POSTED: February 3.    DUE DATE: February 17.

Make sure you read the whole assignment before attempting to answer the questions. Use the code provided. You should NOT define new classes or new members for existing classes.

General Introduction

We will implement a simplified solution for making reservations on a train. A train comprises of different types of carriages, where new carriages are added to a train according to particular criteria which is explained in more detail in the text.

There are different types of carriages but they all have a carriage plan which can be described by a two-dimensional array. Each carriage plan comprises of a finite number of rows and columns. It should be possible to use the individual methods, which are to be implemented, on carriages of all types. In particular, we will concentrate on two types of carriages:

- **Sleeping car**
  Figure 1 shows the plan of a sleeping car. A sleeping car comprises of 6 compartments, there each compartment has 3 sleeping berths. The rows represent compartments and the columns represent different types of sleeping berths. The 3 sleeping berths in each compartment are denoted as follows:
  - *lower berth*, and column number 0 denotes all the lower berths in a sleeping car.
  - *middle berth*, and column number 1 denotes all the middle berths in a sleeping car.
  - *upper berth*, and column number 2 denotes all the upper berths in a sleeping car.

  i.e. each column number denotes a different type of place that can be reserved in a sleeping car. Each berth in a sleeping carriage has a place number as shown in figure 1.

- **Saloon car**
  Figure 2 shows the plan of a saloon car. A saloon car comprises of 9 rows of seats, there each row of seats has 4 seats. The rows represent seat-rows and the columns represent different types of seats. The 4 seats in each seat-row are denoted as follows:
  - *window-left*, and column number 0 denotes all such seats in a saloon car.
  - *aisle-left*, and column number 1 denotes all such seats in a saloon car.
  - *aisle-right*, and column number 2 denotes all such seats in a saloon car.
  - *window-right*, and column number 3 denotes all such seats in a saloon car.

  i.e. each column number denotes a different type of place that can be reserved in a saloon car. Each seat in a saloon car has a place number as shown in figure 2.

Note that a place in a carriage can be denoted in two different ways: either by its place number or by (i,j)-pair, called its place position, there i is the row-number and j is the column-number. In addition to the plan, a carriage keeps track of number of places that are vacant at any given time. Figure 1 and 2 show this for a sleeping car and a saloon car respectively. Each sleeping car has 3 different type of places, and a saloon car has 4 different types of places given by the number of column in the carriage plan. In addition, a carriage has a carriage number which is allocated as carriages are put in service for a train.
### Figure 1: Sleeping car

<table>
<thead>
<tr>
<th>Compartment nr. 0 [0]</th>
<th>Compartment nr. 1 [1]</th>
<th>...</th>
<th>Compartment nr. 5 [5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
</tr>
</tbody>
</table>

No. of compartments is 6

No. of berths pr. compartment is 3

No. of vacant places of each type: 6 6 6

### Figure 2: Saloon car

<table>
<thead>
<tr>
<th>Seat-row nr. 0 [0]</th>
<th>Seat-row nr. 1 [1]</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>34</td>
<td>35</td>
<td></td>
</tr>
</tbody>
</table>

No. of seat-rows is 9

No. of seats pr. seat-row is 4

No. of vacant places of each type: 9 9 9 9
Criteria for inserting a new carriage and place reservation are as follows:

- A new carriage is only introduced in a train if all the places in the old carriage have been reserved. In other words, a sleeping car in a train must be filled before a new sleeping car is inserted in the train, and likewise for a saloon car.

- The above rule can lead to a passenger not getting the type of place which the passenger wishes. In this case, the system should reserve a place of the type that currently has the least occupancy, i.e. the type of place that has the least number of places occupied in the carriage.

- It is the first vacant place of any type which is reserved in a carriage. The places with lower place numbers being filled before places with higher place numbers.

