Classes or Entities in the Navigator

Campus Map

The Campus Map is represented as a **row-column grid or 2D-Array**, where each cell corresponds to a specific location on the campus. Every Campus Map should have a name. The map is initialised at runtime using command-line arguments, and the map layout is loaded from a file. Once loaded, the map remains fixed for the entire run of the program.

> **Modification**: The campus map now has a name instead of an ID. ArrayLists can be used to create a Campus Map.

MapPositionType

Each cell in the map is of a specific type (e.g., `START`, `PLACE`, `OPEN`, `BOUNDARY`, or `RESTRICTED`) and may or may not contain some places. The cell type in the map can be represented using the following symbols:

- `#` → `BOUNDARY` — a blocked location, not walkable.
- `.` → `OPEN` — a walkable path with no events.
- `X` → `RESTRICTED` — an area marked as restricted and cannot be entered.
- `S` → `START` — the student's starting location.
- `PLACE` — a place has different symbols as defined below.

Place

If a cell is of type `PLACE`, then it has some other attributes too. A place is defined using a place type, name, a score associated with the place and a boolean value that represents if the place has been visited before. Some places have a list of scheduled events. Place Types are defined as below:

- `C`→ `CAFETERIA` - cafeteria where you can get food.
- `L`→ `LIBRARY` - library to study.
- `G` → `SPORTS_CENTRE` - a place to visit sports facilities and exercise.
- `@` → a `LECTURE_HALL` or `EVENT_HALL` that contains one or more scheduled events.
- `E` → `EMPTY PLACE` - a location (Lecture or Event Hall) that contained events before, but all events have been visited.

Additionally, `EVENT_HALL` and `SPORTS_CENTRE` are bookable for professional use. Whereas, `CAFETERIA` has a food menu, `SPORTS_CENTRE` has a list of facilities, and `LECTURE_HALL/EVENT_HALL` has a list of events that are printable.

> **Addition**: The `SPORTS_CENTRE` (`G`) is a newly introduced place type in Assignment 2.

Events

Some places can hold events. Every event has an ID, a date, a start time, an end time, and a score. There are three types of events:

- `LECTURE` - a lecture can have a course code and a lecturer.
- `SEMINAR` - a seminar can have a name and multiple speakers.
- `EXAM` - an exam can have a course code.

exam —no specker
seminar — more spekers
lecture —1 speaker

9 data point —> exception

A `LECTURE` can be scheduled either in the Lecture Halls or the Event Halls. `SEMINAR` and `EXAM` can be scheduled only in Event Halls. `CAFETERIA`, `LIBRARY`, `SPORTS_CENTRE` does not allow scheduling any kind of Events.

> **Addition**: Assignment 2 introduces **three event types:** `LECTURE`, `SEMINAR` and `EXAM` and Bookable Venues.

inheritance

Student

The student is represented on the map using the symbol `P`. Student is able to:

- Move in four directions: Up, Down, Left, Right.
- Visit places and attend events.
- Book places and view schedules/menus.

The student's current position, move/hits history, score, and session data must be continuously tracked.

Score History

The student will have a history of their moves, hits, events and places visited along with the scores associated. A score comprise of following details:

- map name: name of the map currently being visited.
- number of moves: number of valid moves made on the current map within a session. (Session is discussed below)
- number of hits: number of boundary hits or restricted places made on the current map within a session.
- place name: name of the place visited on the map.
- event name: name of the event visited at a place.
- date: date of the event.
- time range: time range of the event in the format start time **-** end time (Eg: 10:30 - 11:30)
- session ID: ID of the current session. (Session is discussed below)
- score: score for the place or place + event visited.

How to accumulate scores:

- Every time the student moves successfully, the move is recorded.
- If the student hits a boundary or a restricted place, the number of hits is increased.
- If the student visits a place with no events for the first time, the score associated with the place is added to the student's score.
- If the student attends an event, the score associated with the event and the place is added to the student's score.
- If the student visits the Library, Cafeteria, Sports Centre or any of the Lecture Halls or Event Halls, their details are stored, like the name of the place, name of the event, date, start and end time of the event, accumulated moves, accumulated hits and the total score.

This will be discussed in detail under the Main Menu section.

Session Manager

Every time the program is run, a new session is started, and a unique session ID is provided as input from command-line arguments.
This session ID serves as the identifier for the current gameplay session. The map layout remains fixed for the duration of the session, and all student activities are tracked against the session ID.

The program can now load score histories from previous sessions from a file. Before the program terminates, the current session history is added to the session score file. This ensures that all student activity is preserved and can be accessed in future runs. session score.txt

File Handling & Command Line Args

Command Line Arguments

The program accepts the following command line arguments in the specified order:

- `rows` (integer) – number of rows in the campus map (e.g., 7)
- `columns` (integer) - number of columns in the campus map (e.g., 6)
- `Session ID` (integer) - A unique session ID as an integer (e.g., 123)
- Campus Map file (String) - path to the campus map file. The name of the campus map must be extracted from the filename, excluding the extension. (e.g., if the file name is `data/maps/hawthorn.txt` then the map name would be `hawthorn`).
- Events file (String) - path to the events file

```
java CampusNavigatorEngine 7 6 123 data/maps/hawthorn.txt
data/events/events1.txt
```

Error Scenarios

> **Worst Case Scenario: Your program may behave incorrectly if the params are not handled explicitly. These inputs will be tested with varied data. The data files' names may change, but will always be String.**

The program can encounter error scenarios mentioned below and must handle them by printing the error messages and terminating the program.

- Less than 5 arguments present. Print `Invalid number of Command Line Arguments. Usage: java CampusNavigatorEngine <rows> <cols> <session id> <location file> <events file>` and terminate the program.
- If rows or columns are less than 4. Print `Error: Rows and columns must be at least 4 to allow proper map layout.` and terminate the program.
- If the map file or events file is not found then print `Unable to process file. Exiting program.` and terminate the program.

If valid command line arguments are present, initialise the Campus Map with rows, columns and map name.

> **Assumption**: The data types of the command line arguments will always be correct, but the data can be invalid.

> **Out of Scope:** You can assume that the rows, columns and session ID are always of type String that can be parsed into integers or long. You do not need to handle other data types such as floating-point numbers or characters. Additionally, you can assume that the

session ID is always unique. Additionally, the file names will always be String but may contain invalid file paths and to be handled accordingly.

