FACULTY OF SCIENCE

ACADEMY OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

MODULE	**COMPUTER SCIENCE 2A** CSC02A2
CAMPUS	AUCKLAND PARK CAMPUS (APK)
SUMMATIVE ASSESSMENT OPPORTUNITY II	..SAO2_SPEC
DATE: 2023-06-14	**SESSION:** 10:00 - 12:30
ASSESSOR(S):	**MR A MAGANLAL**
	MR D OGWOK
MODERATOR:	**PROF HB VADAPALLI**
DURATION: 150 MINUTES	**MARKS:** 120

Please read the following instructions carefully:

1. You must complete the assessment **by yourself** within the prescribed time limits.
2. No communication concerning the assessment is permissible during the assessment session except with **ACSSE** staff members and **ACSSE** assistants/tutors.
3. You are bound by all university regulations including, but not limited, to assessment, plagiarism, and ethical conduct.
4. You may not leave the venue in the first 30 minutes of the paper.
5. You may not leave the venue in the last 30 minutes of the paper.
6. You may not directly take any code from any source, including your own previous submissions. All code must be written by yourself during the assessment.
7. Answer **each** question in a *separate* project.
8. This paper contains **2** question(s).
9. This paper consists of **5** page(s) excluding the cover page.
10. The paper and provided files can be found under
    `File Storage -> Computer Science 2A Practical SPEC Assessment Provided`
11. Submissions are to be uploaded to
    `File Storage -> Computer Science 2A Practical SPEC Assessment Submission`
12. Compulsory secondary submissions to Dropbox

Question 1: Problem Modeling and Text Processing

Problem statement

The company **FloorMart** is designing a system to process and store information about its **Food**s. **Food** is an abstract class that stores a *barcode* (**String**), 8 characters long, and *quality* (**String**). The **Food** *barcode* consists of 4 uppercase letters followed by 2 digits followed by 2 uppercase letters.

There are two (2) *kinds-of* **Food**:

- **CannedFood** stores *weight* (**double**)
- **FrozenFood** stores *temperature* (**int**) and *diet* (**EDietType**)

EDietType is an enumeration with values ***VEGETARIAN***;***LACTOFREE***;***MEAT***;***VEGAN***. Finally the **Store** class is used to store **Food**s. **Store** *has-a* **List** of **Food**s and operations to add/retrieve instances of **Food** to/from the **List**.

FloorMart has provided text files to process. Each file contains data for multiple **CannedFood** or **FrozenFood** instances. One instance can appear on a line with the following format:

CannedFood format

```
1 //Each element is separated by a TAB character
2 FOOD_BARCODE FOOD_QUALITY CANNEDFOOD_WEIGHT
```

FOOD_BARCODE - the barcode of the **Food**
FOOD_QUALITY - the quality of the **Food**
CANNEDFOOD_WEIGHT - the weight of the **CannedFood**

FrozenFood format

```
1 //Each element is separated by a TAB character
2 FOOD_BARCODE FOOD_QUALITY FROZENFOOD_DIET FROZENFOOD_TEMPERATURE
```

FOOD_BARCODE - the barcode of the **Food**
FOOD_QUALITY - the quality of the **Food**
FROZENFOOD_DIET - the diet (**EDietType**) of the **FrozenFood**
FROZENFOOD_TEMPERATURE - the temperature of the **FrozenFood**

Instructions

For the problem stated above do the following:

- Create a UML class diagram that has all the classes with their attributes and operations.[1]
- Create a **Java** project that contains the following:
  - Create all data classes with their attributes and operations.
  - Create a **FoodFileHandler** class with a static ***readStore*** method.
    The ***readStore*** method must accept a **File** handle as parameter and return a **Store** containing **Food** instances read from the file.
    The ***readStore*** method must read each line of the file and determine if it is **CannedFood** or **FrozenFood** using appropriate regular expressions. Based on the regular expression instantiate the correct class and add it to the **Store**.

~~ Assessment continues on the next page. ~~

[1] Constructors, basic accessors/mutators do not need to be shown in the UML Class Diagram.

Note:

- Data classes must follow correct Object Oriented Design Principals.
- Data classes in the must be placed in the **`acsse.csc2a.supermarket.model`** package.
- The **`FoodFileHandler`** class must be placed in the **`acsse.csc2a.supermarket.file`** package.
- The ***`readStore`*** method must make use of ***`Automatic Resource Management`***.
- The ***`readStore`*** method must handle any exceptions.

In the **`Java`** project create a **`Main`** class that will be the entry point of your program. The **`Main`** class must use the **`FoodFileHandler`** ***`readStore`*** method to *read and process* **all** the provided text files, one at a time. After reading each file iterate through the **`Store List`** and display only the following:

- Display **`CannedFood`** on the standard output stream in any format that you choose.
- Display **`FrozenFood`** on the standard error stream in any format that you choose.

Submission Guidelines

Your submission for this question must follow the naming convention below:

`SURNAME_INITIALS_STUDENTNUMBER_ CSC02A2_2023_Q1`

Zip the project and submit the zipfile onto EVE.

Multiple uploads

If you already have submitted once and want to upload a newer version then submit a newer file with the same name as the uploaded file in order to overwrite it.

Important Note

Submissions which **do not compile** will be capped at 40%!

Marks are awarded for a correctly functioning application and not for having related code.

