COMP1161 PROJECT PART 2

Due March 27, 2023

OVERVIEW

After testing of the prototype application by Jamrock's Ministry of Transport, a decision has been made to add a user interface that allows management of the data. The application should allow distributed access to low power devices over bandwidth constrained environments, and therefore a console style interface will be made available. Several teams will work in conjunction to complete the application, and another team has already completed the main interface as well as a part of the planner management functionality. You have been contracted to complete the following deliverables as an individual contributor:

1. Completion of the bus management functionality, which includes.
    a. Writing methods to read data files
    b. Building entry and edit screens
    c. Implementing exception tolerant routines.
2. Preparing javadoc files for a generalized Bus object, as well as specialized Buses.

The starting code provided is a version of a solution to the initial project assignment.

SUBMISSION INSTRUCTIONS
Your submission will be made on OurVLE as a single compressed file with the name score_<score>_<IDNUMBER>. As an example, if your ID number is 620000001, and you earned a score of 85.5 the zip file will be named **score_85.50_620000001.zip.** The file will be generated by Tester.java after running the test cases. Tester.java is able to access information on your ID number because the starting code requires you to enter your ID number before running tests. **THE ZIP FILE SHOULD NOT BE MODIFIED BEFORE IT IS SUBMITTED. UNAUTHORIZED MODIFICATIONS (INCLUDING RENAMING) WILL RESULT IN AN AUTOMATIC 0 BEING AWARDED FOR THIS PROJECT.** Please note that the submission file contains safety features to detect unauthorized modifications, which will be validated during final assessment. Detection of unauthorized modifications will result in rejection of your submission. Scores returned by the autograder will be moderated.

Note that test cases are placed in a folder named **cases** that is in a subdirectory of the java user directory***. If you are using an IDE that uses different relative paths, you may need to move the folder "cases" to the location expected by the IDE to be the java user directory. You can extract that information by executing 'System.out.println(System.getProperty("user.dir"));' .***

BUSINESS RULES

Architecture

The primary objective of this phase of the project is to incorporate interfaces that allow data to be managed in the system. The starting code includes a Tester class that allows you to validate the completion status of tasks. Eleven (11) test cases are incorporated. Tasks 0-1 validate understanding of the interface by allowing users to load test cases that contain Planner information, then add and edit planner data. Tests 2 to 6 validate the programmer's ability to read bus data from a file, and instantiate a bus or the appropriate subclass. Tests 7-9 validate the programmer's ability to accept information to create entry and edit interfaces that accept information of a bus or the appropriate subclass. Test 10 validates the ability to use the javadoc tool to assist in system documentation.

- Completion of tasks 0-1 simply involve following instructions, and do not require manipulation of code.
- Completion of the tasks 2-6 involves completing the method **loadBuses** in class **WorkArea**. LoadBuses accepts one argument which is the name of a file that contains bus information. Each row in the file contains information on one bus. The number of items on a row prescribe the type of bus to be created. Table 1 serves as a key that is to be used to interpret the data in the bus file.
- Completion of the tasks 7-9 involve completing the methods **manageBuses, createBus** and **findBus** in class **EntryScreen,** as well as method **updateLocalData** in the **Bus class,** which is appropriately overloaded in subclasses.
- Completion of task 10 requires preparation of the Bus class and it's subclasses for documentation using javadoc, and creating the associated technical documentation.

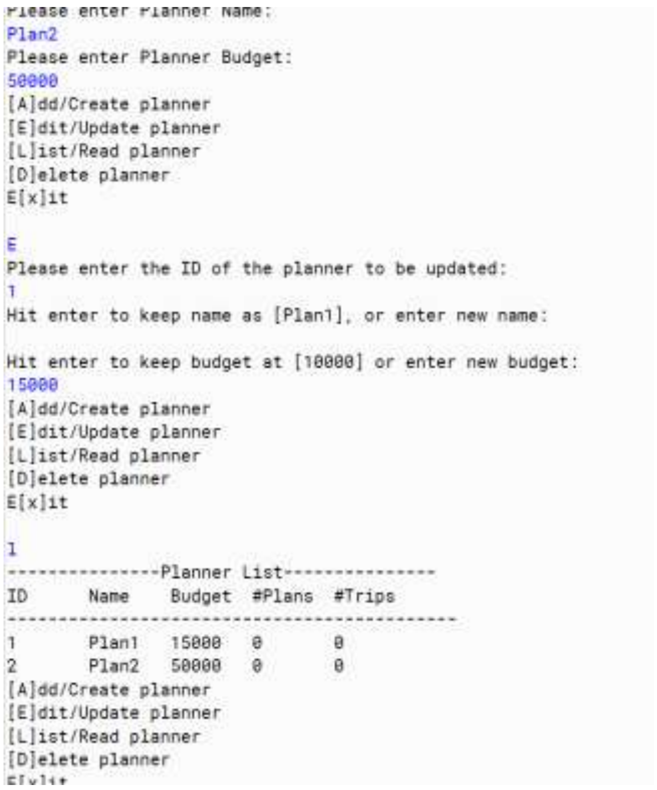
Detailed Process.

