P RC grade: Resit work (2019/20) for Coursework #2

Deadline for submission 16:00, Friday 16rd April, 2021

Submission information

The work should be emailed as a single ZIP file attachment to the module leader at C.Evans@mdx.ac.uk

Submission should comprise a single 'ZIP' file. This file should contain a separate, cleaned[1], NetBeans project for each of the three tasks described below. The work will be compiled and run in a Windows environment, so it is strongly advised that you test your work in a Windows environment prior to cleaning and submission.

When the ZIP file is extracted, there should be three folders (viewed in Windows explorer) representing three independent NetBeans projects as illustrated by Figure 1 below.

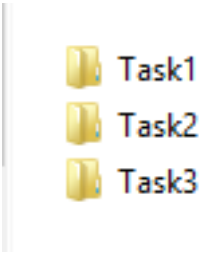

Figure 1: When the ZIP file is extracted there should be three folders named Task1, Task2 and Task3

Accordingly, when loaded into NetBeans, each task must be a separate project as illustrated by Figure 2 below.

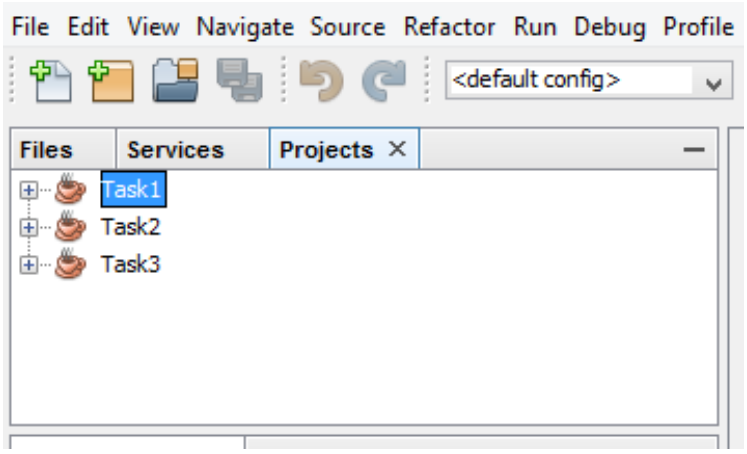

Figure 2: Each task must be a separate NetBeans Project

To make this easier, a template NetBeans project structure is provided.

[1] In the NetBeans project navigator window, right-click on the project and select 'clean' from the drop-down menu. This will remove .class files and reduce the project file size for submission.

Task 1 (20 marks)

Create a NetBeans 8 project for this task, named *Task1*.

You are required to write a Java 8 program that opens and reads a delimited data file that is located relative to the NetBeans project root folder. The delimited data file contains information about prize winning music albums ranked by sales. The data file is called *albums.txt*. The data file should not be altered and should be considered as a *read-only* data file.

Within the 'albums' data file, there are 20 entries that each represent a single album. On the first line of each album entry there are six data fields representing the following information (in order): the current sales ranking, the title of the album, the name of the artist, the year the album was released, and the total number of sales to date ('M' denotes million and 'K' denotes thousand). The first line is then followed by the list of song tracks for the album (one per line, in the original published order), with each track indicating the duration of the song (minutes:seconds) in braces. Each album entry is separated by a dashed line in the text file.

You are required to implement a Java class to represent an album. The program should parse the data file, create an object for each album, and store all the objects into a suitable collection. Figure 3 provides a partial UML class representation of the class that you will need to implement. It indicates required data members and accessor (i.e., *getter*) methods that map to those data members, and a constructor, and a *toString()* method. It is left to you to determine class data types and how the Album objects should be initialised.

Figure 3: UML class specification for *Album* class

Once all the objects are loaded into the collection, the program should present the User with a console-based menu to interact with the data set. This menu should loop until the User enters a character to exit the menu (e.g., zero as illustrated below). In addition to an exit option, the menu should offer three other options: list all albums, select a single album to view, and search in song titles.

