# PROJECT ASSIGNMENT 2

**Issue Date : 29.03.2024 - Friday**
**Due Date : 21.04.2024 - Sunday (23:00)**
**Advisor :** [redacted]
**Programming Language :** Java 8u401 (1.8.0_401)


## 1 Introduction

Bus is one of the most important medium of transport as it is the optimal solution in the trade-off between number of passengers and number of available destinations. Moreover, managing are the core of transportation vehicle management as companies want to ensure about ownership of the seat properly. Making this system online both brings consistency and easiness at the management. In this project, you are expected to create a system that handles seats of the buses with giving options as creating/cancelling a bus voyage, selling and refunding available seats, and taking a view to voyages.

## 2 Definition of the System

There will be three types of buses, one of them will be standard bus which has seat plan as 2+2, the other one will be premium bus which has seat plan as 1+2 (it is 2+1 indeed but as the premium seats are at the left side of the bus, it is named as 1+2 for better understanding of yours), and the last one will be the minibus which has seat plan as 2 (it only has two standard seats in a row) and its tickets are not refundable. Singular seats will be considered as premium seats whereas the double seats will be considered as two standard seats. Price of the premium seats may differ from the standard seats which can be defined at the initialization step of the bus. Seats will be numbered according to their position, the topmost and leftmost seat will be numbered as 1 and the seat at its right will be numbered as 2, the leftmost seat of the row behind the first row will be numbered just after the rightmost seat of the first row.

# 3 Definition of Commands

## 3.1 General Information

Your code must trim each line and skip the lines that they are empty after the trimming, and your code must process and printout trimmed version of each command to the output file as shown in the Sample I/O Files. Definitions below includes all kinds of valid usages, beware that anything out of this format is invalid and your program must successfully handle all of these situations. Some example invalid situations could be improper command line arguments, improper number of arguments, type error for arguments (such as delivering string where it is supposed to be integer), erroneous commands etc. Your program must ignore erroneous commands just notifying the user by `"ERROR: <ERROR_CAUSE>"` format. You must be aware of everything shared on Piazza and your program must contain any kind of errors even if it is not stated at given Sample I/Os, if an error was stated at Sample I/Os before the submit system allows you to submit your assignment, you must use the exact same format at the given I/O, otherwise, you can use any kind of description at the <ERROR_CAUSE> part as long as it describes the error well. Please do not hesitate to ask about edge cases as I can simply update the Sample I/Os according to your questions for making this project more understandable for you.

## 3.2 Initializing a Voyage

This command's mechanics are as follows:

```
INIT_VOYAGE <TAB> Minibus <TAB> <ID> <TAB> <FROM> <TAB> <TO> <TAB> <#ROWS> <TAB> <PRICE>
INIT_VOYAGE <TAB> Standard <TAB> <ID> <TAB> <FROM> <TAB> <TO> <TAB> <#ROWS> <TAB> <PRICE> <TAB> <REFUND_CUT>
INIT_VOYAGE <TAB> Premium <TAB> <ID> <TAB> <FROM> <TAB> <TO> <TAB> <#ROWS> <TAB> <PRICE> <TAB> <REFUND_CUT> <TAB> <PREMIUM_FEE>
```

Sample commands could be as follows:

```
INIT_VOYAGE <TAB> Standard <TAB> 7 <TAB> Ankara <TAB> İstanbul <TAB> 12 <TAB> 400 <TAB> 12
```

Output for this command is as follows:

```
Voyage 7 was initialized as a standard (2+2) voyage from Ankara to İstanbul
with 400 TL priced 48 regular seats.  Note that refunds will be 12% less
than the paid amount.
```

Explanation: Command above, creates a standard (2+2) voyage from Ankara to İstanbul with ID of 7 (which is unique) with 12x(2+2)=48 standard seats price of 400.00 TL for each, and refunds will be 352.00 TL as the refund cut is 12%.

```
INIT_VOYAGE <TAB> Premium <TAB> 18 <TAB> İstanbul <TAB> Ankara <TAB> 14 <TAB> 420 <TAB> 10 <TAB> 15
```

Output for the command is as follows:

```
Voyage 18 was initialized as a premium (1+2) voyage from İstanbul to
Ankara with 420.00 TL priced 28 regular seats and 483.00 TL priced 14
premium seats.  Note that refunds will be 10% less than the paid amount.
```

