CS-102 – ASSIGNMENT 1 – 2020 SPRING

GIUSEPPE TURINI – KETTERING UNIVERSITY

INTRODUCTION

To complete this assignment, you need knowledge of the material covered in CS-101, as well as knowledge of the ADT list covered at the beginning of CS-102. The goal is to design and implement a simple database system to store tennis players and tennis matches and some statistics.

INPUT FILE FORMAT

Your program will accept 1 argument in the command line: a file name. This file should be in the current directory, and it should contain the input data for the tennis database (see below).

Tennis Player Line.

Each tennis player is represented by this info separated by `/` and in this order:

1. The word `PLAYER`, a token indicating that this line represents a tennis player.
2. An alphanumeric sequence representing the unique id of the player (e.g. `FED81`, `DJO87`).
3. A word representing the first name of the player (e.g. `ROGER`, `NOVAK`, `RAFAEL`).
4. A word representing the last name of the player (e.g. `FEDERER`, `DJOKOVIC`, `NADAL`).
5. A number representing the year the player was born (e.g. `1981`, `1987`, `1986`).
6. A word representing the country of the player (e.g. `SWITZERLAND`, `SERBIA`, `SPAIN`).

This is an example of a player line: `PLAYER/FED81/ROGER/FEDERER/1981/SWITZERLAND`

Tennis Match Line.

Only single matches are allowed, each represented by this info separated by `/` and in this order: [1]

1 The word `MATCH`, a token indicating that this line represents a tennis single match.

2 An alphanumeric sequence representing the *id* of the *first player* of the match (e.g. `FED81`).

3 An alphanumeric sequence representing the *id* of the *second player* of the match (e.g. `NAD86`). [2]

4 An 8-digit numeric sequence representing the match date, in the format YYYYMMDD.

5 A word representing the location (i.e. the tournament) of the match (e.g. `WIMBLEDON`).

6 An alphanumeric sequence representing the match score (e.g. `6-4,2-6,7-6`).

This is an example of a match line: `MATCH/NAD86/DJO87/20150101/ROME/6-2,6-3,6-4`

Input File Example.

The following is an example of the content of the data file which could be read by this program:

```
PLAYER/FED81/ROGER/FEDERER/1981/SWITZERLAND
PLAYER/DJO87/NOVAK/DJOKOVIC/1987/SERBIA
PLAYER/NAD86/RAFAEL/NADAL/1986/SPAIN
MATCH/FED81/NAD86/20070308/WIMBLEDON/6-4,2-6,7-6
MATCH/DJO87/FED81/20121231/US OPEN/6-2,3-6,6-7,7-5,6-1
MATCH/NAD86/DJO87/20150101/ROME/6-2,6-3,6-4
```

Note: The correct file format lists all the players before any match. Any input file violating this requirement should be discarded, and an appropriate error message should be displayed.

Note: All the strings should be stored internally and processed in uppercase.

Note: Input from the console and from files should be handled using the Java `Scanner` class. [3]

Your program will begin by reading in this information from the input file, and then storing it in the tennis database using the appropriate format.

[1] See: Wikipedia - Types of Tennis Match.
[2] Note: the second id cannot be equal to the first id.
[3] See: Java Standard Edition (SE) 8 - Scanner.

FUNCTIONALITY

Once started, and the input file loaded successfully, your program will then interact with the user through the console, by offering the following commands:

1 *Print All Players.*

Print the list of players, sorted alphabetically by their ids. The program should print all players in the database in an appropriate format (see example), including the win/loss record for each player.

The same sorted order (sorted alphabetically by player id) should be used to store the data.

2 *Print Matches of a Player.*

Print all matches of the input player, sorted by date (most recent first). The program asks the user for a player id, and then prints all matches including that id as the first or second player.

If input id cannot be found in the database, an appropriate error message should be displayed.

The same sorted order (sorted by date, most recent first) should be used to store the data.

3 *Print All Matches.*

Print all matches, sorted by date (most recent first). The program should print the match details including the full name of each player (instead of their ids).

The same sorted order (sorted by date, most recent first) should be used to store the data.

4 *Insert New Player.*

Insert a new player. The program should prompt the user to input the appropriate information on the console, and add the specified data to the database.

If the input player has an id already stored in the database (duplicate id), the player should be rejected, and an appropriate error message should be displayed.

5 *Insert New Match.*

Insert a new match. The program should prompt the user to input the appropriate information on the console, and add the specified data to the tennis database.

Only matches with both players in the database should be stored internally, all the other matches should be considered invalid and discarded, and an appropriate error message should be displayed.

6 *Exit.*

Exit the program. The program should terminate.

CONSOLE SESSION EXAMPLE

The following is an example of a console interactive session:

```
CS-102 Tennis Manager - Available commands:
1 --> Print All Players
2 --> Print All Matches of a Player
...
6 --> Exit
Your Choice? 1

DJO87: NOVAK DJOKOVIC, 1987, SERBIA, 214/34
FED81: ROGER FEDERER, 1981, SWITZERLAND, 285/47
NAD86: RAFAEL NADAL, 1986, SPAIN, 195/27

CS-102 Tennis Manager - Available commands:
1 --> Print All Players
2 --> Print All Matches of a Player
...
6 --> Exit
Your Choice? 2
Enter Player ID: DJO87

2015/01/01, R.NADAL - N.DJOKOVIC, ROME, 6-2,6-3,6-4
2012/12/31, N.DJOKOVIC - R.FEDERER, US OPEN, 6-2,3-6,6-7,7-5,6-1
```

Note: Remember to include in your prints the win-loss record (print players) and the player full name (print matches), as well as the required sorting for each print.

