Project Refactoring TicTacToe (DUE DATE Dec 6th)

In a previous session, we have analyzed the TicTacToe program to identify bugs in the code. Now that we have a clean codebase (in the repository as SingleClassTTT.java), we will refactor the code to create a more Object-oriented version of the TicTacToe game.
Your able Professor has started to do this work, but got tired and now it is your job to finish what he has begun.
Goal: Use the old code (SingleClassTTT.java) to make the new code running again. Submit your results in the GitHub repository.

Expected Behavior:

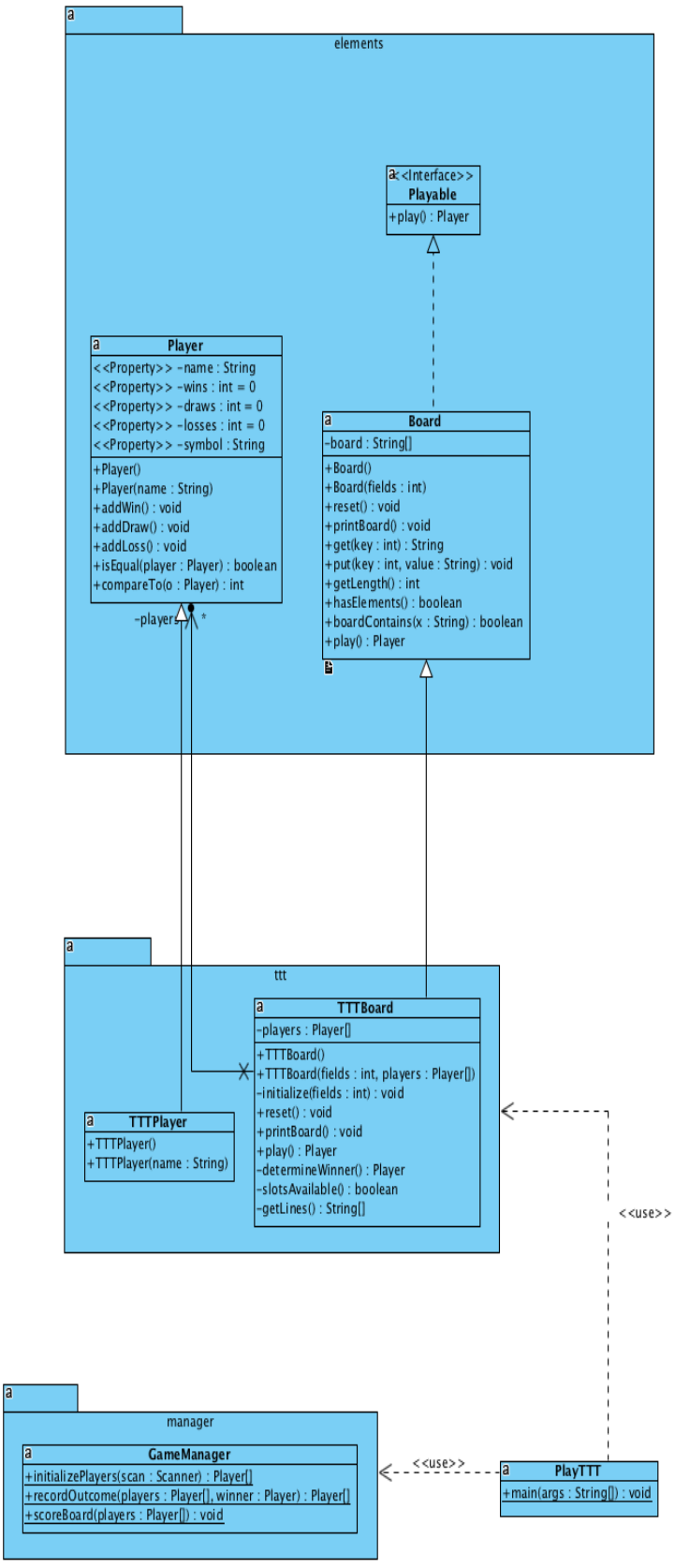
```
kb@Kamal-2020MBP Project_2911 % java PlayTTT
Please enter name of Player 1:
Kamal
Please enter name of Player 2:
Rohan
Welcome Kamal(X) and Rohan(O)
Have a great game!
/---|---|---\
| 1 | 2 | 3 |
|-----------|
| 4 | 5 | 6 |
|-----------|
| 7 | 8 | 9 |
/---|---|---\
Kamal: please enter the number of an available slot
1
/---|---|---\
| X | 2 | 3 |
|-----------|
| 4 | 5 | 6 |
|-----------|
| 7 | 8 | 9 |
/---|---|---\
Rohan: please enter the number of an available slot
2
/---|---|---\
| X | O | 3 |
|-----------|
| 4 | 5 | 6 |
|-----------|
| 7 | 8 | 9 |
/---|---|---\
Kamal: please enter the number of an available slot
4
/---|---|---\
| X | O | 3 |
|-----------|
| X | 5 | 6 |
|-----------|
| 7 | 8 | 9 |
/---|---|---\
Rohan: please enter the number of an available slot
5
/---|---|---\
| X | O | 3 |
|-----------|
| X | O | 6 |
|-----------|
| 7 | 8 | 9 |
/---|---|---\
Kamal: please enter the number of an available slot
7
/---|---|---\
| X | O | 3 |
|-----------|
| X | O | 6 |
|-----------|
| X | 8 | 9 |
/---|---|---\
The winner is Kamal
Would you like another match? (Y/N)
N

_____________________________________
Score Board
_____________________________________
Kamal: | 1 wins | 0 losses | 0 draws |
Rohan: | 0 wins | 1 losses | 0 draws |
_____________________________________
```

