CSC 207 Software Design
Winter 2015 — Assignment 1

1 Logistics

- **Due date:** Thursday 12 February 9:00pm
- **Group size:** Individual

2 Overview

For Assignment 1, you will implement a set of Java classes according to the UML diagrams provided in this assignment description.

3 Learning Goals

By the end of this assignment, you should have:

1. gained better understanding of specifying object oriented program design with UML,
2. implemented a given OO design specified by a UML class diagram in Java,
3. become more familiar with Java generics, interfaces, abstract types, and inheritance,
4. practised writing high quality Java documentation.

4 What to do for this assignment

1. Update your subversion repository to get the starter files of the assignment. You should see a directory called `a1` which contains directories `A1soln`, `src`, and the starter packages.
2. Start Eclipse and select `a1` as a workspace to work in on this assignment.
3. Create a new Java Project called `A1soln`. You should now be able to view the starter packages.
4. Complete the Java program according to the provided UML diagrams, as specified in this handout.
5. To submit your work, add and commit all your changes under the existing directory `A1soln`.

   Do **not** commit the files and directories generated by Eclipse, such as `bin`, `doc`, `.project`, etc. Marks will be deducted for submitting these files.

5 The Maze Game

We are implementing a simple maze game. Our maze is a 2-dimensional grid. Each cell in this grid is either a hallway or a wall. We denote a wall with the symbol `X` and an empty hallway with a blank space. There are two players in this game — two monkeys. We denote them with symbols `1` and `2`. The monkeys' objective is to eat as many bananas as possible. We denote bananas with the symbol `B`. Some bananas are stationary, and some bananas move about the grid. We denote mobile bananas with the symbol `M`. Figure 1 (left) shows an example initial state of the maze game. Notice the two players (`1` and `2`), four bananas (`B`), and two mobile bananas (`M`).

```
XXXXXXXXXXXXXXXXXXXXXXXX
X        B     X  B  X X
X X   X      X MX  XXX X
XXX XXXXXXXX X  X      X
X   X    X   X  XXXXXX X
X1  X  XXX   X   2   X X
X   X      M   XXXX  X X
X XXX XXXXXX      X    X
X X   X    X  X   XXXXXX
X XXXXX  X XXXXX       X
X B      X   B   X   X X
XXXXXXXXXXXXXXXXXXXXXXXX
```

```
XXXXXXXXXXXXXXXXXXXXXXXX
X  .......1    X  B  X X
X X.  X      X  X  XXX X
XXX.XXXXXXXX X MX      X
X ..X    X   X  XXXXXX X
X.. X  XXX   X   ... X X
X   X          XXXX. X X
X XXX XXXXXX      X2   X
X X   X    X MX   XXXXXX
X XXXXX  X XXXXX       X
X B      X   B   X   X X
XXXXXXXXXXXXXXXXXXXXXXXX
```

Figure 1: Example Initial and Intermediate States of the Game

In our implementation, monkey player 1 moves left when the user presses the `'a'` key, moves right on the `'d'` key, moves up on the `'w'` key and moves down on the `'s'` key. Similarly, monkey player 2 moves left on the `'j'` key, moves right on the `'l'` key, moves up on the `'i'` key and moves down on the `'k'` key.

The two players are allowed to move in any order, they do not need to take turns. If there are mobile bananas, they move randomly with each player's move.

As a player moves, the hallway cell is marked as "visited", which means no player can enter it again and no mobile banana can move there. We denote a visited hallway with the symbol **.**. Figure 1 (right) shows an example state of a maze game after players 1 and 2 made some moves. Notice that the mobile bananas moved and that player 1 ate a banana!

Each time a player eats a banana, the player's score is incremented. In our implementation, mobile bananas are worth more.

The game ends when either all bananas are gone (the monkey with the higher score wins, or, if the two scores are equal, we declare a tie) or if the game cannot continue because both players are stuck (in which case we declare no winners).

6 The Implementation

We have largely designed the Maze Game for you. Your task is to study the provided UML diagrams and produce the corresponding Java implementation.

`Grid<T>` is a (generic!) interface that defines a two dimensional grid of objects of type `T`. The class `ArrayGrid<T>`, which implements this interface, will define, among others, a `toString()` method, which needs to produce a `String` representation of the `Grid` in exactly the same way as specified in Figure 1. See Figure 2 for the UML class diagram.

Our Maze Game will store and manipulate a `Grid` of `Sprite`s. Carefully study the UML diagram that defines the various `Sprite`s that will make up the grid, and the relationship between them. See Figure 3 for details.

The class `MazeGame` is where all the action will happen! It keeps track of the `Grid` of `Sprite`s, the two `Monkey` players, and all the `Banana`s. In addition to the getters, this class will implement the following methods:

- The constructor of `MazeGame` takes a name of a text file and reads and initializes the `Grid` from it. The starter code has examples of the input files.
- The method `move` takes a character and, if it is a valid move (see the above description and the starter code), changes the position of the corresponding player accordingly. Notice that the mobile bananas move randomly at this time. If the input does not specify a valid move, the method does nothing.
- The method `hasWon()` returns the ID of the player who has won this game (1 or 2), returns 3 if there is a tie, and returns 0, if the game is not over yet (even if both players are stuck).