File Handling

filehandler
map gernerator

line.split(",")
create student obj. —> read marks
arraylist —> .add(mark)

Once the Campus Map is initialised with rows and columns, it is important to read data from the files and load them into the appropriate objects of classes. The files are present in the `data` folder. This folder is provided to you along with all the files that are used in visible and hidden test cases.

Assumption: **While handling the files in code, you can assume the folder name `data` remains unchanged.**

Folder Structure

```
data/
├── maps/              # Contains map layout files
├── events/            # Contains scheduled events for places
└── session_scores.txt  # Stores past session scores
```

- **data** - There will be a data folder next to your code files and packages. This is the root folder that contains all the data files and subfolders.
- **maps** - This folder contains files that define campus maps.
- **events** - This folder contains files for events at places: the event hall and lecture hall.
- session_scores.txt: This file is initially empty with a header. Your program must keep **appending** data to it before terminating.

Campus Map File

Example: `data/maps/hawthorn.txt`

The campus map file defines the layout of the campus, including the position information for different types of places. Each line in the file represents a place on the map.

Example:

```
1,4,LECTURE_HALL,John Medley Building,8.1,no
```

Each line has data in the order:

```
row_id,col_id,place_type,place_name,score,restricted
```

- Row - row index of the place on the map
- Column - column index of the place on the map
- Place Type - one of the following types: `LECTURE_HALL`, `EVENT_HALL`, `LIBRARY`, `SPORTS_CENTRE`, and `CAFETERIA`

- Place Name - name of the place
- Place Score - a score associated with the place. For the Cafeteria, this score is negative, and for all other places, this is positive. **This is expected to be a double.**
- Restricted - "yes" or "no" value indicating whether the place is restricted boolean

Total data fields: 6 (separated by commas)

Events File

Example: `data/events/events1.txt`

The events file contains information about various events happening at some places on the campus. Each line in the file represents an event associated with a specific place on the campus.

Example:

```
1,4,LECTURE,2023-10-15,10:00,12:00,5.0,Data Structures,Dr. Smith
```

Each line has data in the order:

```
row_id,col_id,event_type,date,start_time,end_time,score,name,speaker
```

- Row - row index of the place on the map
- Column - column index of the place on the map
- Event Type - one of the following types: `LECTURE`, `SEMINAR`, `EXAM`
- Date - event date in the format YYYY-MM-DD
- Start Time - start time of the event in the format HH:mm (24-hour format)
- End Time - end time of the event in the format HH:mm (24-hour format)
- Score - a positive score associated with the event. **This is expected to be a double.**
- Event Name - name of the event
- Speakers - names of the speakers of the event, separated by '#' if multiple speakers are available for an event. For example, "Dr. Smith#Prof. Johnson"

Total data fields: 9 (separated by commas)

When you quit the program, all unattended events on the map should be written back to this file, and all attended events should be deleted.

Session Scores File

Example: `data/session_scores.txt`

This file name is not provided in the command line arguments. You can safely assume that the folder/filename will never change for session scores. The session scores will always be stored in session_scores.txt and will be present in the assets folder. The session scores file contains the session scores for different sessions identified by their unique session IDs. Each line in the file represents a session and its associated score. Example:

```
hawthorn,123,David Wood Lecture Theatre,Modern History,2024-05-14,09:00-10:30,4,0,25.70
```

Each line has data in the order:

```
map_name,session_id,place_name,event_name,date,time_range,moves,hits,scores
```

- Map Name - name of the map
- Session ID - unique identifier for the session
- Place Name - name of the place visited
- Event Name - Name of the event attended (can be empty for some places and represented by –)
- date - date of the event attended (can be empty for some places and represented by –)
- Time Range - time range of the event attended in the format start time-end time (can be empty for some places and represented by –)
- moves - Total number of moves made when visiting the place or attending the event
- hits - Total number of hits (boundaries or restricted places) when visiting the place or attending the event
- score - The score accumulated when visiting the place or attending the event

Total data fields: 9 (separated by commas)

When you quit the program, the current session score must be added/appended to this file in the order of session IDs.

Note that

1. All files have a header describing the column names, so you have to do special handling of the header vs the data.

2. row_id, col_id refer to the array indices. Array indices start from 0. So row_id 1,col_id 3 means the 2nd row, 4th column.

Tip: Sometimes file handling is buffered internally by the operating system as well. This means if you PrintWriter.print , you may end up with an empty file. This can be an intermittent issue. To avoid this, please use **`.flush()`** method.

Note that the file names may change. The folder names won't change. You must not hardcode the file names in the code except for data/session_scores.txt. -3 mark penalty will be applied for hardcoded file names.

In this assignment, the maps are not initialised by randomisation. The data is loaded from the files. Therefore, you **must not** use PositionGenerator.java from Assignment 1.

Order of reading the data

1. The program should start by reading the campus map file present in the command line arguments. The first row is the header row. The program must ignore this while reading the data. Any empty line is simply skipped. Read the next available line. Iterate through all the rows of the file, then the program should start processing each line by splitting the data by comma and processing individual data points and filling the campus map with places. You can use the `.split(",")` method to split a line by a comma.
2. Next, read the events file, follow the same steps as above when reading the file and fill the Event Hall and Lecture Hall with events.
3. Then, read the session scores file and fill the session manager with the previous session scores.

While processing individual data points, the program may face some invalid data or an invalid line format. In this case, skip the line and proceed to read the next line. The program must not be terminated in those cases. See the next section for Exception handling.

Writing the files back

1. The program must write the changes made to the session scores during the program execution back to the session_scores.txt file.
2. The program must write the remaining unattended events data back to the events file. The order of the events data at a specific location is determined by the order of input, and the order of events at different places does not matter. ??

While writing the data back, the header must be reserved as is. The program must update the session scores and events file only once at the end of the program when the user quits from the main menu (See Option 6 in Main Menu slides) or when all places and events are visited and the program terminates automatically.

IMPORTANT: Do not forget to use printWriter.flush() methods even though it is not a BufferedWriter. Sometimes files in EdStems are not written back.

As you write into the files, your files are modified by the program. While running all the test cases, subsequent test cases work on the modified file data. So if you face any error, recheck the file data saved.

NOTE: Windows vs Linux file systems handle folder paths differently. You must comply with what EdStem follows, i.e. Linux-style paths.

Exception Handling with Files

Exceptions during File Handling

Several exceptions may happen in the program.

> **Worst Case Scenario: Your program may behave incorrectly if not handled explicitly. These kinds of inputs will be tested.**

1. **FileNotFoundException or IOException** - The file paths that are provided to the program are not present in the directory. Or are unavailable for File read/write operations. In this case, Java raises FileNotFoundException or IOException. Your program should print the error message and terminate the program. Generally, these would never happen while writing the file, but only while reading the file. But the program must handle both cases in the code.