Explanation: Command above, creates a premium (1+2) voyage from İstanbul to Ankara with ID of 18 (which is unique) with 14x(1+2)=42 seats price of 420 TL for standard seats and 420x(1+15%)=483 TL for the premium ones, and refunds will be 378.00 TL and 434.70 TL respectively as the refund cut is 10%.

```
INIT_VOYAGE <TAB> Minibus <TAB> 6 <TAB> Ankara <TAB> Polatlı <TAB> 8 <TAB> 100
```

Output for this command is as follows:
`Voyage 6 was initialized as a minibus (2) voyage from Ankara to Polatlı with 100.00 TL priced 16 regular seats. Note that minibus tickets are not refundable.`

Explanation: Command above, creates a minibus (2) voyage from Ankara to Polatlı with ID of 6 (which is unique) with 8x(2)=16 standard seats price of 100.00 TL for each, and tickets are not refundable.

### 3.3 Selling Ticket

This command's mechanics are as follows:
`SELL_TICKET <TAB> <VOYAGE_ID> <TAB> <SEAT_NUMBER>`
`SELL_TICKET <TAB> <VOYAGE_ID> <TAB> <SEAT_NUMBER>_<SEAT_NUMBER>...`

Note that more than one seat can be sold at the same command just adding seat numbers at the end of the command with a underscore separator between the seat numbers. Moreover, command must not processed even if one of the seats are not available for selling.

Sample commands could be as follows:
`SELL_TICKET <TAB> 15 <TAB> 7`

Output for the command is as follows:
`Seat 7 of the Voyage 15 from <FROM> to <TO> was successfully sold for <PRICE> TL.`

Explanation: Command above sells seat 7 of the voyage with ID of 15.

`SELL_TICKET <TAB> 9 <TAB> 5_6.`

Output for the command is as follows:
`Seat 5-6 of the Voyage 9 from <FROM> to <TO> was successfully sold for <PRICE> TL.`

Explanation: Command above sells seat 5 and 6 of the voyage with ID of 9.
Note that all of the ticket prices must added to revenue of the voyage.

### 3.4 Refunding Ticket

This command's mechanics are as follows:
`REFUND_TICKET <TAB> <VOYAGE_ID> <TAB> <SEAT_NUMBER>`
`REFUND_TICKET <TAB> <VOYAGE_ID> <TAB> <SEAT_NUMBER>_<SEAT_NUMBER>...`

Note that more than one seat can be refunded as the same mechanic with the command for selling including the not processing part.

Sample commands could be as follows:
`REFUND_TICKET <TAB> 2 <TAB> 30`

Output for the command is as follows:
`Seat 30 of the Voyage 2 from <FROM> to <TO> was successfully refunded as <REFUND_AMOUNT> TL.`

Explanation: Command above refunds seat 30 of the voyage with ID of 2.

`REFUND_TICKET <TAB> 17 <TAB> 10_11.`

Output for the command is as follows:
`Seat 10-11 of the Voyage 17 from <FROM> to <TO> was successfully refunded as <REFUND_AMOUNT> TL.`

Explanation: Command above sells seat 10 and 11 of the voyage with ID of 17.
Note that all of refund amount must subtracted from revenue of the voyage.

### 3.5 Printing Voyage

This command's mechanics are as follows:
`PRINT_VOYAGE <TAB> <VOYAGE_ID>`
Sample command could be as follows:
`PRINT_VOYAGE <TAB> 24`
Output for the command could be found at Sample I/O files. It simply prints Voyage itself with some information (Voyage ID, departure/arrival stations, total revenue etc.) as stated at the Sample I/O, stars represents the empty seats while the X's represents the allocated ones.
Explanation: Command above prints seating plan of the voyage with ID of 24.

### 3.6 Cancelling Voyage

This command's mechanics are as follows:
`CANCEL_VOYAGE <TAB> <VOYAGE_ID>`
Sample command could be as follows:
`CANCEL_VOYAGE <TAB> 13`
Output for the command could be found at Sample I/O files. It simply tells Voyage 13 was cancelled and then does the same job with the print voyage command. Note that all tickets that are available at the cancel stage must be treated as returned without any cut and they must be subtracted from the revenue of the voyage. **Moreover, you must printout last state of the bus just before the refunds, but you must calculate the revenue according to refunded situation.**
Explanation: Command above prints the seating plan of the voyage with ID of 13, and then cancels it.

