Assessment brief

Time 3 hr + 15 mins for upload

You have been given some minimal starter code that is currently being developed as part of a menu driven **TV Scheduling System**. The system will allow analysis of available data, about TV Programme Scheduling Data across a number of days and channels.
Not all the requirements have been implemented. It is your task to implement these and raise the coding standards of all the code.

Create a **project solution** (named **P3<your name and student id> e.g. P3**CollinsMatthew12345678**). Ensure your name and student number are placed in the Javadoc comments of all the classes / Test classes etc you create.**
Create a **package** within the project named **p3**.

2 initial files have been provided.
Add **TVScheduleManager.java** to the p3 package in the solution and add the **SampleTVData.csv** file at the project root.
Ensure your name and student number are placed in the Javadoc comments of all the classes you create.

- The Launcher has been partially completed with a selectable console driven menu and a skeleton Read Data method.

The application should run (start) from the **TVScheduleManager.java**, initially reading in the data from the **SampleTVData.csv** file and then allow the user to perform a number of menu driven operations.

Assessment Structure

Part 1 (50%): Implement the **ProgramTimeSlot** Class based on the structure of the data in the csv file and implementation details below. Only limited validation/business rules required at this stage of development.

Appropriately Unit Test the **ProgramTimeSlot** Class. No unit testing required for other classes and their methods at this time. **– light touch testing sufficient (see details in section** 1.2 Unit Testing Expectation**)**

Update the partially completed **readData(String filename)** method in **TVScheduleManager.java** to read the data from the file into a JCF List of ProgramTimeSlot objects.

Part 2 (50%): Edit and further develop **TVScheduleManager.java** to implement the missing menu driven functionality by implementing and calling the required methods from the **processMenuChoice** method.

Focus on necessary methods for the tasks outlined as menu options. No need to spend time on improving user input functionality at this stage of development, though this may be a future consideration of the project.

Do consider examples of good practice in terms of method configurability, validation, and potential reuse in both the tasks in the current specification and future requirements which may be added. The provided starter code has been configured to recover from exceptions thrown by methods.

Part 1 – Accommodation class, Testing and Data Import (50% Weighted)

1.1 Specification

Using your knowledge of OOP you should add/update code based on the following:

Analyse the data in the **SampleTVData.csv** and create a class (**ProgramTimeSlot.java**) and any other associated files/classes you deem necessary for your design.

In this instance, each column in the provided file corresponds directly with the expected attributes of the required object design, and each row represents an instance of an object.
The following allowable values/limited business rules have been agreed at this stage of development but may be updated in future as requirements evolve.

Attribute	**Allowable Values**
day	MONDAY - FRIDAY Suggested java.time.DayOfWeek enum type
startTime	Suggested java.time.LocalTime type
channel	String
name	String
category	NA, ENTERTAINMENT, NEWS, DRAMA, COMEDY

Appropriate getters/setters and constructor(s) should be provided for all attributes.

No specific validation is required at this time, especially for fields using pre-built types from the Java.time package, but assume that validation requirements could be added in future and design the class appropriately.

The implementation will also assume that the ProgramTimeSlot class will have the following public method which you will need to implement.

printDetails() : void	Simple method to print details of the current objects instance variables to the console in a formatted manner.
	
	Example of expected format
	
	`FRIDAY - 21:00 Ch: ABC1`
	`Chat Show Couch Guy (ENTERTAINMENT)`

Other methods etc. can be added as the developer deems necessary.

1.2 Unit Testing Expectation

Conduct appropriate light touch Unit Testing of the behaviour of the **ProgramTimeSlot** Class and its publicly accessible methods.

For the purposes of this assessment, it is not necessary to test any other classes/methods.

Business rules requirements have deliberately been kept minimal to reduce the overall testing expectation.

Test data for all relevant fields is expected, regardless of whether the test itself has been implemented or not.

It is not necessary to test void methods which have no impact on the state of the object. (e.g. printDetails type methods)

If fully autogenerated methods such as equals() are deemed necessary and included in your implementation it will not be expected that unit tests of these are included for the purposes of the assessment.

