Assessment brief

Time 3 hr + 15 mins for upload

You have been given some code that is currently being developed as part of an **Online Accommodation Booking System**. The system will allow analysis of available data, about a series of different types of places to stay, prices etc in a number of cities across the world.
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. P3CollinsMatthew12345678). 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 **Launcher.java** to the p3 package in the solution and add the **Rooms.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 partially completed Read Data method.

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

Assessment Structure

Part 1 (50%): Implement the **Accommodation** 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 **Accommodation** 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 **readRoomData(String filename)** method in **Launcher.java** to read the data from the file into a JCF List of Accommodation objects.

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

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.

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 **Rooms.csv** and create a class (**Accomodation.java**) and any other associated files/classes you deem necessary for your design.

In this instance, each row in the provided file corresponds directly with the expected attributes of the required object design.
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
Name	Non empty String, at least 1 char
Type	HOTEL, HOSTEL, BNB
Stars	Whole numbers 1 to 5 inclusive
City	Non empty String, at least 1 char
Price	Non negative number
Rooms	Non negative whole number

Appropriate getters/setters should be provided for all attributes.

An attempt to set a value outside allowable ranges should give an appropriate exception with an appropriate exception message e.g. “Invalid name” etc.

The implementation will also assume that the Accommodation 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
	
	`Name: The Plaza`
	`City: New York`
	`Type: HOTEL`
	`Capacity:50 Rooms @ £450.00`
	`Rating *****`

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

1.2 Unit Testing Expectation

Conduct appropriate Unit Testing of the behaviour of the **Accommodation** Class and its publicly accessible methods.

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

Expected business rules of the fields have deliberately been kept simple to reduce the overall testing expectation.

Recognising time constraints, **it will be sufficient to demonstrate rigorous testing methodology for just the CONSTRUCTORS(S) and STARS getter/setter pair.** (as stars has the most detailed business rules) – however it should be appreciated that proper unit testing of all fields can be useful in diagnosing implementation issues.
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 **readRoomData** method in **Launcher.java** has been partially completed (minimal). Finish the implementation of this method so that it will work with your **Accommodation** class. Each line of the provided file, assuming it contains valid data, should correspond to an Accommodation object to be read into the overall List returned by the method.

Lines with data which violate the business rules should ideally be skipped and not included in the List returned 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.

Part 2 – Launcher and Menu Options Tasks 3-9 (50% Weighted)

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

2) Display the **number of places to stay** in the current list (repeatable as may change due to other menu options) – this menu option will already be completed in the provided starter file.

   Sample output:
   ```
   Number of places to stay is 100
   ```

3) Display **printed details** for all places to stay in the current list in the console. Number the entries 1-X as their details are displayed.

   Sample output:

   ```
   1)
   Name:    The Ritz-Carlton
   City:    Los Angeles
   Type:    HOTEL
   Capacity:60 Rooms @ £400.00
   Rating   *****


   2)
   Name:    The Beverly Hilton
   City:    Los Angeles
   Type:    HOTEL
   Capacity:75 Rooms @ £250.00
   Rating   ****

   ...Etc.
   ```

4) Display details of the **3 least expensive** places to stay in **Los Angeles** with a **4 star rating, ordered by price – low to high** - same output format as task 3

5) Display the number of unique city names in the current List
   Sample output (not necessarily correct number):

   ```
   Number of Cities: 20
   ```

6) Display details of the **4 most expensive BnBs** in **Dublin, ordered by price – high to low** - same output format as task 3

7) Display the **average price** of a **hotel** in **New York** correct **to 2 decimal places**.
   Sample output (not necessarily correct number):

   ```
   Average Price New York: £395.50
   ```

8) **Remove** all entries from the current list with **fewer than 10 rooms available in Paris**
   Verify by rerunning menu option 2 and reverse by rereading the file with option 1.

9) In a **separate Thread**, w**rite out to a csv file**, a simple 2 column structure of **city names** and the **average cost of a hotel there.**
The output file should be named **averageCosts.csv**

It should be **ordered alphabetically by city name**, and averages should be **formatted to 2 decimal places**.

Sample output, not necessarily correct numbers (as viewed in Excel – note number formatting obscured)

	A	B	C
1	City	Average Price	
2	Amsterdam	165	
3	Berlin	99.5	
4	Chicago	450.67	
5	Dublin	100	
6	Florence	100	

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]