### 3.7 Z Report

Sample command could be as follows:
`Z_REPORT`
Output for the command could be found at Sample I/O files. It simply prints out all of the available voyages at that moment in the ascending order of their ID. It can be imagined as all voyages prints themselves one by one according to their ID, and they are separated with 16 dashes (- character). At the beginning of the Z Report you must print `Z Report:` text. Note that, your program must printout Z Report at the end of the output whether or not requested, if Z Report is the last command at the input, you do not have to do anything special, but if the last command is not Z Report, then you have to printout Z Report without telling `COMMAND: Z_REPORT`.

## 4 Restrictions

- Your code must be able to execute on our department's developer server (dev.cs.hacettepe.edu.tr).
- You must obey given submit hierarchy and get score (1 point) from the submit system.
- **You must benefit from all of the four pillars of OOP; abstraction, encapsulation, inheritance, polymorphism; any solution that does not contain even one of these concepts will not be accepted, moreover, partial point deductions may exist due to partially erroneous usage of these concepts.**
- Your code must be clean, do not forget that main method is just a driver method that means it is just for making your code fragments run, not for using them as a main container, create classes and methods in necessary situations but use them as required. Moreover, use the four pillars of Object-Oriented Programming (Abstraction, Encapsulation, Inheritance, Polymorphism) if there is such a need, remember that your code must satisfy Object-Oriented Programming Principles, also you can benefit from exceptions and even if create your own exception class if you need any.
- You are encouraged to use lambda expressions which are introduced with **Java 8**.
- You must use JavaDoc commenting style for this project, and you must give brief information about the challenging parts of your code, do not over comment as it is against clean code approach. Design your comments so that if someone wants to read your code they should be able to easily understand what is going on. You can check here to access Oracle's own guide about JavaDoc Sytle.
- You can benefit from Internet sources for inspiration but do not use any code that does not belong to you.
- You can discuss high-level (design) problems with your friends but do not share any code or implementation with anybody.
- Do not miss the submission deadline.
- Source code readability is a great of importance. Thus, write READABLE SOURCE CODE, comments, and clear MAIN function. This expectation will be graded as "clean code".

- Use UNDERSTANDABLE names for your variables, classes, and functions regardless of the length. The names of classes, attributes and methods must obey to the Java naming convention. This expectation will be graded as "coding standards".
- You can ask your questions through course's Piazza group, and you are supposed to be aware of everything discussed in the Piazza group. General discussion of the problem is allowed, but **DO NOT SHARE** answers, algorithms, source codes and reports.
- All assignments must be original, individual work. Duplicate or very similar assignments are both going to be considered as cheating.
- Submit system for this homework will be opened a few days before deadline, so please be patient.

# 5 Execution and Test

Your code must be executed under **Java 8u401 (1.8.0_401)** at **dev.cs.hacettepe.edu.tr**. If your code does not run at developer server during the testing stage, then you will be graded as 0 for code part even if it works on your own machine. Sample run command is as follows:

- Either `javac8 BookingSystem.java` or `javac8 *.java` command for compilation.
- `java8 BookingSystem input.txt output.txt` command for run.

# 6 Grading

| Task | Point |
|---|---|
| **Output** | 30* |
| **Output (Including Errors)** | 50* |
| **Comments in JavaDoc Style** | 20* ** |
| **Total** | 100 |

*** Even though producing correct output and commenting seems like enough to get full credit, you must obey to the given rules in the PDF (for example using concept of OOP and four pillars of OOP etc.) otherwise you may face with some point deductions which may result with a grade that is as low as zero. There will be two overall multipliers about quality of your OOP and clean code separately which will vary in between 0 and 1! Note that there may be any other multipliers or point deductions in case of violation of the rules.**
**** The score of your comments will be multiplied by your overall score (excluding the comments part) divided by the maximum score that can be taken from these parts. Say that you got 60 from all parts excluding the comments and 20 from the comments part, your score for report is going to be 20*(60/80) which is 15 and your overall score will be 75.**

# 7 Submit Format

File hierarchy must be zipped before submitted (Not .rar, only not compressed .zip files because the system just supports .zip files).

```
- b<StudentID>.zip
    - <src>
       - BookingSystem.java
       - *.java (Optional)
```