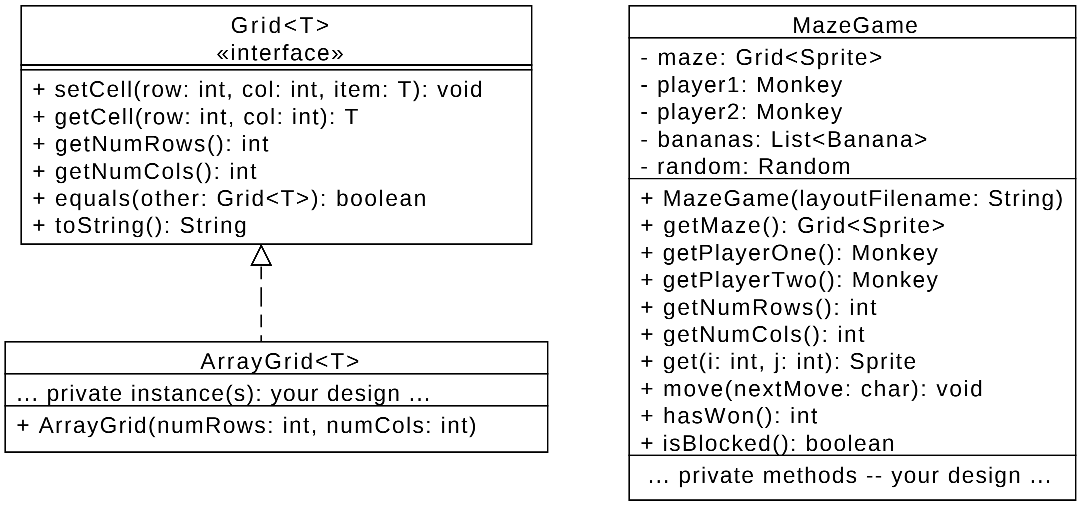
Grid<T>
«interface»
+ setCell(row: int, col: int, item: T): void
+ getCell(row: int, col: int): T
+ getNumRows(): int
+ getNumCols(): int
+ equals(other: Grid<T>): boolean
+ toString(): String
ArrayGrid<T>
... private instance(s): your design ...
+ ArrayGrid(numRows: int, numCols: int)
MazeGame
- maze: Grid<Sprite>
- player1: Monkey
- player2: Monkey
- bananas: List<Banana>
- random: Random
+ MazeGame(layoutFilename: String)
+ getMaze(): Grid<Sprite>
+ getPlayerOne(): Monkey
+ getPlayerTwo(): Monkey
+ getNumRows(): int
+ getNumCols(): int
+ get(i: int, j: int): Sprite
+ move(nextMove: char): void
+ hasWon(): int
+ isBlocked(): boolean
... private methods -- your design ...

Figure 2: Grid and MazeGame

- The method `isBlocked()` returns `true` if and only if no player can move at this point.

You will no doubt define several `private` methods (helper methods) in `MazeGame`, in addition to the required `public` methods specified in the UML diagram.

Finally, the interface `UI` defines a user interface for the Maze Game. We have provided a very simple GUI implementation. Your task is to provide a text UI that implements the given interface. The text UI should read from `System.in` and print to `System.out`. See Figure 4 for the UML class diagrams.

7 The Starter Code

Your repository contains the starter code for this assignment. Make sure to study it carefully before you begin coding! Pay careful attention to the class `MazeConstants`: you should be making extensive use of it in your implementation.

8 Evaluation

In addition to correctness, your submission will also be marked for style. Please follow the style guidelines outlined in `style.pdf`.

9 Checklist

Have you...

- followed the style guidelines?
- documented your code well? Generated the HTML documentation to see what the TAs will grade?
- tested your code on the lab computers?
- committed all of your files in the correct directory?
- verified that your changes were committed using `svn list` and `svn status`?