Enter the names of players
Send a welcome statement with names of players and their symbols (X) or (O)

Alert the player by name to make an entry

Ask for a rematch

Print out a well formatted Score Board
Horizontal and vertical boarders align



Structure of the code

The code is in three packages.

- **elements**: contains all super classes and Interfaces
- **ttt**: contains the realization of Boards and Players for TicTacToe
- **manager**: contains a helper class to printout scoreboards etc.

The main class is PlayTTT, which initializes the game.

Player is a class that implements a generic player. The state/data of the player is characterized by a name, a symbol and the number of losses, wins and draws the player has had.
The methods of the player manipulate these variables. The Player also contains a utility method *isEqual* that allows to compare two player objects

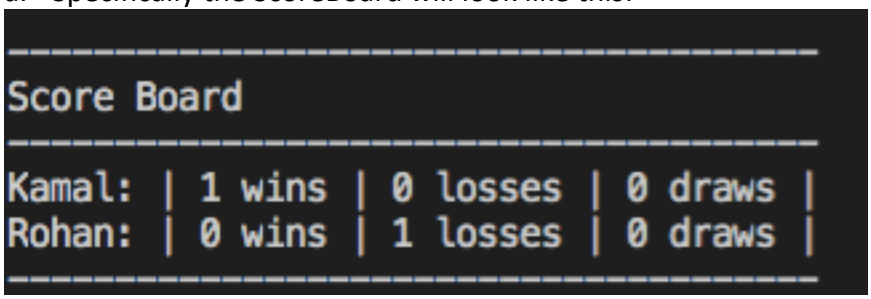
Board describes a generic game board. The state of the board is a String array of fixed size. The methods of the board allow to manipulate the board array by adding values, getting values, etc.
The Board class also implements the **Playable** Interface. This requires the board to implement a play method. The board can be played on. The play method returns a Player, e.g. a winner of the game.
None of these classes are specific to TicTacToe. You could envision other games that could use the same classes, abstractions.

The TicTacToe specific classes are **TTTPlayer** (which inherits from Player) and **TTTBoard** (which

inherits from Board). These classes override methods as required (e.g. the play method in TTTBoard is where the actual TicTacToe game is played).
The **GameManager** is a utility class that keeps track of menial tasks and helps to keep the code small in the main **PlayTTT** class. The GameManager uses static methods to offer its services.

Tasks and comments:

1. Your main task is to fix the code by addressing all the TO-Do's.
2. Compile the code with javac PlayTTT.java and run it with java PlayTTT
3. Follow the expected behavior above in detail, including formatting of the output.
    a. Specifically the ScoreBoard will look like this:



If the names are longer the borders on top and between wins, losses and draws must adjust automatically so that this format stays the same.

How to:

1. Go through the code and understand how it is structured. Start from the main class and follow the execution path.
2. The entire code is document and the individual tasks you need to complete are marked with comments //TO-DO. If you search in your editor for "TO-DO" you will find all tasks in each class.
3. Each //TO-DO is a small programming task that will require you to understand the basic principles of OOP, but sometimes TO-DO's are simple coding tasks.
4. You can use the SingleClassTTT.java to see how the code works with only one main class and copy and paste aspects of the code over.

What not to do:

1. You cannot change the inheritance structure or introduce new Interfaces or classes.
2. You cannot change the expected behavior. The output needs to be exactly the same.

Grading

- Total points: 30
    - 20 points for working code
    - 10 points for consistent code (i.e. you demonstrated that you understand OOP)