CSC8001
Assessed Coursework 2 (week 9, 10 and 11): Interactive system

Aims:

- To gain experience at designing an interactive system of practical importance.
- To reinforce your knowledge about the standard `java.util.ArrayList<E>` class.
- To gain experience at using `java.lang.Comparable<E>` interface, inheritance and generic classes in Java.

Background:

Sometimes we want to keep the items in a list in sorted order, which we can achieve with a sorted list. The fundamental difference between a sorted list and an unsorted one is the "adding an item"/insertion method. Having a definition of a class for lists, we can obtain a sorted list class by altering the existing or adding new insertion method.

Specification:

Task 1:

Derive a `SortedArrayList<E>` class from the `java.util.ArrayList<E>` class in such a way that the items of a sorted array list are sorted in ascending order. This subclass of `ArrayList<E>` class will be needed to complete Task 2 (see Additional assumptions (2) below). For simplicity, you can provide only your new insertion method in the `SortedArrayList<E>` class. It is not a part of this project to consider all the mutator methods from `ArrayList<E>` class and see whether/how they should be overridden to make sure that a sorted list preserves its order when they are invoked on the list.

Task 2:

You are asked to write a program to help a World Sports Championships ticket office clerk in their work. Your program should read a list of available events and a list of registered clients from a file. The content of the input file should have the following form: The first line contains an integer representing the number of available events, and is followed by the information about the events (two lines for every event: one line containing the name of the event and the second one containing the number of tickets available for this event). The next line contains an integer representing the number of registered clients, followed by the information about the clients (one line for every client with their first name and surname). Example file content is given below:

```
6
Tennis
8
Athletics
4
Handball
66
Equestrian Jumping
7
Football
2
Volleyball
41
```

```
3
Emma Williams
Anna Smith
John Williams
```

The program should be able to store the information about events and clients:

1. For each event, the information required is: the name of the event and the number of tickets left for this event.
2. For each client, the office should know the first name, surname and the chosen events together with the number of tickets bought for each of the events. We assume that a client can buy tickets for at most 3 events. We also assume that no two clients share both the first name and the surname.

After the initial information has been read from the file, the clerk will be working with the program interactively. The program should display a menu on the screen offering a choice of possible operations, each represented by a lower case letter:

- f - to finish running the program.
- e - to display on the screen the information about all the events.
- c - to display on the screen the information about all the clients.
- b - to update the stored data when tickets are bought by one of the registered clients.
- r - to update the stored data when a registered client cancels/returns tickets.

You should implement all the above operations.

Additional assumptions:

1. When f is selected, the program stops running and all the data are lost. The program could be extended to save all the data to a file, but this is not a part of the project!
2. To store events and clients you should use `SortedArrayList<E>` class. Events should be sorted in the ascending order of names of events. Clients should be sorted in the ascending order of their surnames. If two clients have the same surname then the first names should decide their order. You can assume that each client has exactly one first name and exactly one surname.
3. When tickets for an event are ordered by a client, it must be checked whether the client is a valid client, and that the event is on the list of the available events. If not, an appropriate message should be displayed on the screen. If the request is a valid one, the program should check whether there are enough tickets left for the ordered event. You can assume that in one transaction a client is ordering tickets for one event. If the tickets are not available (or there are not enough tickets), a note (in a form of a letter) should be printed to a file informing the client that the tickets are not available. You can assume that the order is either fully satisfied or not at all. If the ordered number of tickets is available, the transaction can be processed and the stored information should be updated accordingly.
4. When tickets (a ticket) are (is) cancelled by a client, it must be checked whether the client is a valid client, whether the event specified by the tickets is on the list of events and whether the tickets have been purchased by this client. If not, an appropriate message should be displayed on the screen. If the request is a valid one, the stored

information should be updated accordingly. You can assume that a client can cancel tickets for one event only per one transaction.

5. We assume that the ticket office sells only the tickets initially allocated to it, serves only its registered clients and processes transactions sequentially, one after the other.

Submission Notes:

You should submit your Java files (files with .java extension) through NESS as well as two text files (files with .txt extension) showing that you have tested your program. One text file should be the output file used by your program, and the second one should contain the record of clerk's interactions with the program displayed in the Terminal Window. To make sure that the complete screen output of a program can be inspected in the Terminal Window of the BlueJ environment, tick "Unlimited buffering" from the "Options" menu in the Terminal Window. To produce the text file containing the content of the Terminal Window in BlueJ you can use "Save to file…" from the "Options" menu in the Terminal Window. If your input file contained different data than the example input data provided in this specification, then you also need to submit your input file (as a text file).

Deadline: The deadline for this coursework is **16:30 on Friday, 13th December 2019**.