On starting the program, the following menu should be displayed to the console:

```
List albums.........1
Select album........2
Search titles.......3
Exit................0

Enter choice:>
```

The User can simply exit the program by entering zero. The three other menu options allow the User to inspect the information in the data set (note again that this program is entirely *read-only* and there is no requirement to add, update or delete any part of the data set). The necessary interaction of the program with respect to these options is illustrated in Appendix A.

Note that console output should be neatly formatted, and some consideration will be given to formatting when the program is assessed. In particular, when the option to view a single trade company details is selected, it must result in the invocation of the *toString()* method for that particular *Album* object. You are encouraged to explore and utilise a *StringBuilder* object when implementing the *toString()* method for the *Album* class.

Task 2 (25 marks)

Create a new NetBeans project, called Task2.

For this task you are required to write a Java 8 program that incorporates a series of classes that represent the core logic of an application and provides a NetBeans 8 console user interface. The required Java application relates to a proposed software system to manage some aspects of charity running events over a particular year. Below is an initial description of the system domain with a few very simple use cases (i.e., just simple, natural language statements).

System domain

The charity organises several running events each year. There are two kinds of running event: fun runs which are over 5 km, and half-marathons. A 5 km fun run takes place in a park whereas a half-marathon takes place at either a park or a town. A venue of either kind (park or town) may host many charity running events over the course of the year.

Each charity running event has a specified date and start time, with each event being on a different date. Several hundred competitors can enter for each event. Each competitor can enter multiple events per year but can only have a single entry per event. Each entry for an event is allocated an event number which must be unique amongst all entries for that event. All competitors entering for a half-marathon must be 16 or over but there are no age restrictions on 5 km fun runs.

Use cases

1. **List Event Information**. The user of the software system identifies a running event by selecting it from a list provided by the system. Once an event is selected, the system displays the name of the event's venue, along with the current number of entries for that event. If the event is a half-marathon, the system also displays the number of water stations along the route of the race.

2. **List Venue Details.** The user of the software system identifies a venue by selecting it from a list provided by the system. Once a venue is selected, the system displays a list of the dates and start times of each running event taking place at that venue and, in the case of a park, the number of changing facilities available there. If there are no events taking place at the venue the system informs the administrator of this fact.
3. **Search Competitor's Event Entries.** The administrator identifies a competitor by entering a name, or partial name. For all competitors that match, the system displays the competitor's name and age. For each running event for which the competitor has an entry, the system displays the event date, along with the event number the competitor has been allocated for that event. The system also states whether the event is a 5 km fun run or a half-marathon.

Your task is to design a Java 8 program for the charity running event management system described above, which provides a console-based (text I/O) user interface that facilitates the three listed use cases. An initial analysis of the domain has already been completed, and a class model has been designed as shown in Figure 4.

