Class Maze

java.lang.Object
  Maze

```
public class Maze
extends java.lang.Object
```

Implement a basic maze. EN.500.132 Bootcamp Java

Constructor Summary

Constructors

Constructor	Description
`Maze()`	Create the internal structure of a maze of a default size.
`Maze(int r, int c)`	Create the internal structure a maze of a specified size.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
`Cell`	`getCellAt(int r, int c)`	Return the Cell object stored at the given (row, column) position.
`int`	`getNumCols()`	Get the number of columns in the maze.
`int`	`getNumRows()`	Get the number of rows in the maze.
`boolean`	`isValid()`	Validate the cells of a maze as being consistent with respect to neighboring internal walls.
`boolean`	`readMaze (java.lang.String s)`	Read a maze from a plain text file whose name is supplied as a parameter to this method, and validate the mazes's wall structure.
`java.lang.String`	`setCellAt(int r, int c, java.lang.String d)`	Set the contents of a Cell in a given (row, column) position.
`boolean`	`solve()`	Solve the maze, assuming start in top left cell and finish in bottom right cell.
`boolean`	`solve(int srow, int scol, int erow, int ecol)`	Solve the maze from a given starting point to ending cell.
`java.lang.String`	`toString()`	Create and return one (long) string that contains the row and column dimensions of the maze, then a newline, followed by the string representation of each cell, one row at a time, with each cell separated from the next with one space and each row separated from the next by a newline ('\n').

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Details

Maze

```
public Maze()
```

Create the internal structure of a maze of a default size.

Maze

```
public Maze(int r,
            int c)
```

Create the internal structure a maze of a specified size.

Parameters:

`r` - the desired number of rows in the maze

`c` - the desired number of columns in the maze

Method Details

toString

```
public java.lang.String toString()
```

Create and return one (long) string that contains the row and column dimensions of the maze, then a newline, followed by the string representation of each cell, one row at a time, with each cell separated from the next with one space and each row separated from the next by a newline ('\n').

Overrides:

`toString` in class `java.lang.Object`

Returns:

the string representation

readMaze

```
public boolean readMaze(java.lang.String s)
                 throws java.io.IOException
```

Read a maze from a plain text file whose name is supplied as a parameter to this method, and validate the mazes's wall structure. This method assumes the specified file exists. The first line in the text file must contain the number of rows and columns, respectively. Each subsequent line provides the wall information for the cells in a single row, using a 4-character string ("bit string") in NESW (north-east-south-west) order for each cell. A 1 "bit" indicates the wall exists, a 0 "bit" (or any character other than 1) means no wall.

Parameters:

`s` - is the external name of the file to read

Returns:

true if a valid maze is created, false otherwise

Throws:

`java.io.IOException` - if file is not well-formatted

isValid

```
public boolean isValid()
```

Validate the cells of a maze as being consistent with respect to neighboring internal walls. For example, suppose some cell C has an east wall. Then for the maze to be valid, the cell to C's east must have a west wall. (This method does not consider external walls.) This method does not check for solvability of the maze.

Returns:

true if valid, false otherwise

getCellAt

```
public Cell getCellAt(int r,
                      int c)
```

Return the Cell object stored at the given (row, column) position. This method assumes its arguments describe a legal position.

Parameters:

`r` - the row position of the Cell in the Maze object

`c` - the column position of the Cell in the Maze object

Returns:

the Cell object that is at the specified position

setCellAt

```
public java.lang.String setCellAt(int r,
                                  int c,
                                  java.lang.String d)
```

Set the contents of a Cell in a given (row, column) position. This method assumes its arguments describe a legal position.

Parameters:

`r` - the row position of the Cell in the Maze object

`c` - the column position of the Cell in the Maze object

`d` - the data String to store at the specified position

Returns:

the former contents of the cell

getNumRows

```
public int getNumRows()
```

Get the number of rows in the maze.

Returns:

the number of rows in the maze

getNumCols

```
public int getNumCols()
```

Get the number of columns in the maze.

Returns:

the number of columns in the maze

solve

```
public boolean solve()
```

Solve the maze, assuming start in top left cell and finish in bottom right cell. This method changes data values inside explored cells, so that cells which are determined to be part of the final path ("the solution") through the maze will now contain the string "P" as their data, while cells which were explored but not selected as part of the solution path will now contain "x" as their data. If no complete solution path in the maze exists, no cells' data will be permanently changed to "P", but many may now contain "x".

Returns:

true if solved, false if fails

solve

```
public boolean solve(int srow,
                     int scol,
                     int erow,
                     int ecol)
```

Solve the maze from a given starting point to ending cell. This method changes data inside explored cells, so that cells which are part of the final path through the maze contain "P" as their data, while cells which were explored but not selected as part of the solution path contain "x" as their data. If no complete solution path in the maze exists, no cells' data will be permanently changed to "P", but many may now contain "x".

Parameters:

`srow` - the start row index

`scol` - the start col index

`erow` - the end row index

`ecol` - the end col index

Returns:

true if solved, false otherwise