CSC207H: Assignment 1

Group size: Individual

Deadline: Wednesday 23 May at 10 PM.

Summary: Write `Player`, `Game`, and `Tournament` classes to pass a given set of unit tests.

Learning objectives

- Use IntelliJ to develop Java classes
- Understand the basics of Java's type system and how to handle input and ouput
- Do test-driven development using Java's unit test framework (JUnit)
- Work with Java syntax

Javadoc and other documentation is not required for this assignment.

Problem domain

Consider a tournament that is made up of any number of games. Each game can have any number of players. Each game has an identifier and a list of players. Each player can play in any number of games.

For example, game "A1" can contain "Alice Player1" and "Bo Player2". Likewise, game "A2" can contain "Alice Player1" and "Chen Player3". The list of games for "Alice Player1" should contain "A1" and "A2".

To organize a tournament, your program will be able to take an input file that lists players with their corresponding games. Your main method should allow the user to type the identifier for a game. Your program will print to the screen a list of all players (from the input file) in that game. For a given game, the same player should not appear twice when you print to the screen .

Starter code and setting up

We provide a set of unit tests in the `TournamentTest` class and nearly-empty classes for `Player`, `Game`, and `Tournament`. Each `Player` has a list of games in which they play, and each `Game` has a list of players who play in it.

As always, proper capitalization is vital.

You will write class `Tournament` which will contain your main method. First it will take a file called `PlayerList.txt` as input. `PlayerList` will contain at least one line for each player. Each line will be formatted like this:

```
PlayerName, rank | game1 | game2 | ... | gameN |
```

Where the number of games `N` can be a different natural number for each palyer, including zero. For example, one line in the file could look like:

```
Selena Williams, 1 | A1 | C3 | A2 |
```

Then, your main method will print to the screen: `Enter a game:`. If the user types in the name of a game that appeared in `PlayerList.txt`, for example: `A2`, and the only player in `PlayerList.txt` that will play in A2 are Serena Williams, and Roger Federer, then the output to the screen will look like this:

```
A2 (Serena Williams, Roger Federer)
```

Your code should look for this information in the file `PlayerList.txt`. The user should not have to specify the file name. Your filepath should be "PlayerList.txt" without any folders or slashes. It is up to you to create your own PlayerList.txt for testing purposes and experiment with your system to see which folder should contain `PlayerList.txt` in order for your code to run.

Your program should continue to tell the user to enter a new game until the user enters the word `exit`, at which point your program should terminate. If the user types in a string that is neither `exit` nor the name of an game in `PlayerList.txt`, the program will print to screen `This is not a valid game.`

It will be useful to know about: overriding the `equals` and `toString` methods, usage of `this`, syntax and methods for `ArrayList` of type `Game or Player, and casting.`

Here are the instructions for getting started:

1. Create a folder called "assignment1" and, inside that, create a folder called "src".
2. Create a project in IntelliJ called assignment1 by importing the "assignment1" folder. If you are asked if you want to open the project in a new window, say yes.
3. Download starter.zip. Unzip it and move each file into the src folder of your project.
4. Mark the "src" folder as "Test Sources Root". Each system is different, but you will likely be able to do this by right-clicking on "src" in the left pannel and go to "Mark Directory As...".
5. You should now see Player, Game, Tournament and the test file under src on the left panel of your IntelliJ window. You will write the code for the missing methods in Player and Game. They are:

   Class Game: getId, getPlayers, addPlayer, hasPlayer, equals, and hasSamePlayers, and toString.

   Class Player: getName, getRank, addGame, getGames, equals, and toString.

   You will also write the code for your main method in Tournament.

   You will also have to add JUnit to your project. This looks different on different operating systems. You can find and share advice for accomplishing this on the discussion board or drop by office hours. If IntelliJ displays a red lightbulb somewhere in your test file, you may be able to add JUnit by clicking on it.

   Your job for this assignment is to write those methods and make the test cases pass. Then write the main method. If you pass all of the test cases **on the teaching lab computers** and handle a test file of the specified format, in the specified manner, you will earn 100% on the assignment.

   To do this, make sure that IntelliJ did **not** automatically import any extra libraries beyond java.util or a JUnit package. Also, be sure that no package names appear at the top of any of your files. **This applies to Assignment 1 only.**

A suggestion for doing this assignment

First, create method *stubs* so that all the unit tests compile. A method stub is a method header and just enough of a body to get it to compile.

For example, you might write these stubs for getId and hasPlayer in class Game:

```
    public String getId() {
        return null;
    }

    public boolean hasPlayer(Player p) {
        return false;
    }
```

Once your code compiles, run the tests. Many will fail, but your setup phase is done!

Now pick a unit test and read it carefully. We recommend starting with testPlayerGetName because it's first, but you're welcome to start elsewhere. Write just enough code for the unit test to pass. Move on to the next unit test and do the same thing: read it carefully so that you understand what it is testing, and then write just enough code for it to pass.

You will sometimes need to iterate over part of the contents of an ArrayList. You can use a loop that looks like this one, which iterates over all the items in the list:

```
    int i = 0;
    while (i != this.games.size()) {
        // call this.games.get(i), cast it to type Game, and do something with it.
        i = i + 1;
    }
```

You can, of course, also use a for loop.

Line separators

Unix, Windows, and Mac all use different newline separators. Linux and MacOS use \n, but Windows uses \r\n. System.lineSeparator() returns the appropriate one, depending on the operating system on which the code is executed. You **must** use it to indicate newlines; this is relevant to method toString in class Player.

Handling Input

You will want to explore the split method in class String when writing your main method. This method takes a regex as an argument. Regex is short for Regular Expression. We will be learning about these later in the course. For now, you will need to know that "\\s" is the regex for a single blank space, "\\," is regex for a comma, and "\\|" is the regex for the symbol | in Java.

There is a file called ReadWrite.java in the Week 1 Readings on the course website. You can re-use any of the code in this file, if appropriate. It demonstrates various methods for inputting text into your program. You are responsible for looking up unknown methods or classes on PCRS, the Oracle website, or from reliable alternative sources. Specifically, you may want to look up the Scanner class and the methods it contains.

Submitting the assignment

You will submit the assigment on MarkUs. The link for this will be active on the course website by Monday 14 May. Once there, log in with your UTORid and click on Assignment 1. Try to do this before the actual deadline so that you can seek assistance if required.

Re-mark Requests

Your code for a1 must run on the teaching lab servers. Once the marks for a1 have been released on MarkUs, we will send out an e-mail regarding the re-marking request process. It is unlikely that you will need to do this, since your assignment will be graded on the teaching server. So we will see the same results that you see when you run your assignment on IntelliJ on a computer in a teaching lab.