Computer Science 2A

Practical Assignment 04

Assignment date:	2024-03-12
Deadline	2024-03-12 17h00
Marks: 134	

This practical assignment must be uploaded to eve.uj.ac.za **before** 2024-03-12 17h00. 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 Advanced Object Orientation [2]

As previously discussed with the **Firework Management Bureau (FMB)** [3] you will need to validate **`Firework Display`** transmissions, sent by the lead **`PyroTechnician`**. However, the **`FMB`** forgot to mention that **`Fireworks`** come in two types: `Rocket Fireworks` and `Fountain Fireworks`. Each `Firework` possesses the following attributes:

- Firework ID - a unique alphanumeric identifier for the firework
- Firework Name - the official name of the firework
- Firework Fuse Length - the delay before explosion (in seconds)
- Firework Colour - the color (`E_COLOUR`) emitted by the firework upon detonation

Furthermore, the two categories of `Fireworks` exhibit additional, type-specific properties:

For `Rocket` Fireworks:

- Rocket Black Powder Colour - the trail color (`E_COLOUR`) emitted by the Rocket before explosion
- Rocket Star Count - the quantity of Star Spores released upon explosion
- Rocket Star Radius - the dispersion radius of the Star Spores from the explosion's epicenter

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

[2] Please note that the practical is actually due on 2024-03-19 12h00. This is to test who actually reads their practicals. If you read this, do not have a reaction or tell anyone else ;)

[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.

For `Fountain` Fireworks:

- Fountain Duration - the total display time for the Fountain
- Fountain Transition Colours - a sequence of colors (`E_COLOUR`) exhibited throughout the Fountain's display duration

Data formatting for `Rocket` and `Fountain` Fireworks is as follows:

Rocket Information Format

```
// Data is separated with TABs
F_ID F_NAME F_FUSE_LENGTH F_COLOUR R_STAR_COUNT R_RADIUS R_BP_COLOUR
```

Fountain Information Format

```
// Data is separated with TABs
F_ID F_NAME F_FUSE_LENGTH F_COLOUR FT_DURATION FT_TRANSITION_COLOURS
```

The attributes of each `Firework` type are meticulously defined as follows:

`F_ID` - Represents the Firework's Unique Identifier. **`Rockets`** are distinguished by an *FR* prefix, whereas **`Fountains`** commence with an *FF* prefix, both followed by a sequence of 6 digits to ensure uniqueness (e.g., FR123456, FF123456).

`F_Name` - Denotes the recorded name of the Firework, providing an easily recognizable identifier such as "Bouncing Betty."

`F_FUSE_LENGTH` - Specifies the delay before explosion, measured in seconds (e.g., 1.2 seconds), allowing precise timing for display coordination.

`F_COLOUR` - Indicates the primary **`E_COLOUR`** emitted by the Firework upon detonation, defining its visual impact.

`R_STAR_COUNT` - The count of Star Spores a Rocket Firework releases at the point of explosion. This number must be at least 1, providing a measure of the display's intensity (e.g., 20 stars).

`R_RADIUS` - The radial distance, in metres, that Star Spores cover from the Rocket's explosion point, determining the spread of the display (e.g., 1.5 metres).

`R_BP_COLOUR` - Defines the colour of the smoke trail produced by the Rocket prior to its explosion, adding to the visual spectacle (e.g., RED).

`FT_DURATION` - Represents the Fountain Firework's total display time in seconds, which can vary significantly (e.g., 20, 10.5, 5.0 seconds) depending on the design.

`FT_TRANSITION_COLOURS` - Enumerates all transition colours produced by the Fountain during its display. These colours are evenly spaced throughout the duration and formatted within brackets, separated by a pipe symbol (e.g., [YELLOW|CYAN]), to illustrate the dynamic colour changes.

First and foremost, the **FMB** insists, with a wink and a nudge, that a **`Firework Display`** simply refuses to come to life without the guiding hand (and occasionally, the convenient shoulder to cry on) of a **`PyroTechnician`**. Seriously, who else would they direct their standing ovations to—or, in the rare event of a less-than-stellar performance, find conveniently available for a friendly, "constructive" chat? That being said, a **`PyroTechnician`** is mandatory for the creation of a **`Firework Display`**.

To safeguard the sparkle and ensure every burst of color hits the night sky just right, the **FMB** mandates a dive into the deep end of advanced object-oriented programming. The upgrades on

the docket are non-negotiable but promise to be as enlightening as a perfectly timed firework finale:

1. Implement a `Firework` class hierarchy, acknowledging that direct instantiation of a `Firework` object is impractical.
2. Override the `toString` method in all classes to enhance object representation.
3. Introduce the `IDisplayable` interface in the `acsse.csc2a.fmb.model` package with a `display` method for consistent printing of the class's textual representation to the console.
4. Enhance the `DisplayFileHandler` class for improved data encapsulation and processing:
   (a) Incorporate a regular expression pattern in each class to verify the completeness of data for object construction.
   (b) Implement a static `validate` method in each class to assess data validity based on the provided line, issuing informative error messages as necessary.
   (c) Design a static `processLine` method in each class for object instantiation based on the provided line. Please note that in cases where the class needs an additional parameter to instantiate an object, it should be provided as an additional parameter.
   (d) Modify the `readDisplay` method in `DisplayFileHandler` to utilize class-specific methods for data validation and processing.

In this practical exercise, utilize the provided **p04.jar** file and accompanying text files, which are categorized as follows:

- *Dirty files* contain errors preventing the formation of a Firework Display.
- *Partial files* may result in a Firework Display, albeit with potential corrupt Firework data.
- *Clean files* are free from errors and will successfully generate a Firework Display.

To test the application, call the **`P04MainTester`** class's ***`main`*** method (provided to you in the **p04.jar**) in **your `Main`** class.

Marksheet

1. Updated UML class diagrams for all classes **[15]**
2. **`Firework`** Inheritance hierarchy **[5]**
3. **`IDisplayable`** Interface
   (a) definition **[02]**
   (b) ***`display`*** method **[02]**
4. Each **`acsse.csc2a.fmb.model`** Class
   (a) Override ***`toString`*** method in each class **[10]**
   (b) Realise the **`IDisplayable`** interface **[10]**
   (c) Define a final Regex Pattern **[10]**
   (d) Implement the static ***`validate`*** method **[10]**
   (e) Implement the static ***`processLine`*** method **[10]**
5. **`DisplayFileHandler`**
   (a) modify ***`readDisplay`*** method to use Class-specific methods for data validation and processing. **[10]**
6. **`Main`**
   (a) The project runs with the **`P04MainTester`**'s ***`main`*** method provided in the **`p04.jar`** **[05]**
7. Packages **[05]**
8. Coding convention (structure, layout, OO design) **[05]**
9. Commenting (normal and JavaDoc commenting) **[05]**
10. 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**	P04

`Berners-Lee_TJ_209912345_ CSC02A2_2024_P04`

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.