`Unable to process file. Exiting program.`

custom exception — new class

2. **InvalidLineException -** Every file expects a minimum number of fixed data points. When there are fewer than the expected number of data points present in a line, the program should raise an InvalidLineException, skip reading the line, print an error message including the line number and move on to the next line.

`Invalid line for maps. Skipping this line <line number> from the maps file.`

`Invalid line for events. Skipping this line <line number> from the events file.`

`Invalid line for session score. Skipping this line <line number> from the session score file.`

Line numbers must start from 1(and not 0). The header line is counted a Line number 1, the first data line is at line number 2. For example: `Invalid line for events. Skipping this line 3 from the events file.`

3. **InvalidFormatException -** Some data points are expected to be in a particular format. If they are not in the correct format, the program must print an appropriate error message, including the line number, skip the line and move on to the next line.

> **Note** that if one line has more than one error, only the first error encountered is printed, and the program moves on to the next line.

The order of processing the datapoints and respective errors is as follows:
For campus map file:

- If the row or column is not within the map boundary or on the map boundary, print:

    - `Invalid row format. Skipping this line <line number> from the maps file.`
    -
    - `Invalid column format. Skipping this line <line number> from the maps file.`
- If the score is not a valid data type or if the score for the cafeteria is not negative, print:
    - `Invalid score format. Skipping this line <line number> from the maps file.`
- If location type is invalid and not one of the listed Places - `CAFETERIA`, `LIBRARY`, `LECTURE_HALL`, `EVENT_HALL`, `SPORTS_CENTRE`, print:
    - `Invalid place type format. Skipping this line <line number> from the maps file.`

For event file:

- If the row or column is not within the map boundary or on the map boundary, print:
    - `Invalid row format. Skipping this line <line number> from the events file.`
    - `Invalid column format. Skipping this line <line number> from the events file.`
- If the date is not in the format YYYY-MM-DD, print the following error. Note, we are not checking the actual value of day/month/year here. Consider that they are always valid.
    - `Invalid date format. Skipping this line <line number> from the events file.`
- If start or end time is not in the format HH:mm or has an incorrect time like 25:60, print:
    - `Invalid start time format. Skipping this line <line number> from the events file.`
    - `Invalid end time format. Skipping this line <line number> from the events file.`
- If the score is not a valid double type or the score for `CAFETERIA` is not negative, print:
    - `Invalid score format. Skipping this line <line number> from the events file.`
- If event type is invalid, i.e. not one of `LECTURE`, `EXAM` or `SEMINAR`, print:
    - `Invalid event type format. Skipping this line <line number> from the events file.`

For the session scores file:

- You can assume that the session scores file is always in the correct format. You do not need to handle any exceptions for these because the session file is maintained by the program and is always expected to have correct data.

Assumption: Since we are reading a comma-separated file, none of the string data like description, will have a comma as data. A comma is only used to represent a data delimiter and not the data itself.

4. **DataNotFoundException -** If any of the required data points are missing or empty, the program must raise a DataNotFoundException, print an error message including the line number as follows, skip the line and move on to the next line. The order of processing the data points and respective errors is as follows:

For campus map file:

```
Place type cannot be empty. Skipping this line <line number> from the maps file.
Place name cannot be empty. Skipping this line <line number> from the maps file.
Score cannot be empty. Skipping this line <line number> from the maps file.
Restricted field cannot be empty. Skipping this line <line number> from the maps file.
```

For event file:

```
Event type cannot be empty. Skipping this line <line number> from the events file.
Date cannot be empty. Skipping this line <line number> from the events file.
Start time cannot be empty. Skipping this line <line number> from the events file.
End time cannot be empty. Skipping this line <line number> from the events file.
Score cannot be empty. Skipping this line <line number> from the events file.
Event name cannot be empty. Skipping this line <line number> from the events file.
```

For the session scores file:

- You can assume that the session scores always have the required data, and any missing data is indicated by -. You do not need to handle this type of exception for this file.

5. **InvalidMapException -**

- If the map file is empty or only contains the header, then print `No places found in map file. Exiting program.` and terminate the program.
- If the map position (row, column) specified in the campus map file does not correspond to a valid place on the campus map (i.e., the position is already occupied), the program must print the following message and terminate the program.

```
Map position [<row>,<column>] already occupied. Exiting program.
```

- If the line from the events file does not correspond to a valid place on the campus map (i.e., the position is not of type `PLACE` or `RESTRICTED` then the program must print `No valid place at given location. Exiting program.`
- If the line from the events file corresponds to a place, but the place is not a lecture hall or event hall or is restricted, or an event other than a lecture is added to a lecture hall, the program must print `Event cannot be added to the place. Exiting program.` and terminate the program gracefully.

Worst Case Scenario: Remember that the method that throws the exception does not catch and handle it. See the marking scheme.

Exceptions during Program Execution

6. **MovementBlockedException** - If the player tries to move to a boundary, the program must raise a MovementBlockedException and print `You have hit the edge of the map.` If the player enters a restricted place, the program must raise a MovementBlockedException and print `You cannot enter that area.` Your program must catch the exception, print the message and continue the program flow.

Main Menu

Program Run

When the program is run with the command-line arguments, you must validate the command-line arguments. (This has been discussed here). If the command line arguments are validated correctly, print the welcome message.

```
Welcome to Campus Navigator.
```

After printing the welcome message, initialise the map, load the campus map from the campus map file, events from the events file and the session scores from the session scores file. If any exceptions occur during file handling, handle them as described in the Exception Handling section here. Then print the main menu.

Main Menu

> **Modifications:** This section is referred from Assignment 1 with slight modifications, as Options 2, 3, and 5 are new.

When the program is run, the user is first presented with the Main Menu, where they can choose to begin a new campus visit, print event schedules, book places, view current score history, view score history from the previous sessions or quit the program. (Please see the command statements present in `Messages.java` and use the appropriate methods.)

```
Welcome to Campus Navigator.


=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
>
```

Option 1: Visit the Campus

> **Modifications:** This section is referred from Assignment 1 with slight modifications.

If the user selects option 1 (*Visit the campus*), the student can start navigating the map. The output shows:

- **The map name**
- A grid of the campus (e.g., 7×6) with symbols
- The student's starting location is marked with `P`
- A prompt for movement using directional keys:

```
Welcome to Campus Navigator.


=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 1
Map Name: southbank
# # # # #
# P L . #
# L @ @ #
# C . @ #
# # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
>
```

Movement in the map is discussed here.

