CSE-212 Object-Oriented Programming and Design

Project (Individual, 20 points)

Deadline: December 9 (Thursday) 23:59

The Game

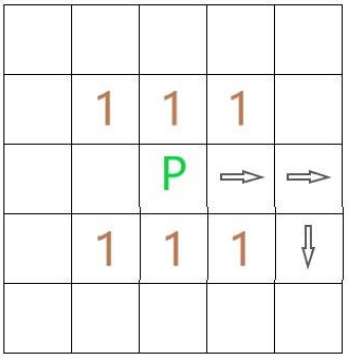
Introduction

The purpose of the whole project is to provide you with the opportunity to review some of the ideas and techniques you should be familiar with from this course, including file I/O, manipulation of arrays, and fundamental principles of object-oriented development.

You should be able to complete this assignment within a few days. Please, start early so that you have time to ask questions for clarification.

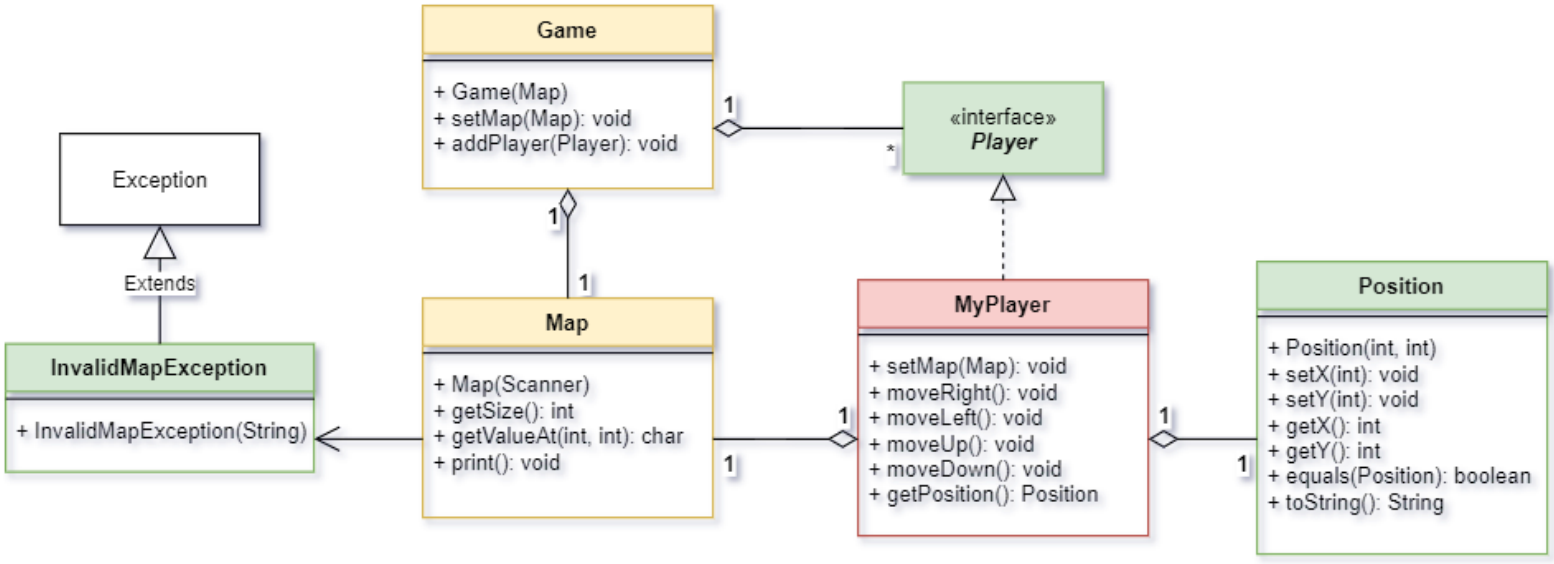
Project Requirements Overview

You will be implementing a simple game simulation based on a 2D map where the user (player) can move in four main directions: right, left, up, and down. There is no GUI, but the aim is to design all the necessary components and structure them in a way so that it can serve as a base for some more specific game types.



For testing purposes, you need to determine the final position of a player after completing a given sequence of moves. Detailed design requirements are given below, and sample input/output test cases attached at the end of this document.

UML Class Diagram (link to the file)



Design Requirements

Certain specific design requirements that you must follow are described below:

1. *Position class*
   - Stores a position (x and y coordinates).