The total number of carriages in a train cannot exceed the maximum number of carriages specified for the train. To begin with, a train is created with 2 carriages, one of each type: a sleeping car and a saloon car. (Note that the number of carriages in a train cannot be less than 2.) A train has two references which denote the current sleeping car and the current saloon car which currently should be used for place reservation. Reservation in a train means either reserving a particular type of berth in the current sleeping car or reserving a particular type of seat in the current saloon car, all according to the criteria for inserting new carriages and place reservation described above.

Additional details will be given in the problems where appropriate.

Problem 1.(11 parts).

Class Carriage.

/** This abstract class defines a carriage. */
* @author CS211 - Spring 2000
* @version 1.0
*/

public abstract class Carriage {

    /** Carriage number */
    private int carriageNumber;
    /** Number of rows in a carriage */
    private int noOfRows;
    /** Number of columns in a carriage */
    private int noOfColumns;
    /** Plan for carriage. 
     * true indicates occupied.
     * false indicates vacant. 
     */
    private boolean carriagePlan[][];
    /** Array to keep track numbers of vacant 
     * places of each type of place. 
     */
    private int noOfVacantPlaces[];

    /** 1a) Constructs a new Carriage class that initializes 
     * the carriageNumber, noOfRows, noOfColumns, carriagePlan, 
     * and noOfVacantPlaces. 
     */
public Carriage(int carriageNo, int rows, int columns) { //...
}

/** 1b) Returns carriage number of the carriage. */
public int getCarriageNo() { //...
}

/**
 * 1c) Sets the given place in the carriage plan to occupied.
 * Update the count of different type of vacant places.
 * @param (i,j) Place location.
 */
private void setOccupied(int i, int j) { //...
}

/**
 * 1d) Checks if the carriage has vacant places of the type specified
 * in the parameter.
 */
public boolean hasVacantPlace(int typePlace) { //...
}

/** 1e) Check if the carriage is full. */
public boolean isFull() { //...
}

/**
 * 1f) Find the type of place in the carriage which has least
 * occupancy and returns -1 if the carriage is full.
 */
public int typePlaceWithLeastOccupancy() { //...
}

/**
 * 1g) Find row-number of the first vacant place of the specified
 * type in the carriage.
 * If there are no vacant places of the specified type, return -1.
 * @param typePlace Type of Place.
 * @return int The row number found.
 */
private int findRowNoOfFirstVacantPlace(int typePlace) { //...
}

/**
 * 1h) Convert (i,j)-pair in the plan to place number in the carriage.
 * @param (i,j) Place location pair.
 * @return int Place Number.
 */
private int placePositionToPlaceNo(int i, int j) { //...
}

/**
 * 1i) Reserve the first vacant place of the specified type.
 * Returns place number in the carriage.
 */
public int reserveFirstVacantPlace(int typePlace) { //...
}

public void drawCarriagePlan(int placeNo) { //...
}

public abstract String getNameOfTypePlace(int typePlace);

a) Write the constructor which creates and initializes the necessary data structures for a carriage.
Carriage(int carriageNo, int rows, int columns) {...}

b) Write the method getCarriageNo which returns the carriage number of the carriage.
public int getCarriageNo() { ... }

c) Write the method setOccupied which marks the place given by (i,j) as occupied in the carriage plan, and also updates the count of vacant places in the carriage.
public void setOccupied(int i, int j) {...}

d) Write the method hasVacantPlace which checks if there are vacant places of the specified type in the carriage.
public boolean hasVacantPlace(int typeOfPlace) {...}

e) Write the method isFull which checks if the carriage is full.
public boolean isFull() {...}

f) Write the method typeOfPlaceWithLeastOccupancy which returns the type of place that has the least occupancy in the carriage.
public int typeOfPlaceWithLeastOccupancy() {...}

g) Write the method findRowNoOfFirstVacantPlace which returns the row-number of the first vacant place of the specified type in the carriage. If there are no vacant places of the specified type, the method returns -1.
public int findRowNoOfFirstVacantPlace(int typeOfPlace) {...}

h) Write the method placePositionToPlaceNo which converts the place position given by the (i,j)-pair, where i is row-number and j is column-number, to place number in the carriage plan. See figure 3.
public int placePositionToPlaceNo(int i, int j) {...}
i) Write the method `reserveFirstVacantPlace` which reserves the first vacant place of the type specified if this is possible and returns the place number in the carriage, otherwise it returns -1.

```java
public int reserveFirstVacantPlace(int typeOfPlace) {...}
```

j) Write the method `drawCarriagePlan` which “draws” the plan of a carriage on the terminal, where the specified place number is shown with a '*', occupied places are shown with a '+', and vacant places are shown with a '-'.

```java
public void drawCarriagePlan(int placeNo) {...}
```

