CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

ALGORITHMS AND DATA STRUCTURES

ASSIGNMENT 1: Data Structures for Database Management.
PART I: Java ADT, Interfaces and Contiguous-based Data Structures.

1. Background:

The UEFA Champions League is the most prestigious competition for football clubs. After its winter break, last week it was restarted for its last 16 knockout round. In this assignment you are going to complete a Java application for managing a database on the competition top scorers. In this first part of the assignment the application consists on:

- A Java class **myPlayer**, which is used to represent a football player.
- A Java interface **myList**, which is used to represent a list of myPlayer as an abstract data type.
- A Java class **myListArrayList**, which is used as a contiguous data structure-based implementation of the abstract data type myList.
- A Java class **myMain_ADS_A11.java**, which is used for an interactive session with the user, who can perform a set of operations over the database. In particular, the supported operations include file input / output operations for database persistence.
- The text file **database.txt**, with an example of a possible database.

2. Football Player Specification (myPlayer.java):

The Java class **myPlayer.java** is used to represent a football player.

- Each player is specified by its two attributes: name and goals. Whereas the former is a String representing the player's name, the latter is an int representing the amount of goals scored in the competition.
- The class constructor explicitly requires a value for the two attributes to create a new player instance (i.e., a new myPlayer object).
- Public observer / mutable operations (i.e., get / set methods) are provided for access and update an object attributes.
- A method print_info is provided to display by console the player attributes.
- A total order can be established on the players. Basically, given two players p1 and p2, $p1 \leq p2$ if p1 has scored more goals than p2, or if both players have scored the same goals, but the name of p1 is lexicographically smaller than the name of p2. A method smaller is provided, so as to allow each player p1 to compare itself with another player p2, returning true if p1 is smaller than p2, and false otherwise.

3. List of Players Specification (myList.java):

The Java interface **myList.java** is used to represent (as an abstract data type) a list of myPlayer. The interface provides two sets of operations:

- BASIC OPERATIONS:

    0. **myList create_empty();**
       This operation is classified as creator and total. However, Java does not support a constructor in an interface, so it is commented and implemented later on via the constructor of the class implementing the interface.
    1. **int my_get_length();**
       This operation is classified as observer and total. It computes the number of elements in myList.
    2. **myPlayer my_get_element(int index);**
       This operation is classified as observer and partial. It returns the player placed at the position 'index' of myList. If the index is out of bounds, the operation returns an error message.
    3. **void my_add_element(int index, myPlayer player);**
       This operation is classified as mutator and partial. It inserts the new myPlayer player at the position 'index' of myList, shifting one position all myPlayers placed after it. If a player in the list has the same name of the new player being entered, the operation returns an error message. If the index is out of bounds, the operation returns an error message.
    4. **void remove_element(int index);**
       This operation is classified as mutator and partial. It removes the myPlayer player from the position 'index' of myList, shifting one position all myPlayers placed after it. If the index is out of bounds, the operation returns an error message.

- PROBLEM-SPECIFIC OPERATIONS:

    1. **void load_file(String s);**
       This operation is classified as mutator and total. First, it empties myList. Then, it reads the players info stored in the database file 's'. For each player being read, an associated myPlayer object is created and stored at the end of myList. If the path specified in 's' does not contain a database file, then the operation returns an error.
    2. **void show_elements();**
       This operation is classified as observer and total. First, it displays the amount of players stored in myList. Then, it traverses myList, using the method myPlayer::print_info to display by console the info of each player.
    3. **int find_element(String s);**
       This operation is classified as observer and total. It traverses myList to see if a player with name 's' is stored on it. If the player is stored then it returns its index. If the player is not stored, it returns -1.

