Computer Science 2A

Practical Assignment 04

Assignment date:	2024-04-23
Deadline	2024-04-30 12h00
Marks: 208	

This practical assignment must be uploaded to eve.uj.ac.za **before** 2024-04-30 12h00. 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 familiarise you with Advanced OO, JavaFX GUI, and the Visitor Design Pattern.

The **Firework Management Bureau (FMB)**[2] is pleased with your progress but now require a **Graphical User Interface(GUI)** for your application (to make it more modern).

The FireworkDisplay is represented as a 2D grid that will be representing the area where the `Fireworks` will be setup. You can assume the grid will always be 15x15. Create a `FireworkDisplayCanvas` class that extends `Canvas` in the `acsse.csc2a.fmb.gui` package. This node will be responsible for drawing a virtual side view of the re-enactment of the FireworkDisplay. The `FireworkDisplayCanvas` will store an ArrayList of `FireworkEntity` objects. Implement the `redrawCanvas` method to draw a grid where each grid cell is 50 pixels^2. At the correct column (x-location) at the bottom of the grid, draw the `Firework` located in the `FireworkEntity` as Follows:

1. `RocketFirework`: They will be drawn as rectangles with their height slightly bigger than their width.
2. `FountainFirework`: They will be drawn as a circle.

Fireworks should be coloured according to their stored `E_COLOUR`, and rotated according to their angle. We will assume a `Firework` that is pointed straight up, to be angled 90 degrees, one that points to the left as having an angle of 180, and one that points to the right as having an angle of 0. Of course we do not want to risk potentially crashing our running code, so we will make use of the Visitor Design Pattern to handle the drawing of the Entities.

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

[2] Disclaimer - This series of problem statements are a work of ficion. 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.

You need to create the required interfaces and make changes to the `Entity` class to display Entities on the canvas using a Visitor.

You must update application with the following:

- Create appropriate `AbstractVisitable` interface.
- Update `Entity` classes.
- Create appropriate `AbstractVisitor` interface.
- Create appropriate `ConcreteVisitor` class.
- Create a `FireworkDisplayCanvas` that makes use of the Visitor.

Create a `FireworkDisplayPane` class that extends `StackPane` (*javafx.scene.layout.StackPane*). This will act as the root node for your Scene[3][4]. The `FireworkDisplayPane` class is responsible for storing nodes that will be responsible for opening a Layout Binary File (from the last practical) and displaying the contents on a GUI.

The `FireworkDisplayPane` class will have the following components:

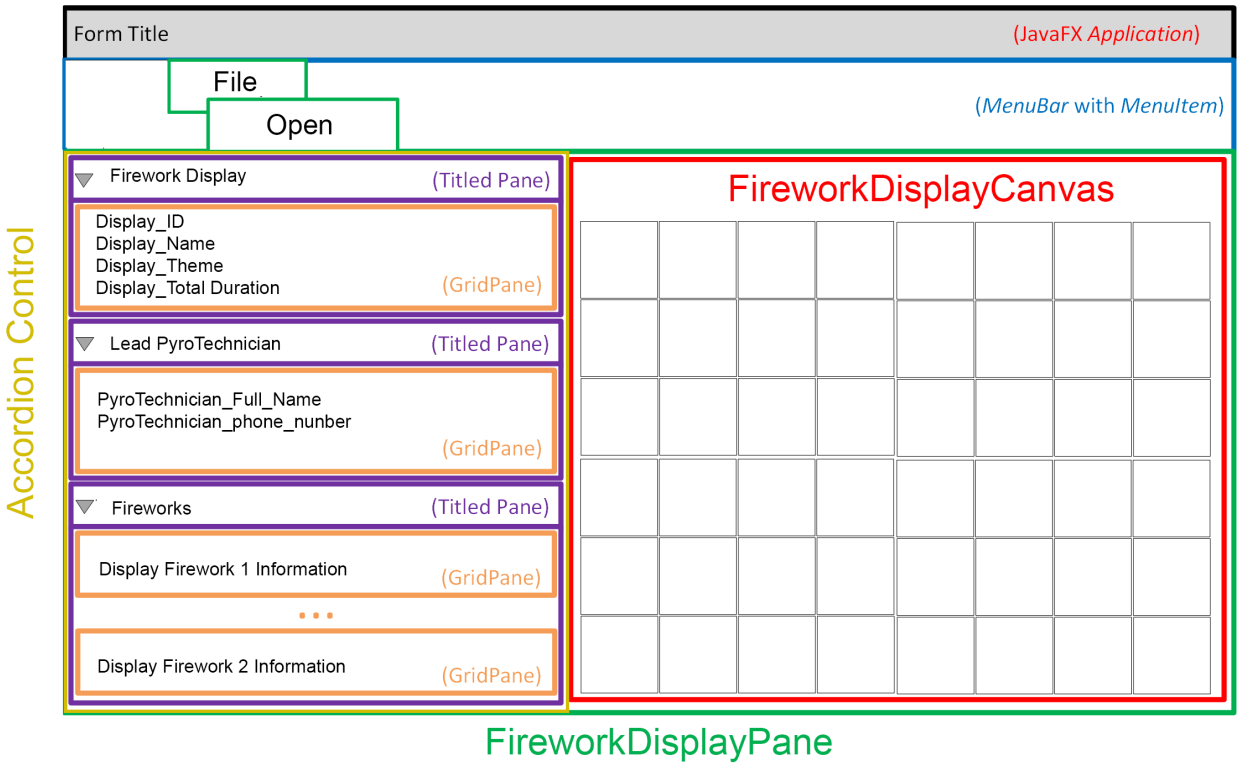
- A `DisplayBundle` instance that is created by the `OrchestrationFileHandler.readLayoutFile` method once a file is selected. This instance provides you with an ArrayList<FireworkEntities> and the associated display object.
- A `FireworkDisplayCanvas` that will handle displaying the FireworkDisplay re-enactment.
- A `MenuBar` containing a `MenuItem` to open a `FileChooser` to select a layout binary file.
- An `Accordion` control to hold the controls that store the FireworkDisplay's information, including:
  - `TitledPane`: Provides an expandible container with a label to store other nodes
  - `GridPane`: Provides a grid to store nodes in rows and columns
  - `Label`: Provides a text label
- You may use the following optional controls (or others that you find useful):
  - `ScrollPane`: Provides a scrollable container ideal for storing multiple Fireworks
  - `HBox` and `VBox`: Provide containers for horizontal and vertical stacking of nodes, respectively

