UNIVERSITY OF JOHANNESBURG

Computer Science 2A

Practical Assignment 03

2016-02-23

Time: Deadline — 2016-03-01 12h00 Marks: 100

This practical assignment must be uploaded to eve.uj.ac.za **before** 2016-03-01 12h00. Late or incorrect submissions **will not be accepted**, and will therefore not be marked. You are **not allowed to collaborate** with any other student.

Good coding practices include a proper coding convention and a good use of JavaDoc comments. Marks will be deducted if these are not present. Every submission **must** include a batch file. See the reminder page for more details.

The Java Development Kit (JDK) has been installed on the laboratory computers along with the Eclipse Integrated Development Environment (IDE).

This practical aims to solidify your understanding of basic Java Object Orientation and problem statements.

Utopia is on the brink of destruction[1]. Unknown alien lifeforms have attacked planet Earth. As a desperate final attempt at survival, you and other other crew members have been sent to find sanctuary aboard the starship SS Invictus.

Unfortunately shortly after lift-off your ship was attacked! Your ship barely makes it to orbit. Miraculously your ship survives and enters hyperspace. Not long after your ship crashes on a distant planet.

You are stuck in your crew quarters on the ship. The door is jammed and you cannot get out. The info screen in your room still works. After a few minutes of trying to figure out how to open the door you are able to run a basic console. Having prior knowledge of the ship you know that the systems all run on the Java Virtual Machine (JVM) and also have the JDK installed. You have access to some of the files on the ship but one stands out amongst them: the crew roster. The file is in plain text, however, there appears to be some file corruption.

In order to reach one of the engineering crew members you need to create a Java Application. Create the necessary Java classes to solve the following problem. Remember to follow proper coding conventions as your life, as well as lives of crew members depend on the quality of application you produce!

[1]Disclaimer - This series of problem statements are a work of fiction. Names, characters, businesses, places, events and incidents are either the products of the author's imagination or used in a fictitious manner. Any resemblance to actual persons, living or dead, or actual events is purely coincidental.

The crew roster stores the information about every crew member aboard the ship. The file stores the details of one crew member per line and has the following format:

```
CREW_ID CREW_RANK CREW_SURNAME CREW_TYPE CREW_SPECIAL
```

where

CREW_ID Unique ID of the crew member, 6 characters long

CREW_RANK Rank of the crew member, abbreviated 3 letter code.

CREW_SURNAME Surname of the crew member, can be any length

CREW_TYPE Type of the crew member, any length but all capital letters(will become more important in later practicals).

CREW_SPECIAL Special field which depends on crew type, any length.

Using the information provided in the problem statement above do the following:

- Create a **CrewMember** class with the required attributes[2].
- Create a **CrewRoster** class with a class method ***readRoster***:
  - The method will take a filename as a parameter.
  - The method will read each line of the file and test the line to check it is valid as per the above description of the format. If a line is not valid it is printed to the error stream.
  - The method will return an array of **CrewMember** instances which is created from the file.
- Create a **Main** class.
  - This class will be responsible for calling the ***readRoster*** method from the **CrewRoster** class.
  - Printing out the elements from array of **CrewMember**s which is returned from the ***readRoster*** method.

Remember to place the relevant classes into the **acsse.csc2a** package[3].

[2]Hint: remember that accessor and mutator methods for attributes are required.
[3]Hint: The **Main** class does not need to be in a package

Marksheet

1. UML class diagram [05]
2. **`CrewMember`** class
   (a) Attributes (4 marks per attribute) [20]
3. **`CrewRoster`** class - ***`readRoster`*** method
   (a) Open file [02]
   (b) Validate line [10]
   (c) Create **`CrewMember`** instance and add to array [02]
   (d) Close file [01]
   (e) Exception Handling [05]
4. **`Main`** class [05]
5. Packages [05]
6. Coding convention (structure, layout, OO design) and commenting (normal and JavaDoc commenting). [10]
7. Correct execution [35]

NB

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

Execution marks are awarded for a correctly functioning application and not for having some related code.

Reminder

Your submission must follow the naming convention as set out in the general learning guide.

```
SURNAME_INITIALS_STUDENTNUMBER_SUBJECTCODE_YEAR_PRACTICALNUMBER
```

Example

Surname	Berners-Lee
Initials	TJ
Student number	209912345
Module Code	CSC2A10
Current Year	2016
Practical number	P00

```
Berners-Lee_TJ_209912345_CSC2A10_2016_P00
```

Your submission must include the following folders:

- `bin` - *(Required)* Should be empty at submission but will contain runnable binaries when your submission is compiled.
- `docs` - *(Required)* Contains the batch file to compile your solution, UML diagrams, and any additional documentation files. Do not include generated JavaDoc.
- `src` - *(Required)* Contains all relevant source code. Source code must be places in relevant sub-packages!
- `data` - *(Optional)* Contains all data files needed to run your solution.
- `lib` - *(Optional)* Contains all libraries needed to compile your solution.

NB

Every submission **must** include a batch file. This batch files must contain commands which will compile your Java application source code, compile the associated application JavaDoc and run the application. **Do not** include generated JavaDoc in your submission. All of the classes/methods which were created/updated need to have JavaDoc comments.