> **Note:** The text in bold represents user input here. You don't need to make the console inputs bold in your code.

Option 2: Print schedule

> **Addition**: This specification is entirely new to Assignment 2.

If the user selects option 2 (*Print schedule*), the program prompts the user to enter row and column indices separated by a comma. There are two possibilities:

- Scenario 1: Valid Input - if the input map position is valid with a printable item. (i.e., Cafeteria has a printable menu, Sports Centre has a list of facilities, Lecture Hall or Event Hall has a list of events).
- Scenario 2: Valid Input - if the input map position is invalid, restricted or not a place with a printable item.

Out of Scope: The program only expects comma-separated two integer values for row-column input. You do not need to handle string or double values as input here.

Scenario 1: Valid Input

This case has 5 possibilities depending on the place type at the input map position.

1. **`EVENT_HALL`**: If the input map position has an Event Hall, print the event schedule and return to the main menu. Additionally, if you have already booked the event hall, the program will show a message before the schedule.

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 2
Map Name: southbank
# # # # #
# P L . #
# L @ @ #
# C . @ #
# # # # #
Enter row column indices in row,col format: 2,2
You have booked this place!
Event Schedule:
--------------------------------------------------------------------
| Exam    | 2024-05-15 | 11:00-12:30 |    Plant Genetics Seminar |
| Exam    | 2024-05-24 | 10:00-11:00 |          Data Structures |
--------------------------------------------------------------------

=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
```

```
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
>
```

2. **LECTURE_HALL**: If the input map position has a Lecture Hall, print the lecture schedule and return to the main menu.

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 2
Map Name: southbank
# # # # #
# P L . #
# L @ @ #
# C . @ #
# # # # #
Enter row column indices in row,col format: 2,3
Lecture Schedule:
------------------------------------------------------------
| Lecture | 2024-05-19 | 12:00-13:30 |  Ecology and Conservation |
------------------------------------------------------------

=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
>
```

3. **SPORTS_CENTER**: If the input map position is a Sports Centre, print the list of available facilities and return to the main menu. Additionally, if you have

already booked the sports centre, the program will show a message before the schedule.

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 2
Map Name: hawthorn
# # # # # #
# S . . @ #
# X . . . #
# . . G . #
# C . . . #
# E P @ . #
# # # # # #
Enter row column indices in row,col format: 3,3
You have booked this sports centre!
Facilities:
-----------------------
| Gymnasium          |
| Swimming Pool      |
| Basketball Court   |
| Tennis Court       |
| Fitness Studio     |
-----------------------

=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
>
```

4. **CAFETERIA**: If the input map position is a Cafeteria, print the menu from the Cafeteria and return to the main menu.

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 2
Map Name: southbank
# # # # #
# P L . #
# L @ @ #
# C . @ #
# # # # #
Enter row column indices in row,col format: 3,1
Menu:
-----------------------
|    coffee |  $2.50 |
|       tea |  $2.00 |
|   sandwich |  $4.50 |
|    muffin |  $2.75 |
-----------------------

=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
>
```

5. If there are no events or lectures scheduled at the place, print a message and return to the main menu.

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
```

```
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 2
Map Name: hawthorn
# # # # # #
# S . . @ #
# X . . . #
# . . G . #
# C . . . #
# E P @ . #
# # # # # #
Enter row column indices in row,col format: 5,1
Nothing happening here.

=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
>
```

Scenario 2: Invalid Input

In this case, print an error message and prompt the user to enter the row and column indices again.

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 2
Map Name: southbank
# # # # #
# P L . #
```

```
# L @ @ #
# C . @ #
# # # # #
Enter row column indices in row,col format: 1,2
This place does not have a schedule. Select a cafeteria, sport centre,
lecture hall or event hall.
Enter row column indices in row,col format: 1,3
This place does not have a schedule. Select a cafeteria, sport centre,
lecture hall or event hall.
Enter row column indices in row,col format:
```

Out of Scope Scenario: The program does not expect to handle two things, and these cases will not be tested.

1. If all the places are booked, and the user selects option 2, the program will still show a prompt and no error message is printed, as this is not handled in the code.
2. Once the user selects option 2, and the user enters an invalid row, col, it will prompt the user again to enter row, col until a valid place is found. There is no option to quit from these recurring prompts.

Out of Scope: The program only expects comma-separated two integer values for row, column input. You do not need to handle string or double values as input here.

Option 3: Book a Place

Addition: This specification is entirely new to Assignment 2.

If the user selects option 3 (*Book a place*), the program prompts the user to enter row and column indices separated by a comma. There are two possibilities:

- Scenario 1: Invalid Input - if the input map position is invalid, a restricted place, already booked or not a place that can be booked (i.e., not a Sports Centre or Event Hall).
- Scenario 2: Valid Input - if the input map position is valid.

Out of Scope: Same as the above two scenarios.Recurring prompts do not handle the case when no further places can be booked; they keep prompting until a place is booked. This may lead to an infinite loop, but these scenarios won't be tested.

Scenario 1: Invalid Input

In this case, print an error message and prompt the user to enter the row and column indices again. See the sample output below.

```
Welcome to Campus Navigator.


=== Campus Navigator ===
```

```
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 3
Map Name: southbank
# # # # #
# P L . #
# L @ @ #
# C . @ #
# # # # #
Enter row column indices in row,col format: 1,3
Place cannot be booked. Select a sports centre or event hall.
Enter row column indices in row,col format: 2,2
This place is already booked. Select another sports centre or event hall.
Enter row column indices in row,col format:
```

Scenario 2: Valid Input

In this case, the program prompts the user to confirm the booking of the place.

- If the user wishes to cancel and inputs `N`, the program returns to the main menu.
- If the user wishes to book the place and inputs `Y`, print a message and returns to the main menu.
    - `You have successfully booked the event hall.`
    - `You have successfully booked the sports centre.`

See sample output below.

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 3
Map Name: hawthorn
# # # # # #
```

S . . @

X . . .

. . G .

C . . .

E P @ .

# # # # # #

Enter row column indices in row,col format: **5,3**

Do you want to book this place? Press Y for Yes or any other key for No.

N

=== Campus Navigator ===

1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.

> 3

Map Name: hawthorn

# # # # # #

S . . @

X . . .

. . G .

C . . .

E P @ .

# # # # # #

Enter row column indices in row,col format: **5,3**

Do you want to book this place? Press Y for Yes or any other key for No.

Y

You have successfully booked the event hall.

=== Campus Navigator ===

1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.