To summarise: Update your `Java` application with the following:

- Create `FireworkDisplayCanvas` class.
- Implement the `redrawCanvas` method.
- Update `FireworkDisplayPane` with the following:
  - Dock the `Firework Display` information to the left of the `Scene` by using an appropriate layout node.
  - Dock the `FireworkDisplayCanvas` to the center of the `Scene` by using an appropriate layout node (an example is shown below).
- `Main` should not be changed. Its should still create an `Application` and set the `Scene` to display all the `FireworkEntity` instances to the user on the grid.

Here is an example of what the layout of the application should be:

[3] Hint: JavaFX uses a Scene to be loaded onto the primaryStage so that the Scene can be displayed.
[4] Created and used in the `Main` class



Then make the following changes to your `Main` class:

- Make the class extend `Application` (*javafx.application.Application*)
- Remove unnecessary imports
- Implement the missing `start` method required by the JavaFX `Application`
- In your `main` method, launch the JavaFX Application (this should be the only code in the `main`)
- Your class you will need to instantiate an instance of the `FireworkDisplayPane`
- Add the `FireworkDisplayPane` instance to a `Scene` and load it onto the `Stage` provided by the `Application`
- Show the `Stage` (with the loaded `Scene`)

Remember to place the relevant classes into the `acsse.csc2a.fmb` subpackages[5]

In this practical exercise, leverage the provided **p07.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

- The redrawCanvas method should draw the grid border, instantiate the visitor and provide it with the graphics context. The drawing of the Fireworks themselves will happen in the visitor when the `Visitables accept` it.

[5]Hint: UI classes such as `FireworkDisplayPane` should appear in the `acsse.csc2a.fmb.gui` subpackage.

Marksheet

1. Updated UML class diagrams for all classes. [15]
2. `FireworkDisplayCanvas`
   (a) Implement `redrawCanvas` [02]
   (b) Draw grid [02]
   (c) Draw `FireworkEntity`s [05]
3. `FireworkDisplayPane`
   (a) `DisplayBundle` instance [01]
   (b) `MenuBar` with `MenuItem` [02]
   (c) `FileChooser` to select file [02]
   (d) Create `DisplayBundle` by calling `OrchestrationFileHandler.readLayoutFile` [01]
   (e) `Accordion` control [05]
   (f) `TitledPane`s [10]
   (g) Use of `GridPane`s with `Label`s and `TextField`s to display team information [20]
   (h) Team info on the left and Canvas in the center [04]
4. `Visitor`
   (a) `AbstractVisitable` interface [5]
   (b) `Entity` sub classes implement `AbstractVisitable` [5]
   (c) `AbstractVisitor` interface [5]
   (d) `ConcreteVisitor` class [10]
   (e) `FireworkDisplayCanvas` uses the Visitor correctly [50]
5. `Main`
   (a) Extends `Application` [01]
   (b) Has `start` method [02]
   (c) `main` launches application [01]
   (d) Has `TeamPane` instance [02]
   (e) Adds `TeamPane` to `Scene` and loads it onto `Stage` [08]
6. Packages [05]
7. Coding convention (structure, layout, OO design) [05]
8. Commenting (normal and JavaDoc commenting) [05]
9. Correct execution [35]

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.