4. **void show_info(String s);**
   This operation is classified as observer and partial. It traverses myList to see if a player with name ‘s’ is stored on it. If the player is stored, it uses the method myPlayer::print_info to display its info by console. If the player is not stored, the operation returns an error message.
5. **void add(String s, int i);**
   This operation is classified as mutator and total. Given a name and a number of goals, it creates a new myPlayer object and stores it at the end of the list.
6. **void update(String s, int i);**
   This operation is classified as mutator and partial. Given a name and a number of goals, it traverses myList to see if a player with name ‘s’ is stored on it. If the player is stored, it uses the method myPlayer::set_goals to update its number of goals. If the player is not stored, the operation returns an error message.
7. **void remove(String s);**
   This operation is classified as mutator and partial. It traverses myList to see if a player with name ‘s’ is stored on it. If the player is stored, it deletes it, shifting one position all other myPlayers placed after it. If the player is not stored, the operation returns an error message.
8. **void bubble_sort(String s);**
   This operation is classified as mutator and total. It traverses myList to sort its players following the aforementioned total order. A bubble_sort-based algorithm as the one seen in the Lecture 02 / Lab02 can be applied. To compare two players, the aforementioned method myPlayer::smaller(player p) can be used.
9. **void write_file(String s);**
   This operation is classified as observer and total. It traverses myList to dump the players’ info into a database text file. For each myPlayer of myList, a line of text is written to the file, following the format:
   “myPlayer::get_name() myPlayer::get_goals() \n”.

4. List of Players Implementation (myListArrayList.java):

The Java class **myListArrayList.java** follows a contiguous data structure-based implementation of the abstract data type myList.

- In concrete, it uses an ArrayList of myPlayer elements as its unique attribute: ArrayList<myPlayer> items.
- The class constructor is used to simulate the implementation of the basic operation 0: myList::create_empty()
  This constructor creates an empty ArrayList.
- The rest of the class methods are devoted to implement the interface myList (i.e., its basic and problem-specific operations).

5. User Interactive Session with the Database (myMain_ADS_A11.java):

The Java class **myMain_ADS_A11.java** is used for an interactive session with the user, who can perform a set of operations over the database. The class provides the following methods:

- A static class Scanner variable, to interact with the user asking him to enter some inputs by keyboard.
- A static method select_option, asking the user to select among the following options:
    1. Load database from file.
    2. Display database players.
    3. Check if player is in database.
    4. Show player info.
    5. Add player to database.
    6. Update player of database.
    7. Remove player from database.
    8. Sort the players of the database.
    9. Save database to file.
    0. Exit

  The method asks the user until a valid option is entered.
- A static method interactive_session(), to trigger the interactive session with the user. This method contains a main loop performing:
    - Ask user for a valid option.
    - Process the action associated to that option.
- The loop (and the interactive_session) is exited when a 0 option is selected.
- The application MAIN method (or entry point), which triggers the execution of the interactive_session().

6. A Database Text File Example (database.txt):

The file **database.txt** represents a possible database to be used. It contains 7 players:

Aguero 27

Messi 93

Cavani 24

Ronaldo 95

Muller 36

Lewandowski 31

7. Exercise:

The files **myMain_ADS_A11.java**, **myPlayer.java** and **myList.java** are provided complete, so you do not need to edit them. The only file you have to edit is **myListArrayList.java**! Regarding **myListArrayList.java**, the basic operations are provided complete, so you do not need to edit them. Moreover, the problem-specific operations 1 and 9 are implemented as well.

GOAL: Implement the problem-specific operations 2…9 of myListArrayList.java!

Important: When implementing the problem-specific operations 2..9 do not use the class attribute items at all. The only operations allowed to use items directly are the basic operations! Thus, whenever you need to access to items for implementing the problem-specific operations, do not access to it explicitly! Instead, use the basic operations my_get_length, my_get_element, my_add_element and my_remove_element to implicitly access to items. This is very important for the second part of the assignment.

8. Submission Date:

Sunday 12th of March.

To submit the assignment, please enclose the entire Eclipse Java Project folder into a single zip file, and upload it to blackboard. Thanks!