2. *Player interface*
   - Interface for a player object, with some basic commands;
   - Moving commands: moveRight, moveLeft, moveUp, moveDown;
   - setMap: to add a map to a player;
   - getPosition: to get the position of a player.

3. *MyPlayer class*
   - Implements the Player interface;
   - Binds a player to a specific map;
   - Moves the player in four directions within the given map;
   - Makes sure a player does not go out of bounds or through the walls.

4. *Map class*
   - Constructs a map from a given input (command-line **or** file);
   - `size`: size of map (number of columns/rows);
   - Fills the map;
   - Provides the starting point for the player;
   - Throws ***InvalidMapException*** with the following messages:
     a) "*Not enough map elements*" - problems with input values
     b) "*Map size can not be zero*" - if map size is zero.

5. *Game class*
   - Creates a game based on some map;
   - Adds a player(s) to the game.

6. *InvalidMapException class*
   - Custom Exception class;
   - Used by the Map class for map input format verification;
   - Prints appropriate exception reason messages.

7. *Map text file*
   - Sample text file here;
   - '0' - for empty cell, '1' - for wall, and 'P' - for the starting point of a player;
   - You can try your own maps as well.

Program Design

Good object-oriented design is required for this assignment. For example, all mutable data should be encapsulated in private fields, and all implementations must satisfy the requirements described above. The UML diagram above shows only the required part that must be implemented (classes and their public methods), but you should consider adding appropriate private data fields and additional helper methods, if necessary. Practicing good design and applying an OOP approach correctly is the primary goal of this assignment.

If you need help, you should ask in MS Teams, by email, during office hours or classes.

Notes on Grading

There are 6 classes in total, and each one needs to be implemented completely and correctly. Points are based on the number of test cases passed. Make sure that your program works exactly as described in the project requirements. Do not attempt to submit someone else's work. It shall be considered as plagiarism, and get ZERO points. Partial completion of the tasks will be worth some points even if you do not finish the whole assignment.

Do NOT share your code with anyone. It would be considered academic dishonesty, and strictly penalized. All the works submitted shall be inspected by a special program and reviewed by the instructors. Any kind of similarity, or not being able to answer questions on the project gives the instructor full right to penalize the work, and even cancel the results that have already been graded because of cheating issues. In simple words, the fact that you did not cheat yourself does not matter. All the sides involved in cheating (which is a crime) will be penalized. Again, do not share your code under any conditions.

Although you are strongly encouraged to ask questions and have discussions in MS Teams, make sure you do not share your code there as well, please.

Turning In

1. Include all your work in a single **JAVA** file;
2. Turn in via **HackerRank** (invitations will be sent to your KBTU email addresses);
3. Your practice instructor may ask you to demonstrate your work and answer some questions.

Works that are not turned in correctly or are late will NOT be accepted.

NOTES on Submission

DO NOT CHANGE the code in the main class (*Solution* class) given in HackerRank because it runs and checks the individual test cases. You should add your classes and implement them so that they meet all the requirements and integrate well with this main class. Modified code may be penalized.

The following are just some sample test cases to start your work from. The actual project should be tested based on HackerRank test cases.

Input Format

N: map size
NxN symbols: N rows and N columns; same row values are separated by space (‘0’-Empty, ‘1’-Wall, ‘P’-Player start position)
Moves: A single string that stores sequential moves of a player (R-Right, L-Left, U-Up, D-Down)

Output

Shows the final position of a player after completing all the moves given in input.

Sample Run of the Program:

Input-1:

```
3
P 0 0
0 0 0
0 0 0
RDD
```

Output-1:

```
Player final position
Row: 2
Col: 1
```

Input-2:

```
6
0 0 0 0 0 0
```

0 1 1 1 1 0
0 1 P 0 1 0
0 0 0 0 0 0
0 1 1 1 1 0
0 0 0 0 0 0
LLDDRRUUU

Output-2:
Player final position
Row: 3
Col: 4

Input-3:
7
0 0 0 0 0 0 0
0 1 1 1 1 1 0
0 0 0 1 0 0 0
0 0 0 1 0 0 0
0 0 0 1 0 0 0
0 0 0 1 0 0 0
0 0 0 P 0 0 0
RRRRDDLLLLLLLURDLURDL

Output-3:
Player final position
Row: 6
Col: 0

Input-4:
5
P 0 0 0 0
0 0 0 0 0
1 1 1 0 0
RDLD

Output-4:
Not enough map elements