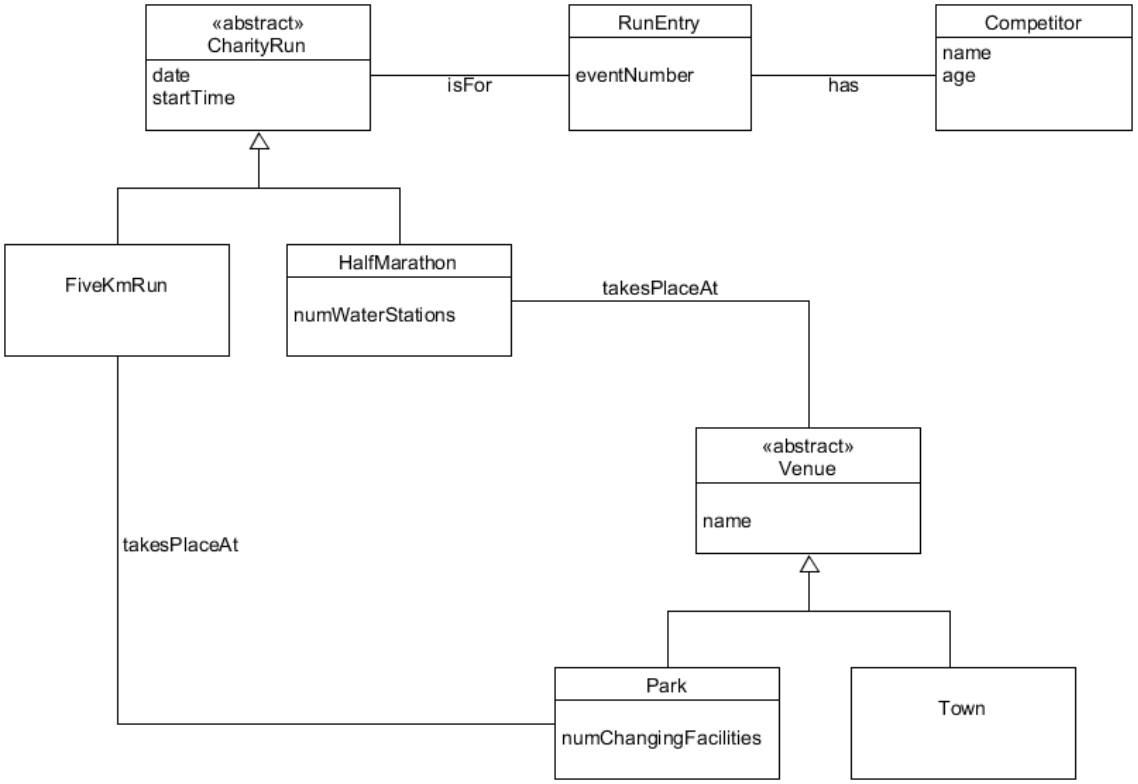

Figure 4: Class model of the charity run domain

The class diagram above is provided as a design for your Java implementation. That is to say, your application must include Java classes for the above class model as a minimum, i.e., you are allowed to extend the model if you wish to, but the above model must be implemented. You are not required to submit any UML for this task.

So that the use cases can be tested using the console-based User interface, your application will require some 'dummy' data to be pre-coded into the application. This should be achieved by 'hard-coding' data within the Java program. It should **not** use an external database link (e.g., such as an SQL database). There is no requirement to insert or update any data for the task via the use cases. All pre-coded data should be *read-only*. When assessing your overall design and implementation, some consideration will be given to applications that demonstrate good encapsulation of system components.

Task 3 (15 marks)

Create a new NetBeans project, called Task3.

For this task you are required to program a JavaFX Graphical User Interface (GUI), with a focus on layout and event handling.

This program must be coded as a JavaFX program following the approach described in Chapters 14 to 16 of the *kortext*. It must not be created using a visual drag-and-drop tool, and must not be an FXML program, neither of which have been taught on the module. If the submission is either a drag-and-drop application, or a FXML application, then it will simply be awarded zero marks.

The GUI application that you need to create is a basic 'surgery theatre controller' simulation. The required design is illustrated in Figure 5 below.

Figure 5: Surgery theatre controller simulation user interface

The application comprises several regions representing controls for clocks (a real time and an elapsed time), room temperature, humidity and pressure, lighting control, and a medical gases display. Each area on screen comprises JavaFX nodes and controls, such as buttons, labels, text, image views and graphical shapes.

a) Write a JavaFX program that replicates the GUI layout and design as illustrated in Figure 5. Styling of widgets must be achieved programmatically using the relevant 'setStyle' method for the respective Node objects. In other words, all styling should be contained within the Java code and not by use of a separate CSS file.

b) With reference to the seven regions in the display, User interaction should be as follows:

- The 'medical gases' region has no interaction and is purely a visual display.

- The ‘daytime clock’ region should have no user interaction but displays the current time which can be derived programmatically in Java from the CPU system clock.
- For the ‘elapsed time’ region (which displays seconds, minutes, and hours), a timer that increments each second should be started when the User selects the ‘start’ button. On selecting the ‘stop’ button the timer should halt. Selecting the ’start’ button again will continue the timer from its current setting (i.e., continual selection of start/stop should act like a pause and resume feature). Selecting the ‘reset’ button should set the elapsed time back to zero.
- For the temperature control region, the selection of the up/down buttons should increment and decrement the temperature display accordingly, in increments of 0.1 degrees Celsius. In addition, the temperature setting should not exceed 27.5 degrees Celsius, or drop below 10.0 degrees Celsius.
- For the humidity control region, the selection of the up/down buttons should increment and decrement the humidity display accordingly, in increments of one percent. In addition, the humidity setting should not exceed 55 per cent, or drop below 30 percent.
- For the pressure control region, the selection of the up/down buttons should increment and decrement the pressure display accordingly, in increments of 10 kPa. In addition, the pressure setting should not exceed 120 kPa, or drop below 50 kPa.
- For the light control region, each of the three light controls has buttons to increase or decrease the light setting. There are six light setting levels in each case. If the User presses the increase (+) button the light setting moves to the next level, and if the User presses the decrease (-) button the light seeing moves to the previous level. The result of this interaction is illustrated in Figure 6, which illustrates the results of interactions in several regions of the display.

