Computer Science 2A

Practical Assignment 05

Assignment date:	2024-03-19
Deadline	2024-03-19 17h30
Marks: 110	

This practical assignment must be uploaded to eve.uj.ac.za **before** 2024-03-19 17h30. Late[1] 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 documentation. Marks will be deducted if these are not present. Every submission **must** include a batch file unless stated otherwise.

The **reminder page** includes details for submission. Please ensure that **ALL** submissions follow the guidelines. The reminder page can be found on the last page of this practical.

This practical aims to Introduce Generics and BinaryIO [2]

Within the **Firework Management Bureau (FMB)** [3] drama unfolds! A fellow PyroTechnician, perhaps more of a PyroAmateur, has been given the boot for their... let's say, ”unique” approach to fireworks display. Just remember, it's not just about making things go boom; it's about making them go boom with style. Their performance was memorable, though not for reasons they'd hoped. While they've sashayed away with the memories of their less-than-stellar display, they've left behind a digital breadcrumb - a binary file detailing their explosive (or not) masterpiece.

Now, the FMB, in their infinite wisdom and with a dash of desperation, have turned to you. Yes, you, with your semi-impressive coding prowess and questionable reading comprehension. They're offering you a shot at redemption, a chance to prove your mettle (or metal, if you're into that kind of pun). So, dust off your coding hat and show the FMB what you're made of by delivering on the following.

First, your task involves developing an `Entity` class within the `acsse.csc2a.fmb.model` package. This abstract class is designed to be immutable, ensuring its attributes cannot be modified after creation, thereby making them effectively read-only. Each instance of `Entity` should include the following, along with the necessary accessors:

- `xLocation` – an integer representing the `Entity`'s position along the x-axis.
- `angle` – a double precision number indicating the `Entity`'s angle of inclination.
- `toString` – an overridden method to generate the textual representation of the object.

[1] Alternate arrangements for exceptional circumstances will been posted on eve.

[2] Please note that the practical is due today. Good job on reading though ;)

[3] 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.

Subsequently, you are to create a `FireworkEntity` class in the `acsse.csc2a.fmb.model` package. This class, similar to its parent, is designed with immutability in mind and should include the following attribute, alongside the relevant accessor(s):

- `firework` – a reference to a `Firework`.

Moving forward, your next step is to manage reading from a binary file. The binary file is structured using only **primitive** data types as follows:

```
Binary File Format
FD_ID F_ID X_LOCATION ANGLE ... F_ID X_LOCATION ANGLE
```

The attributes within the binary file are defined as:

`FD_ID` – Represents the `FireworkDisplay`'s Unique Identifier.
`F_ID` – Represents the Firework's Unique Identifier.
`X_LOCATION` – Indicates the `Firework`'s position on the x-axis.
`ANGLE` – Specifies the `Firework`'s angle of inclination.

To extract information from the binary file, follow these requirements:

1. Implement the `Entity` class hierarchy.
2. Develop a `OrchestrationFileHandler` in the `acsse.csc2a.fmb.file` package with a defined `readLayout` method as follows:
   (a) Accept at least the binary file's name as input.
   (b) Read the binary file's contents and populate an `ArrayList` of Entities.
   (c) Return the populated Entities.
3. Create a Main method demonstrating the process of reading the binary file and displaying its contents.

In this practical exercise, leverage the provided **p05.jar** file and accompanying **JavaDoc** to accomplish your tasks. Note:

- *FD0001.txt* is a text file containing information for the FireworkDisplay with ID "FD0001".
- *layout_1.dat* is a binary file containing the orchestration information.

Hints

- In the **JavaDoc** and **p05.jar**, you will find:
  - New methods in the `FireworkDisplay` class.
  - A `DisplayFileHandler` with an implementation you should be familiar with.
- Text files named according to their **FireworkDisplay** ID. You can use this aspect to find the Firework information.
- A reminder not to over-think the process, but to use the resources provided to you.

Marksheet

1. Updated UML class diagrams for all classes. [15]
2. `Entity` Class [7]
3. `FireworkEntity` Class [6]
4. `OrchestrationFileHandler` Class
   (a) definition [02]
   (b) `readLayout` method [20]
5. `Main`
   (a) Object creation [5]
   (b) Read the Binary File [5]
   (c) Display the Binary File's contents [5]
6. Packages [05]
7. Coding convention (structure, layout, OO design) [05]
8. Commenting (normal and JavaDoc commenting) [05]
9. Correct execution (if it doesn't run from your batch file you get 0) [30]

NB

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

Practical marks are awarded subject to the student's ability to explain the concepts and decisions made in preparing the practical assignment solution. (Inability to explain code = inability to be given marks.)

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

Reminder

Your submission must follow the naming convention below.

```
SURNAME_INITIALS_STUDENTNUMBER_SUBJECTCODE_YEAR_PRACTICALNUMBER
```

Example

Surname	Berners-Lee	**Module Code**	CSC02A2
Initials	TJ	**Current Year**	2024
Student number	209912345	**Practical number**	P05

```
Berners-Lee_TJ_209912345_ CSC02A2_2024_P05
```

Your submission must include the following folders:

Folder	**State**	**Purpose**
`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. All files must be in **PDF** format. Your details must be included at the top of any **PDF** files submitted. **Do not include generated JavaDoc.**
`src`	*Required*	Contains all relevant source code. Source code must be places in relevant sub-packages! Your details must be included at the top of the source code.
`data`	*Optional*	Contains all data files needed to run your solution.
`lib`	*Optional*	Contains all libraries needed to compile and run your solution.

NB

Every submission **must** include a batch file that contains commands which will:

- Compile your Java application source code.
- Compile the associated application JavaDoc.
- 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.

Multiple uploads

Note that only **one** submission is marked. 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.