CS102A Spring 2020 Assignment 4

Designer: WANG Wei, Tester: LIAO Weiduo, WU Shangxuan,YUN Zebin(Junit and Tester) , Coder: MAO Zunyao

Problems

Problem 1: Design a class named VehicleRun [Easy, 20 marks]

Problem 2: Design a class named Passenger [Medium, 25 marks]

Problem 3: Design a class named Bus [Medium, 30 marks]

Problem 4: Schedule buses according to the timetable [Medium, 25 marks]

Notes for this assignment

1. Use the language specified (Java) in the lab / problem description.
2. Use Main as the class name for the Problem 4, and DO NOT include package in your submission.
3. Submit solutions before deadline, and don't wait until the last minute. The server might be overloaded and your submission will wait a long time before getting judged.
4. The required method of problem 4 must be implemented, otherwise it will affect your score.
5. Finish problems by yourself and enjoy!(≧▽≦)/

Background:

Operational data is valuable information to guide further adjustment. Take bus operation as example: operation schedule, routes, charging standards are features. Count the number of passenger getting on and off could help to judge the flow on stations, the congestion level in bus and so on, these information will be used decide whether adjust the arrangement on bus to get a more valuable and comfortable performance. In the assignment you are asked to create few basic but important component in a simplified bus operation system.

Problem 1 Vehicle Run

Design a class named **VehicleRun**

VehicleRun describe one round of bus operation from start station to end station along a bus line. It record the departure time, bus line, whether it is through ticket or not,its fee, and update its income and passengers' number on every station while they get on and get off the vehicle.

(1) **7 private instance variables**

① **departureTime**(Sting): it is the depart time in String format. Such as "06:00" means the vehicle will depart at 06:00.

② **busLine**(ArrayList<String>): it stores every station's name(as String) from the start station to the end station of this bus line.

③ **throughTicket**(boolean): while it's true which means this vehiclerun is Through-Ticket, passenger SHOULD pay the Through-Ticket while get on. While it's false which means the the the fare SHOULD be calculated by the stations took.

④ **fee**(float): if this is through-ticket, fee is the through fare. If this is not through-ticket, the price SHOULD be calculated as the multiplication on fee and the number of stations the passenger took.

⑤ **income**<float>: it records the income of this vehicle run.

⑥ **pCntOn**(int [ ]) and **pCntOff**(int [ ]): every item of pCntOn records the count of passengers who get on the vehicle at the station, the order of items in pCntOn is

the same order as the stations in bus line. So does the pCntOff which records the count of passengers get off the vehicle at every stations in bus line.

(2) **1 constructor** with **4 parameters** as following

```
public VehicleRun(String departureTime, ArrayList<String> busLine, boolean throughTicket, float fee)
```

NOTIC: Both **pCntOn**, **pCntOff** and **income** SHOULD be initialized in the constructor.

(3) **1 public instance method** named **updateCntOnStation.**

In this method, update the value of item (whose index is same as the index of *xstation* in the busLine) in **pCntOn** and **pCntOff** according to the parameter *onNum* and *offNum*.

```
public boolean updateCntOnStation(String xstation,int onNum,int offNum)
```

NOTIC: If the parameter *xsation* is not an item of busLine, then return false, else return true.

(4) **1 public instance method** named **getFare.**

```
public boolean getFare( float fare)
```

In this method,update the instance variable "income" with adding the value of parameter *"fare"*, the value of *fare* SHOULD NOT be negative, if it's a negative, return false.

(5) **getter** and **setter** methods for each private fields. Please design your getter and setter methods in a standard way.

Problem 2 Passenger

Design a class named **Passenger**

Passenger here records basic info of passenger's transportation card, manage its balance and update the boarding station while passenger get on and get off the vehicle.

(1) **4 private attributes**

① **cnt**(int): static class variable with initialized value as 0, plus 1 while an Passenger object is created.

② **id**(int): instance variable. Its value is the order of the creation on object. Such as 1st created Passenger object's id is 1, 2nd created Passenger object's id is 2.

③ **balance**(float): it is the balance of the passenger's transportation card.

NOTIC:

1. balance is allowed to be negative.
2. if the balance is negative, passenger is not allowed to get on the bus.