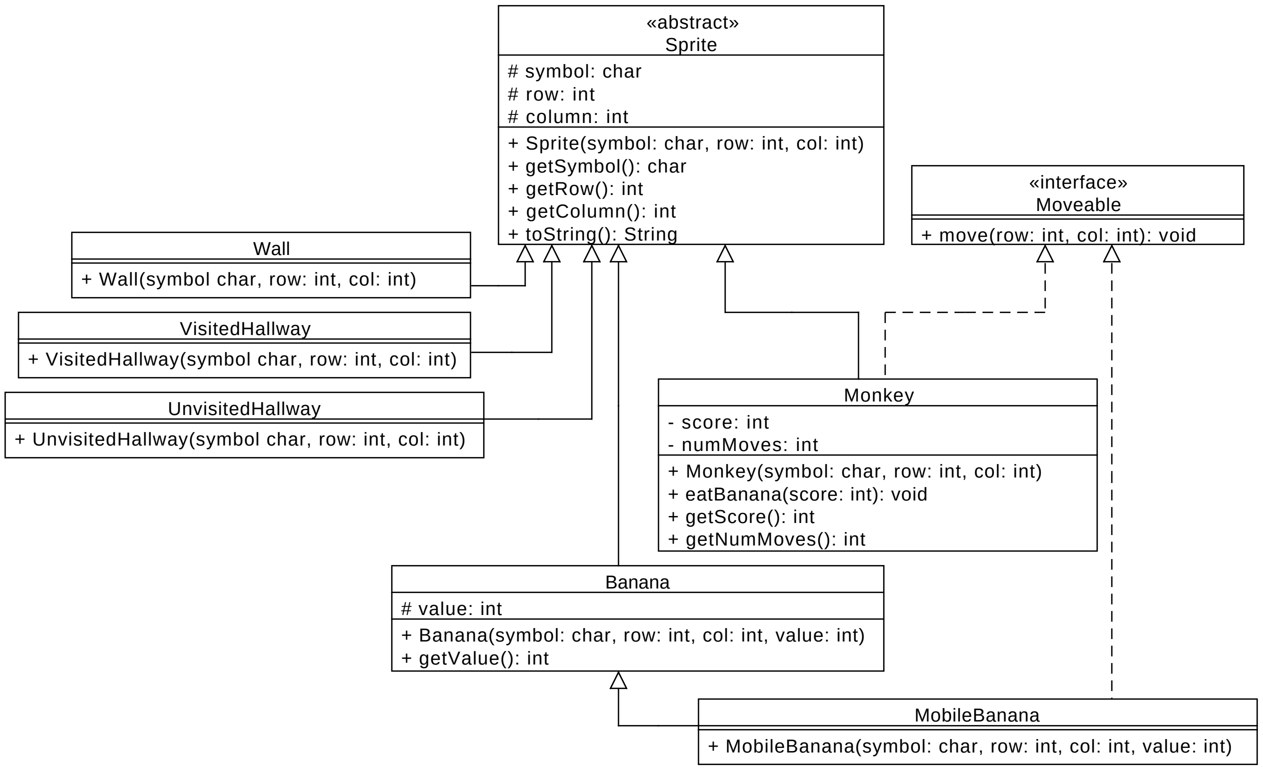
«abstract»
Sprite
# symbol: char
# row: int
# column: int
+ Sprite(symbol: char, row: int, col: int)
+ getSymbol(): char
+ getRow(): int
+ getColumn(): int
+ toString(): String
«interface»
Moveable
+ move(row: int, col: int): void
Wall
+ Wall(symbol char, row: int, col: int)
VisitedHallway
+ VisitedHallway(symbol char, row: int, col: int)
UnvisitedHallway
+ UnvisitedHallway(symbol char, row: int, col: int)
Monkey
- score: int
- numMoves: int
+ Monkey(symbol: char, row: int, col: int)
+ eatBanana(score: int): void
+ getScore(): int
+ getNumMoves(): int
Banana
# value: int
+ Banana(symbol: char, row: int, col: int, value: int)
+ getValue(): int
MobileBanana
+ MobileBanana(symbol: char, row: int, col: int, value: int)

Figure 3: The Various Sprites

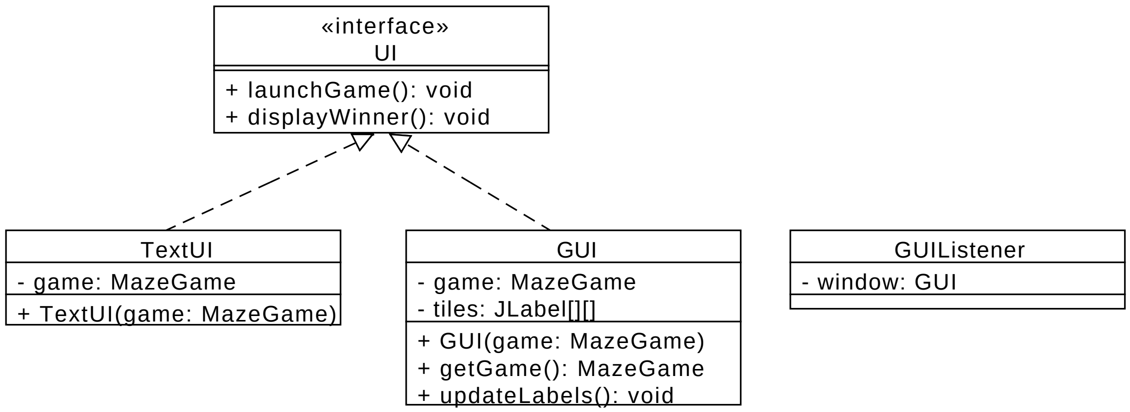

Figure 4: The User Interface