Section 1- Interface Familiarization.

In order to complete task 0, start the program, ensure that your ID number is entered, and then select the option to add an item. List the planners to verify the item has been added appropriately, then exit to the main menu and save a test case as case 0. To complete case 1, start the program, load test case 1, add an item, then edit the planner with ID 0. You can then list the active planner in the system to confirm data, and save the test case as case 1. Figure 2 shows an example of the output after the following actions :

a. Start the system
b. Load test case1
c. Add a planner with the name Plan2 and budget 50000,
d. Edit planner with ID 1 to set the budget to 15000.
e. List planners

Save the current environment data using the Save Test case option. Save as case 1.



```
Please enter Planner Name:
Plan2
Please enter Planner Budget:
50000
[A]dd/Create planner
[E]dit/Update planner
[L]ist/Read planner
[D]elete planner
E[x]it

E
Please enter the ID of the planner to be updated:
1
Hit enter to keep name as [Plan1], or enter new name:

Hit enter to keep budget at [10000] or enter new budget:
15000
[A]dd/Create planner
[E]dit/Update planner
[L]ist/Read planner
[D]elete planner
E[x]it

l
---------------Planner List---------------
ID      Name    Budget  #Plans  #Trips
--------------------------------------------
1       Plan1   15000   0       0
2       Plan2   50000   0       0
[A]dd/Create planner
[E]dit/Update planner
[L]ist/Read planner
[D]elete planner
E[x]it
```

Figure 1

Section 2- Bus Imports

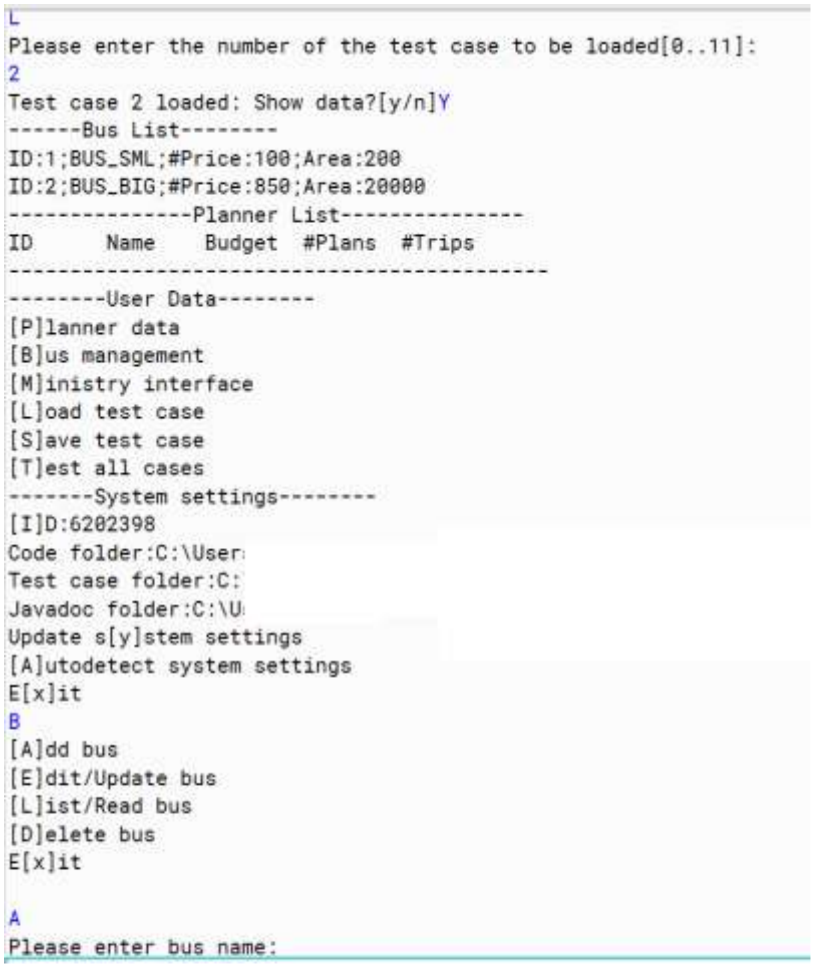
Completion of tasks 2 through 6 requires modification of several classes, including Driver and WorkArea. Driver should be updated to remove the message “(Not yet implemented)”. EntryScreen should be adjusted so that the when the option ‘B’ is selected from the main menu, the screen is updated to show a bus management menu. If the option to add a bus is selected, the screen should display options for the type of menu to be shown. The entry screens should allow the subclasses of class bus to be populated. Figure 3 provides an example of the expected display.