1.3 Read Data

The **readData** method in **TVScheduleManager.java** has been partially completed (minimal/skeleton). Finish the implementation of this method so that it will work with your **ProgramTimeSlot** class. Each line of the provided file, assuming it contains valid data, should correspond to an ProgramTimeSlot object to be read into the overall List returned by the method.

No deliberate bad data lines have been included in the file, but the read method should ideally be written so that lines which violate future business rules the business rules would be skipped and not included in the List populated by the read method. An appropriate error message printed to the console to help identify the cause of the issue in the file would be helpful.

Note also that **spreadsheet programs like Excel may sometimes display data differently from how it is actually stored in the source file (number formatting etc)**, so it may be worth also viewing in a plain text editor like notepad or eclipse's own text editor when analysing the data.

If struggling to complete the read task, it may be worth creating a separate method to create a dummy list of a small number of hard coded sample objects to allow you to proceed in the time available and work on Part 2.

This method has been written on the assumption that a previously existing static list will be populated, but it is advised that other menu driven task methods are written to expect to receive the list as an argument, rather than manipulating the static variable.

Part 2 – Launcher and Menu Options Tasks 2-7 (50% Weighted)

1) Menu option 1 will already have been achieved by completing the readData method.

2) Display the **printed details** of all shows in the schedule list.
Sorted by channel (alphabetically) and chronologically.
i.e. All shows on ABC1 in time order, displayed before shows on another channel which again would be in time order.

```
Sample output (same formatting can be used for other display tasks):
MONDAY - 00:00 Ch: ABC1
Channel Closed (NA)

MONDAY - 06:00 Ch: ABC1
Breakfast with Bill (ENTERTAINMENT)

...Etc.

TUESDAY - 13:30 Ch: ABC1
Local News Lunch (NEWS)

TUESDAY - 14:00 Ch: ABC1
Detective Doctor Guy (DRAMA)

...Etc.

MONDAY - 06:00 Ch: Premium Events
Pay Per View AM (ENTERTAINMENT)

MONDAY - 10:00 Ch: Premium Events
Its 5 O'Clock Somewhere (ENTERTAINMENT)

...Etc.

FRIDAY - 23:00 Ch: Shopping Channel
Channel Closed (NA)
```

3) Display **printed details** for all shows on **ABC1** specifically, sorted by Day then Time.

4) Display **printed details** for all shows on the **Premium Events** channel specifically, on **Friday** sorted by Time.

5) Display **printed details** for any shows, on **Thursday** where the show **title contains the word 'Lunch'** (case insensitive), sorted **alphabetically by show title**.

6) Display what is on NOW, and NEXT, on the ABC1 channel at the current time.
Assuming Java.time data types have been used the following sample code may be useful.

SampleCode

```
//create instance of time based on current system clock
LocalTime t = LocalTime.now();

//simplified time object, just hour and min field
LocalTime now = LocalTime.of(t.getHour(), t.getMinute());

System.out.println(now);



LocalDate date = LocalDate.now(); //as above, but for date

//extract day value from current date (Uses java.time.DayOfWeek enum)
System.out.println(date.getDayOfWeek());
```

Sample output:

```
Now and Next – ABC1
Now: Lunchtime News 12:30
Next: Local News Lunch 13:30
```

7) Breaking News Event – 12:45, Tuesday.
   In a **separate Thread**, insert a new show with the title "BREAKING NEWS" and the category value NEWS into the schedule list. Update all show start times after this point by 30 mins. (no need to consider cases where time would extend beyond midnight for the purposes of this time limited setting. Channel closure buffers are in place in file data already).

   Verify the effect with other earlier display tasks/undo the effect by repeating the read task.

   Sample output:

```
Breaking news, 12:45 Tue. Insert News Bulletin for ABC1. Delay all shows after News Bulletin by 30 mins
New Show Entry added successfully
Times updated by 30 mins after added show
```

Complete as much as you can within the time available balancing your efforts across both sections. When finished, compress (zip) the entire **Eclipse solution** and upload to CANVAS

Now : check the uploads to ensure you have submitted the correct files

It is your responsibility to ensure the correct files have been uploaded. Canvas Assignments have the option to check and redownload what has been submitted if necessary.

[END]