~~ Assessment continues on the next page. ~~

Question 1 Mark Sheet

(a) UML Class Diagram
- i. Classes **[03]**
- ii. Attributes **[03]**
- iii. Operations **[01]**
- iv. Has-a relationship **[02]**
- v. Is-a relationship **[01]**

(b) **`Food`** class
- i. Abstract **[01]**
- ii. Attributes **[02]**
- iii. Operations **[02]**

(c) **`EDietType`** class - Correct Structure **[02]**

(d) **`CannedFood`** class
- i. Correct inheritance **[01]**
- ii. Attributes **[01]**
- iii. Operations **[01]**

(e) **`FrozenFood`** class
- i. Correct inheritance **[01]**
- ii. Attributes **[02]**
- iii. Operations **[02]**

(f) **`Store`** class
- i. Attributes **[01]**
- ii. Operations **[02]**

(g) **`FoodFileHandler`** - ***`readStore`*** method
- i. Regular expression for **`CannedFood`** **[03]**
- ii. Regular expression for **`FrozenFood`** **[04]**
- iii. Correctly using ***`Automatic Resource Management`*** **[04]**
- iv. Read file **[03]**
- v. Test using regular expressions **[04]**
- vi. Instantiation of correct instance **[02]**
- vii. Return **`Store`** **[01]**
- viii. Exception handling **[03]**

(h) **`Main`** class
- i. Use ***`readStore`*** method to get **`List`** of **`Food`** **[01]**
- ii. Display all instances to correct stream. **[02]**

(i) Coding convention + Packages + Commenting **[05]**

Total: 60

~~ Assessment continues on the next page. ~~

Question 2: Problem Modeling and Binary Processing

The company **Mages of the Beach** need to automate the processing of **`Player`** data. Currently the data is stored in a binary file format using only primitive types. The files stored following format:

Player data file format

```
// Data is in binary format, first int indicates an item count
ITEM_COUNT
// Order and types for attributes of Player
PLAYER_PLAYER_ID PLAYER_ROLE PLAYER_PROWESS PLAYER_MANA
```

`ITEM_COUNT` - Number of Players in the file (int)
`PLAYER_PLAYER_ID` - the unique *playerID* of the **`Player`** (String)
`PLAYER_ROLE` - the *role* of the **`Player`** (String)
`PLAYER_PROWESS` - the *prowess* of the **`Player`** (int)
`PLAYER_MANA` - the *mana* of the **`Player`** (double)

Mages of the Beach requires conversion of these files into a new binary file format that uses **`Java`** Objects instead. Additionally they require the contents of the file to be sorted correctly. Comparison between two **`Player`** instances can be done by firstly comparing their *prowess*, then if those are equal comparing their *mana*.

Instructions

For the problem stated above create a **`Java`** project that contains the following:

- Create the **`Player`** class that stores the required data, implements the correct interface for serialization and **`Comparable<T>`** interface.
- Create a **`PlayerFileHandler`** class with the following static methods:
  - ***`readPlayerData`*** method accepts a **`File`** handle as parameter and returns a **`List`** of **`Player`**s read from the file.
  - ***`writePlayerObj`*** method accepts a **`File`** handle and **`List`** of **`Player`**s as parameters and writes the **`Player`**s as objects to the file.

Note:

- The **`Player`** class must be placed in the **`acsse.csc2a.fantasy.model`** package.
- The **`PlayerFileHandler`** class must be placed in the **`acsse.csc2a.fantasy.file`** package.
- The methods in the **`PlayerFileHandler`** class must each make use of ***`Automatic Resource Management`*** and handle any exceptions.

In the **`Java`** project create a **`Main`** class that will be the entry point of your program. The **`Main`** class must use the **`PlayerFileHandler`** to *read and process* **all** the provided text files, one at a time. First read the file with ***`readPlayerData`*** method to get a **`List`** of **`Player`**s, sort the **`List`** and save the **`List`** to a new file with the ***`writePlayerObj`*** method.

.

~~ Assessment continues on the next page. ~~

Submission Guidelines

Your submission for this question must follow the naming convention below:

`SURNAME_INITIALS_STUDENTNUMBER_ CSC02A2_2023_Q2`

Zip the project and submit the zipfile onto EVE.

Multiple uploads

If you already have submitted once and want to upload a newer version then submit a newer file with the same name as the uploaded file in order to overwrite it.

Important Note

Submissions which **do not compile** will be capped at 40%!

Marks are awarded for a correctly functioning application and not for having related code.

Question 2 Mark Sheet

(a) **`Player`** class

i. Implement interfaces **[02]**
ii. Attributes **[04]**
iii. Operations **[04]**
iv. ***`compareTo`*** **[05]**

(b) **`PlayerFileHandler`** **`readPlayerData`**

i. Create **`List`** of **`Player`** **[02]**
ii. Correctly using ***`Automatic Resource Management`*** **[05]**
iii. Read **`Player`**s from file **[06]**
iv. Append new instance to **`List`** of **`Player`** **[02]**
v. Return **`List`** of **`Player`** **[02]**
vi. Exception handling **[03]**

(c) **`PlayerFileHandler`** **`writePlayerObj`**

i. Correctly using ***`Automatic Resource Management`*** **[05]**
ii. Write **`List`** of **`Player`** to file **[02]**
iii. Exception handling **[03]**

(d) **`Main`** class

i. Read file with **`readPlayerData`** method **[02]**
ii. Sort **`List`** of **`Player`** **[06]**
iii. Write file with **`writePlayerObj`** method **[02]**

(e) Coding convention + Packages + Commenting **[05]**

Total: 60

~~ THE END ~~