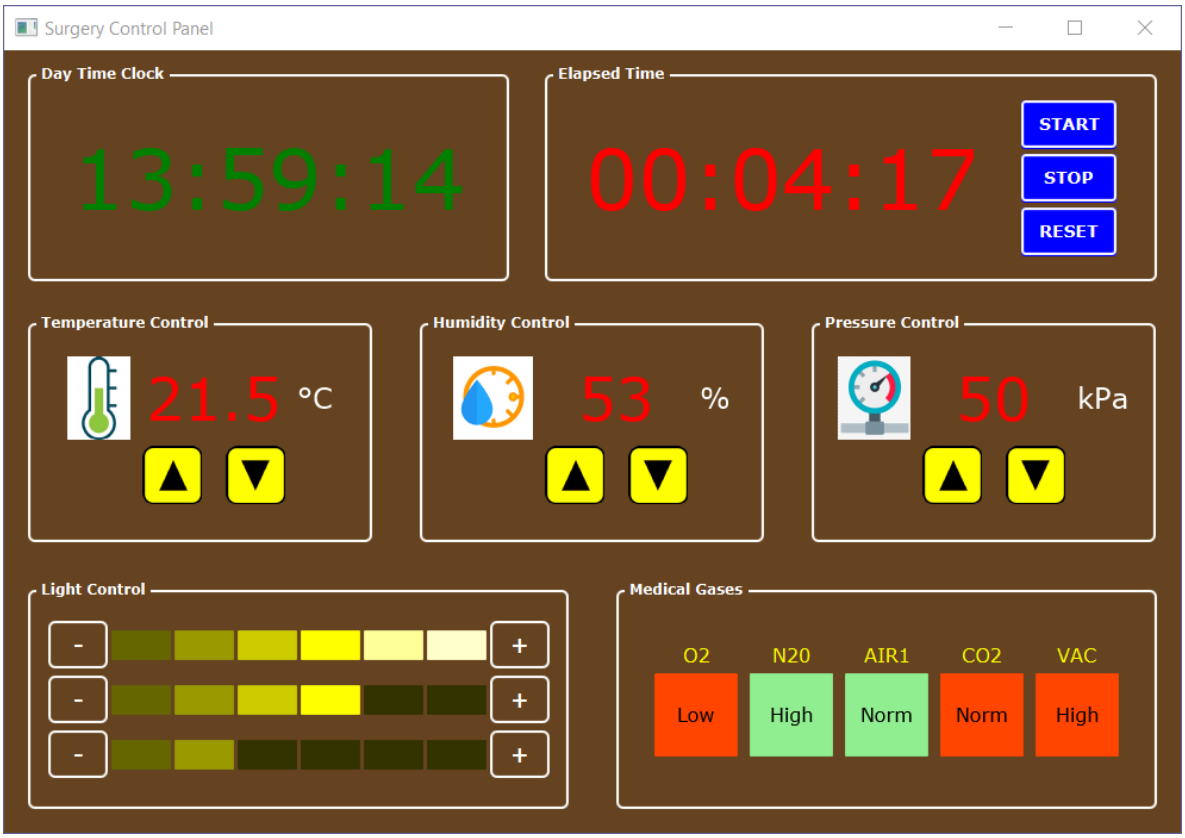

Figure 6: Result of event interaction with the surgery theatre controller simulation

Appendix A: Console interaction examples for Task 1

Option 1: list albums

On selecting option 1, the User should be presented with a neatly formatted listing of the albums. The data displayed should be ordered by the sales ranking, and include the album title, the artist, year of release and sales-to-date. To select a single album to view its full details, the User should be prompted to enter the sales ranking (i.e., option 2).

