Final Project - Maze Runner

For your final project you will be writing the code for "**MazeRunner**", a program that navigates through a given maze. The code for the Maze is already written, and provided in "Maze.java". You just need to write the code that uses Maze and decides how to move through it.

Part 1 - Let the user solve the maze

For this stage, you will need to do the following steps:

1. Download "Maze.java" and make sure IT IS IN THE SAME FOLDER AS YOUR CODE
2. Create a new MazeRunner class.
3. Create a new "Maze" in your MazeRunner class like so:

```
Maze myMap = new Maze();
```

4. Welcome the user to Maze Runner and show them the current state of the Maze. Separate this intro out into a separate method called "intro()".

```
Welcome to Maze Runner!
Here is your current position:
. . . . . .
x . . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

__________________________________
```

For this section, you'll need to print out the current state of the map, which you can do so like this:

```
myMap.printMap();
```

Your map will look like all "." Characters and one "x" character. The "x" represents your current position and the "." Represents an unknown space. As you move through the map the "." Will turn into either walls ("-") or free spaces ("*"). There might even be more surprises in store for you as well.

5. Once you've greeted your user, then offer your user a way to enter in which direction they would like to move, Right (R), Left (L), Up (U) or Down (D). Create a method called userMove that lets the user enter their response to the following question:

```
Where would you like to move? (R, L, U, D)
```

This method should return a string indicating which direction the user chooses to move in. This method should guarantee that the user selection is only one of the given options, and continue to reprompt the user until they enter a desired direction.

6. Before you execute a move in any direction you need to make sure you can move that way according to the map. To do that you'll need to use the following methods:

```
myMap.canIMoveRight() // Returns true if the space to the right is free,
false if there is a wall
myMap.canIMoveLeft() // Returns true if the space to the left is free, false
if there is a wall
myMap.canIMoveUp() // Returns true if the space above is free, false if there
is a wall
myMap.canIMoveDown() // Returns true if the space below is free, false if
there is a wall
```

If you cannot move in that direction, notify the user there is wall in that direction and ask them to pick a new direction to move.

```
Sorry, you've hit a wall.
```

Here is an example of what happens when the user tries to move onto a space, but hits a wall:

```
Where would you like to move? (R, L, U, D) R
. . . . . . . . . . . . . . . . . . . .
* * x . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .

Where would you like to move? (R, L, U, D) U
Sorry you've hit a wall
```

7. If you find a direction you can move in- move there! To do that you'll need the following methods:

```
myMap.moveRight() // Moves your position one to the right and prints out the
new board
myMap.moveLeft() // Moves your position one to the left and prints out the
new board
myMap.moveUp() // Moves your position one above and prints out the new board
myMap.moveDown() // Moves your position one below and prints out the new
board
```

After your user makes a move they will want an update on their location, you can use myMap.printMap() to print out the current state of the map. Here is an example of a successful move:

```
Where would you like to move? (R, L, U, D) R
. . . . . .
* x . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
```

8. Repeat the process of asking a user where they would like to move until you reach the end of the game. You will know you've won when the method myMap.didIWin() returns true. Until you win that method will return false. When the user finally escapes the maze print the following message:

```
Congratulations, you made it out alive!
```

Part 2 – Move Limit

Now that you have a basic Maze Runner, we're going to make it a bit more difficult for your user. For Part 2 add the limitation that users will only have 100 steps to move before the exit to the maze closes! This means you need to keep track of how many moves a user makes. Create a new method called "**movesMessage(moves)**" that takes in an int representing the user's current number of moves and produces these messages when the user hits the following milestones:

Number of Moves	Message
50	Warning: You have made 50 moves, you have 50 remaining before the maze exit closes
75	Alert! You have made 75 moves, you only have 25 moves left to escape.
90	DANGER! You have made 90 moves, you only have 10 moves left to escape!!
100	Oh no! You took too long to escape, and now the maze exit is closed FOREVER >:[

Note that after 100 moves, if you still haven't escaped you need to end the game and tell the user they lost, and then allow the program to exit. When this happens print the following message:

```
Sorry, but you didn't escape in time- you lose!
```

If the user manages to make it out before they run out of moves add a line to the congratulations statement that tells them how many moves it took them:

```
and you did it in XX moves
```

where XX should be replaced with the user's move count

Part 3 – Watch out for pits

As you move through the maze you might come across pits. If your user moves onto a square with a pit they will fall and lose the game. Instead you want to make sure that you are looking ahead and letting them know if there is a pit in front of them. If there is, you should give them the option to jump over it in the direction they were heading. Thankfully the Maze class already has the methods you need to do this.

```
myMap.isThereAPit("R") // Takes in the direction String the user entered in
and returns if there is a pit ahead
myMap.jumpOverPit("L") // Will jump over a pit in the direction given,
skipping that space and landing 2 spaces over in the direction specified.
```

All your code to handle a pit once it's been detected should be put into a separate method called **navigatePit()**. When you find a pit, use the following test to ask the user what to do next:

```
Watch out! There's a pit ahead, jump it?
```

The user should be able to enter in any string that starts with a "y". If they enter in anything else it will be assumed they do not want to jump the pit so they should be asked again Where they would like to move. Here is an example of what your code should do when it encounters a pit:

```
Where would you like to move? (R, L, U, D) R
. . . . . .
* x . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Where would you like to move? (R, L, U, D) R
Watch out! There's a pit ahead, jump it? yes
. . . . . .
* * 0 x . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
```