④ **boardingStation**(String): it is the name of station the passenger get on. While passenger get off the bus, boardingStation is assigned as NULL.

(2) **2 constructors**

```
public Passenger()
public Passenger(float money )
```

NOTIC:

1. While invoke the constructor without parameter to create a Passenger object: "boardingStation" is initialized as NULL, "balance" is initialized as 0.0.

2. While invoke the constructor with a float parameter to create a Passenger object, "balance" is initialized as the parameter, "boardingStation" is initialized as NULL .

3. “cnt” is static variable with initialized value as 0, plus 1 while an Passenger object is created;

4. “id” is instance variable.Its value is the order of the creation on object. Such as 1st created Passenger object’s id is 1, 2nd created Passenger object’s id is 2.

(3) **1 public instance method** named **getOn**

```
public boolean getOn( VehicleRun vr,String station)
```

When a passenger get on at a *station*, this method is invoked.

While passenger get on a bus, update boarding Station and pay the fare if needed.

i. if it’s through ticket
   1. pay the ticket fee
   2. update the current vehicle run(*vr*)’s income by invoke the instance method getFare() of vehicle run(*vr*)
   3. update the boardingStation of passenger with *station.*

ii. if it’s NOT through ticket
   1. update the boardingStation of passenger with *station.*

NOTIC: in following situation, passenger get on failed,the method return false.

i. The station passenger get on is not an item of busLine of current vehicle operation vr.
ii. The balance of passenger is negative.
iii. Passenger has been already on the bus(boardingStation in not NULL).

(4) **1 public instance method** named **getOff**

```
public boolean getOff(VehicleRun vr,String station)
```

When a passenger get off at a *station*, this method is invoked.

While passenger get off a bus, he update boardingStation and pay the fare if needed

a-1) if it’s through ticket:

i. update the boardingStation as NULL.

a-2) if it’s not through ticket,calculate the fare and pay:

fare = fee* the number of stations passenger takes.

i. update the boardingStation as NULL.
ii. pay the ticket fee
iii. update the current vehicle run(*vr*)’s income by invoke the instance method getFare() of vehicle run(*vr*).

NOTIC: in following situation, passenger get off failed, the method return false.

a) *station illegal:*

i. the *station* passenger get off is not an item of busLine of current vehicle run *vr.*
ii. the *station* passenger get off and the station passenger get on is not belongs to the same busLine of current vehicle run *vr*.
iii. passenger get off but he/she havn’t get on the bus(boardingStation is NULL)
iv. the *station* passenger get off is ahead of the station passenger get on.(take index of station in busLine in consideration)

(5) **1 public instance method** named **refill**

```
public boolean refill(float money)
```

NOTIC:

This method refill the account of passenger with parameter *money*. If the value of money is bigger than 0 then update the balance and return true, if not then don't update and return false.

(6) **getter and setter methods** for each private fields. Please design your getter and setter methods in a standard way.

Problem 3 Bus

Design a class named Bus

Bus here record and manager a bus' basic info, including its license plates, the number of seats, all the vehicle runs and the index of its current vehicle run.

(1) **4 private instance variables**:

① **licensePlate**(Sting): the license plate of bus

② **seats**(int): the number of seats in bus

③ **vrs**(ArrayList<VehicleRun>): the set of all the Vehicle Runs of bus

④ **currentVrIndex**(int): the index of current running vehicle run in vrs.

(2) 2 **constructor**s

```
public Bus(String licensePlate, int seats)
public Bus(String licensePlate, int seats, ArrayList<VehicleRun> vrs)
```

NOTIC:

While "vrs" is not initialized by parameter,it SHOULD be initialized as NULL,and "currentVrIndex" SHOULD be initialized with 0 by default.

(3) **1 public instance method** named **arriveStation** :

```
public boolean arriveStation(int currentVrIndex,String xstation,
ArrayList<Passenger> passengersGetOn,
ArrayList<Passenger> passengersGetOff)
```

a. while the bus arrive the *xstation*, it update the current vehicle run(which is identified by the *currentVrIndex* as the index of vehicle run in vrs) with the size of *passengersGetOn* as the number of passengers who get on the bus at the station and the size of *passengersGetOff* as the number of passengers who get off the bus at the station.