Note: This is just an example; your console menu may operate/appear differently.

Note: You should create/test different input files to make sure your program is correct and robust. Do not assume that testing the program on one input file is sufficient.

INTERNAL REQUIREMENTS

Your program should be designed with modularity and modifiability in mind, also because you will be revising and extending this program in the next assignment.

The following are the requirements for the classes and ADTs that you will need to design and code.

Class TennisPlayer.

Define a class `TennisPlayer` to implement the `TennisPlayerInterface`.

The `TennisPlayer` class contains members corresponding to a given player, and methods to deal with the player statistics (e.g. win/loss record).

Class TennisMatch.

Define a class `TennisMatch` to implement the `TennisMatchInterface`.

The `TennisMatch` class contains members corresponding to a given match, and methods to deal with the match details (e.g. winner).

This class should include an internal recursive function to parse the score (recursively on the sequence of set scores) in order to identify the winner (stored internally).

Class TennisDatabase.

Define a class `TennisDatabase` to implement the `TennisDatabaseInterface`.

The `TennisDatabase` class contains methods which allow to manipulate a collection of tennis records and, for this assignment, this class should be designed to store 2 objects:

- An object of type `TennisPlayerContainer`, storing all players.
- An object of type `TennisMatchContainer`, storing all matches.

Class TennisPlayerContainer.

The `TennisPlayerContainer` class implements the `TennisPlayerContainerInterface`.

The `TennisPlayerContainer` class should be a two-level list to store tennis players:

- The upper-level list will be a circular doubly linked list (without dummy head nodes), called `TennisPlayerContainer` and sorted by player id (alphabetically). The nodes in the upper-level linked list will be instances of the `TennisPlayerContainerNode` class, and they will store tennis players, with one node per player.
- Each `TennisPlayerContainerNode` object will also contain a lower-level linked list: a sorted linked list implemented by the class `SortedLinkedList<TennisMatch>`, and storing the tennis matches of that player, sorted by date (most recent first). [4] [5]

[4] Note: remember that each node should also store the data to implement the circular doubly linked list.
[5] Note: all the operations on these lists should exploit their structure and the fact that they are sorted.

- You should define a generic class `SortedLinkedList<T>` implementing a sorted linked list: using the type parameter `T` to customize the type of the data stored in the list; and constraining `T` to represent a *"comparable"* type (see bounded type parameters and wildcards). [6] [7] [8]

  For its implementation, the class `SortedLinkedList` will require a supporting generic class `SortedLinkedListNode<T>` to store the data and the link required for its implementation.

Class TennisMatchContainer.

The `TennisMatchContainer` class implements the `TennisMatchContainerInterface`.

The `TennisMatchContainer` class should be a dynamically allocated array-based list to store tennis matches, sorted by date (most recent first). This list/container should be implemented using an array of `TennisMatch` objects, re-allocating the array (i.e. increasing its physical size) whenever you need to add a new tennis match and the array is full. [9]

Class Assignment1.

You should define a class `Assignment1` with a `main` method which uses a `TennisDatabase` object to store data and perform the operations requested by the user.

Additionally, this is the only class allowed to print-to and read-from the console.

Package TennisDatabase.

All the classes mentioned above, except `Assignment1`, should be part of the `TennisDatabase` package, and the only `public` classes/interfaces in this package should be: `TennisDatabase`, `TennisPlayer`, `TennisMatch`, their associated interfaces, and all the exception classes.

Note: All the interfaces and exception classes and generic classes will be discussed in-class and provided by the instructor right after the beginning of this assignment.

Coding Style.

As always, your program should use good coding style, that will be evaluated accordingly to standard guidelines. [10]

[6] See: The Java Tutorials - Generics.
[7] See: The Java Tutorials - Bounded Type Parameters.
[8] See: The Java Tutorials - Wildcards.
[9] Note: you are free to decide the strategy for re-sizing the array (i.e. the increase in the array physical size).
[10] Note: if you need a reference you can use the Google Java Style Guide.

Exception Handling.

Your program should integrate standard exception handling: catch and properly handle all exceptions generated by any system routines you use, as well as any exceptions your own code generates (that is, your `main` method should not throw any exception).

SUBMISSION PROCESS

Email.

On the 5th Friday, before the end of the day, using your Kettering email account, send an email to the instructor (`gturini@kettering.edu`) with the subject: `CS-102: Assignment 1`. This email should include in attachment a compressed archive (see next section).

ZIP File.

The attachment to your email should be a zip archive, named like `cs102__a1__turini_giuseppe.zip` (please substitute your first and last name), and containing:

- All source code files of your assignment, already arranged in the package-folder hierarchy.
- A text file (`readme.txt`) explaining the reasons of any change/violation of the requirements.

EVALUATION FORM

STUDENT NAME:

FUNCTIONALITY AND INTERNAL REQUIREMENTS ___ / 70

Input file loaded (5)
Print tennis players (5)
Prints tennis matches (5)
Prints matches of player (5)
Insert new player (5)
Insert new match (5)

TennisPlayer class (5)
TennisMatch class (5)
TennisDatabase class (5)
TennisPlayerContainer class (10)
TennisMatchContainer class (10)
Assignment1 class (5)

DESIGN AND STYLE AND SUBMISSION ___ / 30

OOP principles and design (10)

Compilation and exception handling (10)

Coding style and commenting (10)

Submission and delay (…)

TOTAL ___ / 100