```
> 3
Session ID: 11
Map Name: hawthorn
# # # # # #
# S . . @ #
# X . . . #
# . . G . #
# C . . . #
# E P @ . #
# # # # # #
Enter row column indices in row,col format: 3,3
Do you want to book this place? Press Y for Yes or any other key for No.
Y
You have successfully booked the sports centre.

=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
>
```

Option 4: View Score History

> **Modifications:** This section is referred from Assignment 1 with slight modifications.

Each time the student attends an event or visits a place, the system logs the score. Score object is defined here

Note that Places like the Cafeteria, Library and Sports Centre do not have any events. In this case, the Event Name, Date, and Time Range are noted with a –.

These records are stored in the Student's Score History and can be viewed anytime via Option 4 – *View score history*. Note that Option 4 only shows scores from the current session.

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
```

```
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 4
| Map Name       | Session Id     | Place Name                    | Event Name
| Date       | Time        | Moves  | Hits    | Score   |
|================|================|================================|======
========================|============|==============|========|=========|==
=======|
| southbank      | 6              | Baillieu Library              | -
| -          | -           | 1      | 0       |    8.20 |
| southbank      | 6              | David Wood Lecture Theatre    |
Environmental Chemistry       | 2024-05-17 | 10:30-11:30  | 4      | 0       |
15.30 |

=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
>
```

If the user selects Option 4 before any places/events have been visited (but the student moved on the map/hit the boundary), then no scores will be generated. In that case, show the message `No scores yet.` and print the main menu again.

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 4
No scores yet.
```

Option 5: View Score History from the Previous Session

Addition: This specification is entirely new to Assignment 2.

If the user selects option 5 (*View score history from the previous session*) from the Main Menu, the program prompts the user to enter a session ID out of the available session IDs or select an A to print scores from all the sessions.

Out of Scope Scenario: Acceptable inputs are 'A' or any integers. Doubles or other types are not tested.

Sample output:

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 5
Available Session IDs:
 -> 6
 -> 10
 -> 13
 -> 102
 -> 112
Enter session ID to view score history or A to print score of all sessions:
```

Note that the session IDs should be sorted in ascending order.

The student can now enter a session ID. If the session ID is invalid, print `No scores found for Session ID: <Session ID>` and return to the main menu.

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 5
Available Session IDs:
 -> 6
 -> 10
 -> 13
 -> 102
```

```
 -> 112
Enter session ID to view score history or A to print score of all sessions:
2
No scores found for session ID: 2

=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
>
```

For a valid session ID, print the score history for that session ID in the same format as Option 4 and return to the main menu.

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 5
Available Session IDs:
 -> 6
 -> 10
 -> 13
 -> 102
 -> 112
Enter session ID to view score history or A to print score of all sessions:
13
| Map Name       | Session Id     | Place Name                  | Event Name
| Date      | Time        | Moves  | Hits    | Score   |
|================|================|=============================|======
==========================|============|==============|========|=========|==
=======|
| southbank      | 13             | Giblin Eunson Library       | -
| -         | -          | 1      | 0       |    3.50 |
```

```
| southbank      | 13            | Campus Canteen              | -
| -         | -          | 2     | 0      |   -2.90 |

=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
>
```

In case the user selects A, print the scores from all the sessions. Note that the history is printed in

- sorted by session id in ascending order
- for the same session ID, the order is as per the visit order in the map.

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> Available Session IDs:
 -> 6
 -> 13
 -> 27
Enter session ID to view score history or A to print score of all sessions:
A
| Map Name       | Session Id     | Place Name                  | Event Name
| Date      | Time        | Moves  | Hits    | Score   |
|================|================|=============================|======
=========================|============|==============|========|=========|==
=======|
| southbank      | 6             | Baillieu Library            | -
| -         | -          | 1     | 0      |   8.20 |
| southbank      | 6             | David Wood Lecture Theatre    |
Environmental Chemistry       | 2024-05-17 | 10:30-11:30  | 4     | 0      |
15.30 |
```

```
| southbank      | 13            | Giblin Eunson Library         | -
| -         | -          | 1     | 0      |   3.50 |
| southbank      | 13            | Campus Canteen               | -
| -         | -          | 2     | 0      |  -2.90 |
| hawthorn       | 27            | John Medley Building          | Discrete
Mathematics          | 2024-05-15 | 11:00-12:30  | 3     | 2      |   15.30 |

=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
>
```

If no sessions are available in the session scores file, print `No session scores available.` and return to the main menu.

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 5
No session scores available.

=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
>
```

Note that we only update the sessions_scores.txt while terminating the program. And the file is initially empty. This means when the program is run first time, and if the user selects option 5, the program will always output `No`

`session scores available.` as the session history is not saved yet in the file. During the first run, if the user moves on the map, option 4 may show the score history (within the current session), but option 5 will never show score history.

Option 6: Abandon Session and Exit

> **Modifications:** This section is referred from Assignment 1 with slight modifications.

Once the user is in the main menu, they can exit the program at any time by selecting option 6. This gracefully ends the session and displays a farewell message.

Sample output:

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 6
Session abandoned.
Goodbye for now. Visit Again.
```

The program terminates cleanly.

> **Warning:** Do not use System.exit() as it is not a graceful exit.

Invalid Command

> **Carried Forward:** This section is referred as is from Assignment 1

If the user enters an option that is not listed in the menu (such as 7), the program should detect the invalid input, show an error message and prompt the user again with the main menu.

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 7
Invalid choice. Please select a valid option.
```

```
=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
>
```

Out of Scope: The program only expects an integer value for the main menu. You do not need to handle string or double values as input at this point in time for the main menu, but you must handle them for invalid integers.

Movement SubMenu

Navigating the Map

Carried Forward: This section is referred as is from Assignment 1

Once the user selects to visit the campus (Option 1) from the main menu, a movement submenu will be printed so that the student can navigate the campus map by selecting a direction (`U`, `D`, `L`, `R`), or can enter `Q` to return to the main menu as previously mentioned here.

Visiting Places

When you are navigating the map, you will come across different places. These places are discussed in detail here.
Upon visiting the place, you would need to update the ScoreHistory of the student with

- The current number of hits and moves.
- Details of the place.
- Update the current score of the student based on the score of the place. If the place is visited multiple times, then the score is added multiple times.
- If the place is a `LECTURE_HALL` or `EVENT_HALL` then record
    - the details of the event - event name, date, time range
    - add the score of the event and the score of the place to the student's score.