Example of output:
```
0       ++++
1       ++++
2       *+++  
3       ++++  
4       ++++  
5       ++++  
6       ++++  
7       ++++  
8       ---+
```

Problem 2.(3 parts).

Classes SleepingCar and SaloonCar

```java
/**
 * Class defines a sleeping car.
 * @author CS211 - Spring 2000
 * @version 1.0
 */

class SleepingCar extends Carriage {

    /** Sleeping Car type */
    public static final int SLEEPING_CAR_TYPE = 1;
    /** Number of compartments in a Sleeping Car */
    public static final int NO_OF_COMPARTMENTS_PER_SLEEPING_CAR = 6;
    /** Number of Berths in a compartment */
```
public static final int NO_OF_BERTHS_PER_COMPARTMENT = 3;
/** Lower Berth type */
public static final int LOWER_BERTH = 0;
/** Middle Berth type*/
public static final int MIDDLE_BERTH = 1;
/** Upper Berth type*/
public static final int UPPER_BERTH = 2;

/** 2a) Calls constructor of the superclass with carriage-number, etc. */
public SleepingCar(int carriageNo) { //...
}

/** 2b) Defines the abstract method from the Carriage class.
* Converts the type of place specified by column-number to place name. */
public String getNameOfTypePlace(int typePlace) { //...
}

/** 2c) Overrides the method from the Object class.
* Returns a string with indicates it is a sleeping car with a
* particular carriage number. */
public String toString() { //...
}

/**
 * Class defines a saloon car.
 * @author CS211 - Spring 2000
 * @version 1.0
 */

class SaloonCar extends Carriage {

    /** Saloon Car type */
    public static final int SALOON_CAR_TYPE = 0;
    /** Number of rows in a Saloon Car */
    private static final int NO_OF_SEATROWS_PER_SALOON_CAR = 9;
    /** Number of seats per row in a Saloon Car */
    private static final int NO_OF_SEATS_PER_SEATROW = 4;
    /** Window left seat type*/
    public static final int WINDOW_LEFT = 0;
    /** Window left seat type*/
    public static final int AISLE_LEFT = 1;
    /** Aisle right seat type*/
    public static final int AISLE_RIGHT = 2;
    /** Window right seat type*/
    public static final int WINDOW_RIGHT = 3;

    /** 2a) Calls constructor of the superclass with carriage-number, etc. */
    public SaloonCar(int carriageNo) { //...
    }

    /** 2b) Defines the abstract method from the Carriage class.
     * Converts the type of place specified by column-number to place name.
     * @return String of place name.
     */
    public String getNameOfTypePlace(int typePlace) { //...
    }
/** 2c) Overrides the method from the Object class.
 * @return A string with indicates it is a saloon car with a particular
 * carriage number.
 */
public String toString() { //...
}

In this question all the parts should be answered for both the subclass SleepingCar and the subclass SaloonCar.

a) Write the constructors for the subclasses:
   public SleepingCar(int carriageNo) {...}
   and
   public SaloonCar(int carriageNo) {...}

b) Write the method getNameOftypeOfPlace which returns a string containing the name of the
type of place in the subclasses.
For example, type of place 1 for a sleeping car returns the following string: Middle-berth
while the type of place 1 for a saloon car returns the following string: Aisle-left
public String getNameOftypeOfPlace(int typeOfPlace) {...}

c) Write the method toString which returns a string which identifies the type of carriage
represented by the subclasses.
For example, a sleeping car returns the following string: Sleeping car (carriage
no. 12)
while a saloon car returns the following string: Saloon car (carriage no. 13)
public String toString() {...}

Problem 3.(4 parts).