TIPS: instance method in VehicleRun could be used here to update the item (identified by *xstation*) in pCntOn and pCntOff of current vehicle run(identified by *currentVrIndex* in bus).

b. Passengers who get on the bus(passengers who is in the ArrayList *passengersGetOn*) SHOULD pay for the fare if this is through-ticket, Passengers who get off the bus (passengers who is in the ArrayList *passengersGetOff*) SHOULD pay for the fare if this is NOT through-ticket.

NOTIC: In following situation, arrive Station failed, method arriveStation return false.

i. The number of passengers who get off at the terminal station of the busLine of current vehicle is not equal to the number of passengers who are in the bus at this moment(the number of passengers who get on and get off at current

station is not counted in the number of passengers who are in the bus at this moment).

ii. The number of passengers who get off at the station of busLine of current vehicle is larger than the number of passengers who are in the bus at this moment.(the number of passengers who get on and get off at current station is not counted in the number of passengers who are in the bus at this moment)

iii. There are passengers get off the bus while this is the 1st station of the busLine.

iv. There are passengers get on the bus while this is the terminal station of the busLine.

(4) **1 public instance method** named **endCurrentOperation**: A bus ends current operation, while it is NOT the LAST vehicle run in vrs, it means the currentVrIndex could be updated as the sum of currentVrIndex and 1, while it is the LAST vehicle run in vrs, DONOT update the currentVrIndex， print out the message as "WellDone,all the operations of Today has been finished,have a rest!!"

```
public void endCurrentOperation()
```

(5) **getter** and **setter** methods for each private fields. Please design your getter and setter methods in a standard way.

Problem 4 Schedule.

Now getting the number of buses which are waiting for schedule and the time-table of a busLine operation, make the arrangement for each bus (schedule the bus in round-robin) according the time table and print it out.

```
public class Schedule {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int busNum = sc.nextInt();
        String list = sc.next();
        StringBuilder schedule[] = Schedule(list, busNum);
        if (schedule != null)
            for (StringBuilder s : schedule) {
                System.out.println(s.toString());
            }
    }
public static StringBuilder[] Schedule(String timelist, int busNum) {}
```

NOTIC: Finish the Schedule() method, then change the class name from “Schedule” to “Main” while submit this code on OJ.

An example:

Input :

```
2   06:00_06:30_07:00_07:30_08:00
```

Explain: Here 2 is the number of buses waiting for schedule, “06:00_06:30_07:00_07:30_08:00” is the time-table of a busLine operation.

Output:

```
06:00_07:00_08:00
06:30_07:30
```

Explain: the timetable of bus1 is “06:00_07:00_08:00”, the timetable of bus2 is “06:30_07:30”.

NOTIC:

1. *Bus Number* SHOULD be positive(big than 0), if NOT print out “Bus Number is negative or zero, error!” and exit.
2. Every item in time-Table of busLine operation SHOULD be in 24-hour clock. if NOT print out “Time Table error!” and exit.

   The following timetable is NOT OK:

   a) “25:00” : the value of hour(25) is illegal
   b) ”02:61” : the value of minute(61) is illegal
3. The item (separated by ”_”) in time-table of busLine operation SHOULD be in order as time fly, which means that the item in front SHOULD be earlier than later in time. if NOT print out “Time Table error!” and exit.

   For example:

   Following timetable is NOT OK:

   1) ”12:00_06:00_07:00” is illegal: because the front “12:00” is later than 06:00 and 07:00.

   Following timetable is OK:

   1) “06:00_07:00_08:00”
   2) “06:00_07:00_07:00_07:00_08:00” which means 07:00 is a crowded time and it is needed to take 3 buses in operation at 07:00.
4. While the bus Number is bigger than the numbers of items(separated by “_”) in time-table of busLine operation , which means the rest buses are backups, it’s OK.

An example:

Bus Number is 5 while busLine operation *timeTable* is “06:00_07:00_08:00”

Input

```
5    06:00_07:00_08:00
```

Ouput

```
06:00
07:00
08:00
```

5. While the item in time-table of busLine operation repeated, and its repeated times is bigger than bus number, schedule failed, print out “Dispatching failed!” and exit.

Input

```
2    06:00_06:00_06:00_07:00_08:00
```

Ouput

```
Dispatching failed!
```