WorkArea is then to be adjusted to implement the method **loadBuses**. The starting code includes the skeleton of the method, which includes the method header and the name of the file to be accessed.

# cols	Bus Type	Col0	Col1	Col2	Col3	Col4	Col5	Col6
4	Generic	name	Size	Price	Level			
5	Training	name	size	Price	level	Teacher Area		
6	Sport	name	size	price	Level	Spectator Area	#Security	
7	Party	Name	size	price	Level	Spectator Area	#Security	Bar Area

Table 1. Key for data in venue file

On reading a line, the appropriate type of bus is to be created and added to the arraylist of bus. LoadBuses should return the full list of buses as specified in the file. The “[L]ist bus” option should invoke the method listBuses in the reportWriter class, passing System.out as an argument. Listing the buses should serve as a useful method to determine whether the bus information was successfully loaded. Task2 assesses whether the program is able to successfully read a generic bus. Task3 assesses whether the program is able to successfully read a training bus. Task4 assesses whether the program is able to successfully read a sport bus. Task5 assesses whether the program is able to successfully read a party bus. Task 6 assesses whether the program is able to successfully read a combination of bus types.



```
L
Please enter the number of the test case to be loaded[0..11]:
2
Test case 2 loaded: Show data?[y/n]Y
------Bus List--------
ID:1;BUS_SML;#Price:100;Area:200
ID:2;BUS_BIG;#Price:850;Area:20000
---------------Planner List---------------
ID      Name    Budget  #Plans  #Trips
--------------------------------------------
--------User Data--------
[P]lanner data
[B]us management
[M]inistry interface
[L]oad test case
[S]ave test case
[T]est all cases
-------System settings--------
[I]D:6202398
Code folder:C:\User
Test case folder:C:
Javadoc folder:C:\U
Update s[y]stem settings
[A]utodetect system settings
E[x]it
B
[A]dd bus
[E]dit/Update bus
[L]ist/Read bus
[D]elete bus
E[x]it

A
Please enter bus name:
```

Figure 2.

Section 3- Bus Entry and Update

Tasks 7-11 assess the ability of the programmer to create entry and modification screens. For task 7, the programmer is expected enter data into the system by adding the logic behind the options to create a party bus by implementing method createBus in class EntryScreen.

Task 8 assesses the programmers ability to write a screen that will modify generic bus data (by writing the code for method updateLocalData in class Bus).

Task 9 assesses the programmer's ability to write code that modifies a training bus by overriding method updateLocalData in subclasses of Bus.

Section 4.Technical Documentation

Tasks 10 assesses the programmer's ability to enter javadoc comments, and run the javadoc utility on the folder holding the code.

SPECIFIC GRADEABLE TASKS.