We will now discuss and show outputs for different use cases.

Boundary and Restricted Areas:

In *Campus Navigator*, some places cannot be entered.

Boundary (`#`)

Modifications: This section is referred from Assignment 1 with slight modifications.

- Shown on the outer rim of the grid.
- You cannot move beyond the map limits.
- If you try to move outside the map (e.g., up from the top row), the program raise a MovementBlockedException and print `You have hit the edge of the map.`

- Your position does **not** change, and a **hit** is recorded.
- The number of moves is not increased as it is not a valid move.

When attempted, the system displays:

```
Map Name: hawthorn
# # # # # #
# P . . @ #
# X . @ X #
# @ . G C #
# X E . L #
# @ @ @ X #
# # # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
> U
You have hit the edge of the map.
Map Name: hawthorn
# # # # # #
# P . . @ #
# X . @ X #
# @ . G C #
# X E . L #
# @ @ @ X #
# # # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
>
```

Restricted Area (x)

Modifications: This section is referred from Assignment 1 with slight modifications.

- Shown inside the map as restricted zones (e.g., private or secure areas).
- If you try to move into a restricted area, the program raises a MovementBlockedException and prints `You cannot enter that area.`
- Your position does **not** change, and a **hit** is recorded.
- The number of moves is not increased as it is not a valid move.

When attempted, the system displays:

```
Map Name: hawthorn
# # # # # #
# P . . @ #
# X . @ X #
# @ . G C #
# X E . L #
# @ @ @ X #
# # # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
> D
You cannot enter that area.
Map Name: hawthorn
# # # # # #
# P . . @ #
# X . @ X #
# @ . G C #
# X E . L #
# @ @ @ X #
# # # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
>
```

Cafeteria (C)

Carried Forward: This section is referred as is from Assignment 1

- Displays message: *"You can eat here if you are hungry."*
- Increase the number of moves.
- Add the score of the place to the student's score history.
- No event list or input required.
- If you revisit this place, add the score of the place.

```
Map Name: hawthorn
# # # # # #
# S . . @ #
# X . @ X #
# @ . P C #
# X E . L #
# @ @ @ X #
# # # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
> R
You can eat here if you are hungry.
Map Name: hawthorn
# # # # # #
# S . . @ #
# X . @ X #
# @ . G P #
# X E . L #
# @ @ @ X #
# # # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
```

```
>
```

Library (L)

> **Modifications:** This section is referred from Assignment 1 with slight modifications.

- Displays message: *"You can study here."*
- Increase the number of moves.
- Add the score of the place to the student's score history.
- No event list or input required.
- If you revisit this place, add the score of the place.

```
Map Name: hawthorn
# # # # # #
# S . . @ #
# X . @ X #
# @ . G P #
# X E . L #
# @ @ @ X #
# # # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
> D
You can study here.
Map Name: hawthorn
# # # # # #
# S . . @ #
# X . @ X #
# @ . G C #
# X E . P #
# @ @ @ X #
# # # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
```

```
R - Right.
Q - Quit to main menu.
>
```

Note that when you moved away from the cafeteria to the Library, the map still shows C at the previous location. The places should be fixed on the map.

Sports Centre (G)

Addition: This specification is entirely new to Assignment 2.

- Displays message: "*You can exercise here to keep fit.*"
- Increase the number of moves.
- Add the score of the place to the student's score history.
- No event list or input required.
- If you revisit this place, add the score of the place.

```
Map Name: hawthorn
# # # # # #
# S . . @ #
# X . @ X #
# @ . G P #
# X E . L #
# @ @ @ X #
# # # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
> L
You can exercise here to keep fit.
Map Name: hawthorn
# # # # # #
# S . . @ #
# X . @ X #
# @ . P C #
# X E . L #
# @ @ @ X #
# # # # # #
```

```
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
>
```

Note that when you moved away from the cafeteria to the Library, the map still shows `C` at the previous location. The same is applicable for the Sports centre and other places as well. The places should be fixed on the map.

Lecture Hall or Event Hall

Modifications: This section is referred from Assignment 1 with slight modifications.

With Events (@)

When the user visits a place with an event, the program must display the list of events to the user for selection. The user must select at least one event to visit. All other events are optional, and the user will be prompted to input their choice. See the sample output below.

```
Map Name: southbank
# # # # #
# S L . #
# L E @ #
# C P @ #
# # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
> R
Select a lecture id to visit:
Lecture: 1, Course: Modern History Lecturer: Dr. Rachel Lee  Date: 2024-05-14
Time: 09:00 to 10:30  Score: 8.70
Lecture: 2, Course: Landscape Design Lecturer: Dr. Olivia White  Date: 2024-
05-18  Time: 15:00 to 16:30  Score: 7.90
> 1
```

```
You attended event Lecture : "Modern History" and earned 8.70 points.
Do you want to visit other events? Press Y for Yes or any other key for No.
N
Map Name: southbank
# # # # #
# S L . #
# L E @ #
# C . P #
# # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
>
```

If the user wishes to attend more than one event at the same place and inputs `Y` then show the updated list of events again. Note that since the user has already attended Event 1, only Event 2 is available in the list. Once the other event is also visited, then the program displays `No more events here.`

```
Map Name: southbank
# # # # #
# S L . #
# L E @ #
# C P @ #
# # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
> R
Select a lecture id to visit:
Lecture: 2, Course: Landscape Design Lecturer: Dr. Olivia White  Date: 2024-
05-18  Time: 15:00 to 16:30  Score: 7.90
> 2
You attended event Lecture: "Landscape Design" and earned 7.90 points.
```

```
No more events here.
Map Name: southbank
# # # # #
# S L . #
# L E @ #
# C P @ #
# # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
>
```

Once all the events are visited, the user exits the event list display and is shown the map once again. Note that the position of the student is still at the same location. Once you move away from the location, the map should now show `E` there to signify that this is a place where events were held, but there are no more available events.