Rank	Title	Artist	Year	Sales
1	Whatever People Say I Am That's What I'm Not	Arctic Monkeys	2006	1.95M
2	Different Class	Pulp	1996	1.33M
3	Franz Ferdinand	Franz Ferdinand	2004	1.3M
4	The Seldom Seen Kid	Elbow	2008	1.11M
5	Dummy	Portishead	1995	920K
6	Elegant Slumming	M People	1994	759K
7	Screamadelica	Primal Scream	1992	715K
8	XX	The XX	2010	626K
9	Bring It On	Gomez	1998	502K
10	A Little Deeper	Ms Dynamite	2002	498K
11	The Hour OF Bewilderbeast	Badly Drawn Boy	2000	460K
12	An Awesome Wave	Alt-J	2012	445K
13	Myths Of The Near Future	Klaxons	2007	351K
14	New Forms	Roni Size/ Reprazent	1997	336K
15	Boy In Da Corner	Dizzee Rascal	2003	310K
16	Suede	Suede	1993	301K
17	Stories From The City, Stories From The Sea	PJ Harvey	2001	301K
18	Psychodrama	Dave	2019	241K
19	I Am A Bird Now	Antony & The Johnsons	2005	233K
20	Konnichiwa	Skepta	2016	207K

Option 2: select album

On selecting option 2, the User should be prompted to enter the sales ranking of a listed album (that was displayed when option 1 was chosen). Following this, all the details of that album should be displayed, as illustrated below. This should be achieved by invocation of the relevant *Album* object's *toString()* method. Console output should be neatly formatted.

```
List albums.........1
Select album........2
Search titles.......3
Exit................0

Enter choice:> 2

Enter album rank from list [1 - 20] :> 19

Album title:             I Am A Bird Now
Artist:                  Antony & The Johnsons
Year of release:         2005
Sales to date:           233K

Track list:
---------------------------------------
|No. |Title                 |Mins|Secs|
---------------------------------------
|1   |Hope There's Someone  |   4|  21|
|2   |My Lady Story         |   3|  33|
|3   |For Today I Am a Boy  |   2|  36|
|4   |Man Is the Baby       |   4|   9|
|5   |You Are My Sister     |   3|  59|
|6   |What Can I Do?        |   1|  40|
|7   |Fistful of Love       |   5|  52|
|8   |Spiralling            |   4|  25|
|9   |Free at Last          |   1|  36|
|10  |Bird Guhl             |   3|  14|
---------------------------------------


List albums.........1
Select album........2
Search titles.......3
Exit................0

Enter choice:>
```

Option 3: search song titles

On selecting option 3, the User should be prompted to enter a 'search string' to match against the song titles of all the albums. All searches should be case insensitive. The program should return a listing of any songs that match the search string, with the section that matches in upper-case. The returned matches should first display the album and title, followed by the songs that match the search string, including their track number. It is possible that multiple songs can match a given search string, in which case they should be listed by track number.

```
List albums.........1
Select album........2
Search titles.......3
Exit................0

Enter choice:> 3

Enter search word or phrase > night

-----
Artist (Franz Ferdinand) Album (Franz Ferdinand)
Matching song title(s):
-----
Track  2. Tell Her ToNIGHT

-----
Artist (M People) Album (Elegant Slumming)
Matching song title(s):
-----
Track  1. One NIGHT in Heaven

-----
Artist (The XX) Album (XX)
Matching song title(s):
-----
Track 10. NIGHT Time


List albums.........1
Select album........2
Search song titles...3
Exit................0

Enter choice:> 3

Enter search word or phrase > animal

-----
Artist (Suede) Album (Suede)
Matching song title(s):
-----
Track  2. ANIMAL Nitrate
Track 10. ANIMAL Lover


List albums.........1
Select album........2
Search titles.......3
Exit................0


Enter choice:>
```