0. **ADD A PLANNER AND SAVE THE DATA*(2 marks)***

Perform the following actions:

a. Start the system
b. Load test case0
c. Add a promoter with the name **Plan2** and budget **50000**,
d. List promoters to validate system data
e. Return to the main menu and save the data as test number 0

1. **ADD A PLANNER, THEN EDIT A PLANNER AND SAVE THE DATA*(3 marks)***

Perform the following actions:

a. Start the system
b. Load test case 1
c. Add a promoter with the name **Plan2** and budget **50000**,
d. Edit the promoter with ID **1**, setting the budget to **15000**
e. List promoters to validate system data
f. Return to the main menu and save the data as test number 1

2. **READ DATA FROM A BUS FILE AND POPULATE AN ARRAYLIST OF BUSES*(20 marks)***

Perform the following actions:

a. Update the class Driver to remove the text "Not yet Implemented" from the Bus entry in the menu

b. Complete the method loadBuses(String bfile) that reads information bus information from a file (bfile contains the filename) . Each row in the file should trigger the creation of an instance of bus or an appropriate subclass, according to the data in table 1. All buses created should be added to an arraylist. LoadBuses should return the populated arraylist.
c. Start the system
d. Load test case 2
e. List the data to view bus entries, thereby validating accuracy

3. **READ FROM A FILE WITH TRAINING BUSES AND POPULATE AN ARRAYLIST(*5 marks)***
   Perform the following actions:
   a. Start the system
   b. Load test case 3
   c. List the data to view training bus entries, thereby validating accuracy

4. **READ FROM A FILE WITH SPORTS BUSES AND POPULATE AN ARRAYLIST(*5 marks)***
   Perform the following actions:
   a. Start the system
   b. Load test case 4
   c. List the data to view sports bus entries, thereby validating accuracy

5. **READ FROM A FILE WITH PARTY BUSES AND POPULATE AN ARRAYLIST (*5 marks)***
   Perform the following actions:
   a. Start the system
   b. Load test case 5
   c. List the data to view party bus entries, thereby validating accuracy

6. **READ FROM A FILE WITH VARIETY OF BUSES AND POPULATE AN ARRAYLIST (*5 marks)***
   Perform the following actions:
   a. Start the system
   b. Load test case 6
   c. List the data to view party bus entries, thereby validating accuracy

7. **ADD A BUS AND SAVE THE DATA*(15 marks)***

Perform the following actions:

~~a.~~ Your bus menu should have an option to add a bus. It should invoke a method called createBus in class EntryScreen, that accepts a Scanner and a Ministry, allows a user to enter information on a the bus name , size, price and comfort level, and returns the new bus. You should also add the new bus to an arraylist of buses
b. Start the system
c. Load Test case 1
d. Add a general purpose bus with name BUS_BIG, size 50000,price 0 ~~5000~~ and level 2
e. List Buses

Return to the main menu and save the data as test number 7

8. **EDIT A GENERAL PURPOSE BUS AND SAVE THE DATA*(15 marks)***

Perform the following actions:

a. Complete the code for method findBus in class EntryScreen. FindBus accepts as arguments and arraylist of Buses, and a the ID of Bus, and returns the index of the item matching the Complete the code for method updateLocalData in class Bus. UpdateLocalData provides a user interface to ensure that all instance data values for the object are populated.
b. Start the system
c. Load test case 2
d. Using the edit options, change the name of the bus with ID 1 to OTHER_BUS, and set the price to 9999
e. List Buses to validate the changes.
f. Return to the main menu and save the data as test number 8

9. **EDIT SPORTS AND PARTY BUSES AND SAVE THE DATA*(20 marks)***

Perform the following actions:

a. Override method updateLocalData for subclasses of Bus so they allow all instance data items to be entered.
b. Start the system
c. Load test case 6
d. Using the edit options, change the name of the bus with ID 12 to SPT_BIG, and set the competitor area to 5000.
e. List Buses
f. Using the edit options, change the name of the bus with ID 5 to PRTY_SML, set the price to 100 ~~1000~~, and set the bar area to 2 ~~1000~~
g. List Buses to validate the data in the system.
h. Return to the main menu and save the data as test number 9

10. **PREPARING AND RUNNING JAVADOC*(5 marks)***

Perform the following actions:

a. Edit the files that define a bus and it's subclasses and enter javadoc comments that describe the following functions
    i. The constructors
    ii. updateLocalData
b. Execute the javadoc utility on the source

FINALLY, START THE APPLICATION, AND SELECT 'T' TO IMMEDIATELY RUN TESTS. COLLECT THE RESULTING ZIP FILE AND SUBMIT. DO NOT MODIFY THE GENERATED FILE IN ANY WAY BEFORE SUBMITTING.

YOU ARE ABLE TO SUBMIT AS MANY TIMES AS YOU WISH BEFORE THE PROJECT DEADLINE.