If you select an invalid event number, an error message will be shown, and the user will be asked to enter it again.

```
Map Name: southbank
# # # # #
# S L . #
# L E @ #
# C P @ #
# # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
> R
Select a lecture id to visit:
Lecture: 2, Course: Landscape Design Lecturer: Dr. Olivia White  Date: 2024-
05-18  Time: 15:00 to 16:30  Score: 7.90
> 5
```

```
Incorrect event id. Please try again.
Select a lecture id to visit:
Lecture: 2, Course: Landscape Design Lecturer: Dr. Olivia White  Date: 2024-05-18  Time: 15:00 to 16:30  Score: 7.90
>
```

Note that if the student attends different kind of events you should change the event type in the message

```
You attended event Lecture: "Landscape Design" and earned 7.90 points.
You attended event Exam: "Programming" and earned 7.90 points.
You attended event Seminar: "Avoid AI Plagiarism" and earned 7.90 points.
```

Without Events (E)

If you visit a place where all events have already been visited (symbol `E`), you must show the message `Nothing happening here`. The number of moves is still incremented. But the score of the place is not added, nor is the score history updated.

```
Map Name: southbank
# # # # #
# S L . #
# L E @ #
# C P @ #
# # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
> U
Nothing happening here.
Map Name: southbank
# # # # #
# S L . #
# L E @ #
# C P @ #
# # # # #
Enter direction:
```

```
U - Up.
D - Down.
L - Left.
R - Right.
Q - Quit to main menu.
>
```

Session Ends Automatically

Modifications: This section is referred from Assignment 1 with slight modifications.

If all the places are visited at least once and all the Events are visited as well, then the session will automatically be over. The user will be sent to the main menu.

- If the user wants to resume the session, it will show `No session to resume.`
- If the user wants to visit the campus, a new map will be initialised.

The output to be shown before moving back to the main menu:

```
Welcome to Campus Navigator.


=== Campus Navigator ===
1. Visit the campus.
2. Print schedule.
3. Book a place.
4. View score history.
5. View score history from the previous session.
6. Abandon the session and exit.
> 1
Map Name: southbank
# # # # #
# P L . #
# L . @ #
# C . @ #
# # # # #
Enter direction:
U - Up.
D - Down.
L - Left.
R - Right.
```

Q - Quit to main menu.

> **D**

You can study here.

Map Name: southbank

# # # # #

S L .

P . @

C . @

# # # # #

Enter direction:

U - Up.

D - Down.

L - Left.

R - Right.

Q - Quit to main menu.

> **D**

You can eat here if you are hungry.

Map Name: southbank

# # # # #

S L .

L . @

P . @

# # # # #

Enter direction:

U - Up.

D - Down.

L - Left.

R - Right.

Q - Quit to main menu.

> **R**

Map Name: southbank

# # # # #

S L .

L . @

C P @

# # # # #

Enter direction:

U - Up.

D - Down.

L - Left.

R - Right.

Q - Quit to main menu.

> **R**

Select a lecture id to visit:

Lecture: 1, Course: Modern History Lecturer: Dr. Rachel Lee Date: 2024-05-14 Time: 09:00 to 10:30 Score: 8.70

Lecture: 2, Course: Landscape Design Lecturer: Dr. Olivia White Date: 2024-05-18 Time: 15:00 to 16:30 Score: 7.90

> **1**

You attended event Lecture : "Modern History" and earned 8.70 points.

Do you want to visit other events? Press Y for Yes or any other key for No.

Y

Select a lecture id to visit:

Lecture: 2, Course: Landscape Design Lecturer: Dr. Olivia White Date: 2024-05-18 Time: 15:00 to 16:30 Score: 7.90

> **2**

You attended event Lecture : "Landscape Design" and earned 7.90 points.

No more events here.

Map Name: southbank

# # # # #

S L .

L . @

C . P

# # # # #

Enter direction:

U - Up.

D - Down.

L - Left.

R - Right.

Q - Quit to main menu.

> **U**

Select a lecture id to visit:

Lecture: 1, Course: Ecology and Conservation Lecturer: Dr. Henry Green Date: 2024-05-19 Time: 12:00 to 13:30 Score: 8.70

> **1**

You attended event Lecture : "Ecology and Conservation" and earned 8.70 points.

No more events here.

Map Name: southbank

# # # # #

S L .

L . P

C . E

# # # # #

Enter direction:

U - Up.

D - Down.

L - Left.

R - Right.

Q - Quit to main menu.

> **U**

Map Name: southbank

# # # # #

S L P

L . E

C . E

# # # # #

Enter direction:

U - Up.

D - Down.

L - Left.

R - Right.

Q - Quit to main menu.

> **L**

You can study here.

No more places left to visit on campus. Please visit another time.

Goodbye for now. Visit Again.

Guidance: Object Oriented Programming

Quick Tips

- Now that you know about UML, perhaps start designing your solutions by creating a UML diagram first. Think about what classes need to be created. Some of them are provided to you as a scaffold. You can create other classes as well.
- The second step is to associate appropriate data with appropriate classes. Create the data fields as instance variables in those classes.
- The third step would be to create the exceptions.
- The next step is file handling. **This will take the maximum amount of time.**
- The next step is to create a structure for running the menu options with the different Control Flows you have learned.
- Exceptions thrown in a method are not handled in the same method. You should handle them with try-catch block in some other methods where you are invoking the method that is causing the exception. For example, when you use Integer.parseInt and the parseInt method throws a NumberFormatException, you handle it in the method where you are using the Integer.parseInt.
- Create packages to logically group entities and interfaces, and exceptions as well.
- Think about enums.

Interfaces & Inheritance

- Implement the inheritance between related entities. **You must have at least one inheritance hierarchy.**
- Think about what the common operations are for different entities. Interfaces are implemented where unrelated/related entities can have similar behaviour but different implementations. **You must create and implement at least one interface.**

File Parsing

scanner — fis, fr(filereader), file
bufferreader — only fr

fileinputstream
input.hasNextLine()
finally block to close. (fis , scanner input == null outside of block). if != null in finally block
pw —fos, fw, file
read()
write()
output.flush() . complete writing all string

- Use a simple Scanner and PrintWriter instead of over-complicating the reading/writing. Try to minimise duplicate code associated with reading and writing.
- **Do not forget to use flush() for writing.**
- When you read the files, you can read the entire line and then split the data on commas. This shall provide you with different data points in an array. Check the split method in the String class here. You can use this method to perform a split on a comma.

JavaDoc

Your code should be annotated using javadoc comments. We will generate the Javadoc for your code. **You do not need to submit it.** You can run javadoc on your machine (up to any levels of nested packages) by running the command

```
$ javadoc -d docs/ *.java
$ javadoc -d docs/ **/*.java
$ javadoc -d docs/ **/**/*.java
.
.
```

UML

The UML Diagram is discussed in the Week 9 Lecture in detail. The UML Diagram present in the code challenge is just a starter code, which can act as a scaffold or guide you on what are mandatory classes/methods are in the assignment.

