Maze Navigator!

Initializing maze...

Please enter the maze dimensions:

[2,2]

Please enter the values for the maze's row 0:

[true,true]

Please enter the values for the maze's row 1:

[true,true]

Please enter the expected treasure location:

[1,1]

Ready to start?

[Yes]

Player's Position: 0,0

Please enter a move:

1. Up

2. Down

3. Left

4. Right

[Up]

Player's Position: 1,0

Please enter a move:

1. Up

2. Down

3. Left

4. Right

[Right]

Treasure found!

Thank you for playing!

Sample 2

Welcome to the Maze Navigator!

Initializing maze...

Please enter the maze dimensions:

[1,1]

Please enter the values for the maze's row 0:

[true]

Please enter the expected treasure location:

[0,0]

Ready to start?

[No]

Thank you for playing!

Sample 3

Welcome to the Maze Navigator!

Initializing maze...

Please enter the maze dimensions:

[2,2]

Please enter the values for the maze's row 0:

[true,**false**]

Please enter the values for the maze's row 1:

[true,false]

Please enter the expected treasure location:

[1,0]

Ready to start?

[Yes]

Player's Position: 0,0

Please enter a move:

1. Up

2. Down

3. Left

4. Right

[Right]

Invalid move! Select another direction.

Player's Position: 0,0

Please enter a move:

1. Up

2. Down

3. Left

4. Right

[Left]

Invalid move! Select another direction.

Player's Position: 0,0

Please enter a move:

1. Up

2. Down

3. Left

4. Right

[Down]

Treasure found!

Thank you for playing!

Note: Brackets [] indicate input.

Testing

We have included a program that will allow you to create and run output tests automatically in the Starter Code. This will make it easier for you to verify that each possible progression through your solution is correct. Take a look at RunLocalTest.java. There are many utility

features and tools that you do not need to worry about at the moment, instead, focus on the test cases. They are included in the Starter Code and correspond to the output examples included above. Read through them.

You can modify the existing tests or create new ones to evaluate your implementation with other games. You will need to do so to ensure your solution meets expectations. Use larger and more complex boards to help identify bugs. Do not submit on Vocareum until after you have completed extensive testing.

Public Test Cases Note

For many homeworks and projects, we will give you test cases that correspond to several of the ways we will be testing your program. But, we will not give you test cases for ALL of the ways we will be testing your program. You should think of other test cases to use that will fully test every aspect of every feature of your program. Just because your program passes all the test cases we give you does not mean that it is fully correct and will receive a score of 100.

Submit

After testing your solution and verifying that it meets the requirements described in this document, you can submit on Vocareum. You have 4 submission attempts to achieve full points.

// Code

```
import java.util.Scanner;

public class MazeNavigator {

    public static final String WELCOME = "Welcome to the Maze Navigator!";
    public static final String INITIALIZE_MAZE = "Initializing maze...";
    public static final String MAZE_DIMENSIONS = "Please enter the maze dimensions:";
    public static final String MAZE_VALUES = "Please enter the values for the maze's row %d:";
    public static final String TREASURE_LOCATION = "Please enter the expected treasure location:";
    public static final String READY = "Ready to start?";
    public static final String CURRENT_POSITION = "Player's Position: %d,%d";
    public static final String MOVE_SELECT = "Please enter a move:";
    public static final String[] MOVES = {"Up", "Down", "Left", "Right"};
    public static final String INVALID_MOVE = "Invalid move! Select another direction.";
    public static final String TREASURE_FOUND = "Treasure found!";
    public static final String FAREWELL = "Thank you for playing!";

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
    }
}
```

Concepts

Array (1-D, 2-D , n-D array)
Class
Methods (getter/setter)
Constructor

Exception Handling