- There are some classes present, but they are not grouped in packages. You must create packages and add the classes to some packages.
- There are some sample packages created that you can use.
- Some classes include fields and methods. The three dots indicate that additional instance variables and methods can be added as needed.
- The italicised class means it is an abstract class also denoted by <<abstract>>, and the italicised method is abstract. You are free to change the definition of the abstract method by adding parameters if you see that it fit, but the **Event class must be abstract**.

The Intended Learning Outcomes for the final Project are mentioned below -

- **Control Flows** - use branching and looping to navigate the special use cases in specifications.
- **Classes** - identify the entities and encapsulate their data and actions together in a class.
- **Arrays** - to use 1D and 2D arrays and perform associated operations
- **Packages** - identify and group logical entities (classes) together
- **Javadoc** - generate javadocs from comments used in classes
- **UML** - generate a UML Diagram for the classes
- **Inheritance** - implement polymorphism correctly
- **Interfaces** - implement some operations via interfaces
- **File Handling** - reading and writing data from/to different files. **Don't use java.nio.Files methods.**
- **Exception Handling** - handle exceptions gracefully and appropriately
- **Java Collections Framework** - using ArrayLists or HashMaps to store and process data efficiently.
- **Generics** - Generics are optional for this assignment. **Implementing Generics is not the same as using generics, such as ArrayList<>. No penalty or extra marks will be given for using Generics.**

> A warning to those who have previous programming experience in other programming languages (like C or Python) that follow a procedural programming paradigm: Be careful to develop your program using object-oriented principles, as programming and structuring your code in a way that is not object-oriented is an easy way to lose marks for structure in the assignments.

We will also assess your code for its good structure and style.
Use methods when appropriate to simplify your code and **avoid duplicate code.**
Use correct Java naming conventions for class names, variable names, and method names and put them in a well-organised way in your code, i.e. it should be readable as well. Look at the conventions provided by Oracle **here.**
The code structure in a file is very important and **improves readability**.
Look at the code organisation conventions provided by Oracle **here.**
Make sure the **names** you choose are **meaningful** to improve the readability of your code.
Ensure to add meaningful comments in between your code and **Javadoc comments** for classes and methods.

We will provide a marking scheme to help guide you in the development of your program. Also we will guide you how to model your program. But using correct syntaxes and conventions can be learnt here.

Marking Scheme 📝

> **Warning! You must make sure your code runs on edstem. If your code does not compile on edstem, 0 marks will be given.** In this assignment, we are more focussed on how you structure the program in terms of packages, classes, methods, etc. instead of the logic itself. **Read the marking scheme carefully to gain marks.**

COMP90041 Assignment 2: Marking Scheme

Student: ***[Student ID goes here]***

Marking process:

1. Code compiles (Pre-requisite)
2. Run tests
3. Verify test results
4. Inspect the code and fill in the sections below

1. Program Execution: Automated Tests on EdStem

Gain **0.5** marks for each test. Minor differences cause a loss of **0.25** marks
Total tests passed: /20
Total marks (0.5 * tests passed) : **/ 10**

```
Points awarded for this section: out of 10
```

2. Packages

Including: Code structure resembling real-world entities. Clear hierarchy using separate, well-named files for generics, abstract, and concrete classes. Use of package(s) to modularise code. Minimum 3 packages created. Enums identified and created.

```
Points awarded for this section: /1
```

3. OOP & Encapsulation

Including: Proficient use of modifiers (private, public, protected) with appropriate getter/setter methods. No privacy leaks. [1 mark]
Well-considered usage of static methods and free of redundant object passing. Low Coupling, High Cohesion. [1 mark]

```
Points awarded for this section: /2
```

4. Polymorphism

Including:
Inheritance: Proficient use of abstract classes or interfaces, and inheritance. Classes are well-designed without redundancies and with elegant overloading and overriding. [2 marks]
Overloading is shown clearly. [1 mark]
Interfaces are defined for similar method signatures amongst different classes. [1 mark]

```
Points awarded for this section: /4
```

5. Control Flow

Including: Easily traceable program flow. Loops have clear breakout conditions. Proficient use of switch and if-else statements. No more than 3 nested loops (loops mean for/while/do-while. Constants and Enums are used correctly.

No system.exit. System.exit incurs a penalty of 0.5 marks.

```
Points awarded for this section: /1
```

6. File Handling

Including: Correct read/write of files , including overwriting or appending a file wherever necessary. Take care of any file-handling exceptions that might occur in the code due to incorrect usage of files. File Reading/Writing codes are efficient and not duplicated.

Use of java.nio.Files incur a penalty of -2 marks.

```
Points awarded for this section: /2
```

7. Style

Including: Consistency around naming conventions and descriptive naming of classes, methods, and variables. The program code is well indented, spaces are sufficient, and avoidance of overly long lines of code. The code is organised correctly in the files.

```
Points awarded for this section: /1
```

8. Documentation and Javadoc

Run

```
javadoc -d doc **/*.java
```

Including: Javadoc created without errors. Following Javadoc conventions. All major classes and methods are annotated. Additional inline comments clarify complex program sections and private methods. If you have more than 1 level of folders, you must generate javadocs for all of them without any warnings/errors.

???

Two warnings excluded for Javadocs:

- Default constructor warning, where your files do not have a default constructor, and Java provides a default constructor
- Warnings for ENUM values. However, enums defined should be annotated with Javadoc comments.

```
Points awarded for this section: /2
```

9. UML

Including: Major classes and methods are listed with their correct association arrows and multiplicity values. The spatial arrangement allows for easy reading and shows a clear hierarchy between classes. The package allocation is correctly displayed. The Java Library Exception class is shown with an association.

> DO NOT GENERATE UML DIAGRAMS FROM INTELLIJ. MARKS WILL BE DEDUCTED. **0 marks provided for UML generated from IntelliJ or any other software that can automatically generate the diagrams from Java files.**

```
Points awarded for this section: /3
```

10. Exception Handling

Including: All exceptions handled, including any system-generated exceptions like FileNotFound or Arithmetic Exceptions or NumberFormatExceptions

```
Points awarded for this section: /2
```

11. Correct Data Structures/Collections

Including: correct data structures used from the Collections Interface, like ArrayList, HashMaps, etc, wherever required.

```
Points awarded for this section: /2
```

Total Marks for the Assignment 2: / 25

Generics are not part of the rubrics in this Assignment.

If you have any questions regarding your mark, please contact the lecturers.