Class Train.

/**
 * Defines a class for making place reservation on a train
 * @author CS211 - Spring 2000
 * @version 1.0
 */
class Train {
 /* Array for carriages. */
 private Carriage[] carriages;
 /* Index of last carriage inserted in the train */
 private int currentIndex;
 /* Current Saloon car for making a reservation */
 private Carriage currentSaloonCar;
 /* Current Sleeping car for making a reservation */
 private Carriage currentSleepingCar;
/**
 * 3a) Constructs an array of carriages with maxNoOfCarriages
 * and initializes all instance variables.
 * If input is invalid, error message is printed.
 */
public Train(int maxNoOfCarriages) { //...
}

/** 3b) Prints out a summary for the train, consisting of:
 * - number of saloon cars in the train.
 * - number of sleeping cars in the train.
 */
public void trainSummary() { //...
}

/**
 * 3c) Insert a new carriage of the type specified by the parameter
 * thisCarriage.
 * Reference to current sleeping car/saloon car in the train is
 * updated when a new carriage is inserted.
 * Reference to the newly inserted carriage is returned.
 * If an error occurred, error is printed and null is returned.
 */
private Carriage insertNewCarriage(Carriage thisCarriage) { //...
}

/**
 * Makes the reservation in the appropriate carriage with the
 * given type of place.
 */
public void placeReservation(int carriageType, int typePlace){
    switch(carriageType){
        case SaloonCar.SALOON_CAR_TYPE:
            placeReservation(currentSaloonCar,typePlace);
            break;
        case SleepingCar.SLEEPING_CAR_TYPE:
            placeReservation(currentSleepingCar, typePlace);
            break;
        default:
            System.out.println("Error: Unknown Carriage type. " +
                                " Reservation not made!");
    }
}

/** 3d) Makes a reservation according to the criteria given in the text.
 * Parameter currentCarriage specifies the carriage which is the current
 * carriage for place reservation.
 * Parameter typePlace specifies the type of the place which is desired.
 * *
 * (1) Print out the specified place type.
 * (2) Make the reservation according to the text.
 * (3) Print the following:
private void placeReservation(Carriage currentCarriage, int typePlace) {
    //...
}

a) Write the constructor **Train**(int maxNoOfCarriages) which creates and initializes the necessary data structures for a train: the array for carriages and the current carriages for place reservation.

    public Train(int maxNoOfCarriages) {...}

b) Write the method **trainSummary** which writes on the terminal a summary of different types of carriages in the train.

    Example of output:
    
    Train Summary:
    No. of Sleeping carriages: 11
    No. of Saloon carriages: 7

    public void trainSummary() {...}

c) Write the method **insertNewCarriage** which inserts a new carriage of the type specified by the parameter **carriageType**. The current sleeping car/saloon car references are updated accordingly when a carriage is inserted in the train. The method returns the reference of the carriage which is inserted.

    public Carriage insertNewCarriage(Carriage carriageType) {...}

d) Write the method **placeReservation** which makes a reservation according to the criteria specified in the text. The parameter **currentCarriage** is the current carriage for making the reservation. The parameter **typeOfPlace** specifies the type of place desired. The method writes on the terminal the following information: the type of place specified, the carriage plan showing the place reserved, the carriage type and carriage number, and name of the type of place that is actually reserved.

    Example of output:
    
    Specified place type: window-right
    0       ++++
    1       ++++
    2       **+++*
    3       +---+
    4       ++--
    5       ++--
    6       +---
    7       +---
    8       ---

    Place number 9 in Saloon Car (Carriage no. 6) aisle-left reserved.

    public void placeReservation(Carriage currentCarriage, int typeOfPlace) {...}
Problem 4. (2 parts).

Class TrainReservationSystem
/**
 * This class implements the user interface for making reservations
 * on a train.
 * @author CS211 - Spring 2000
 * @version 1.0
 */

public class TrainReservationSystem {

/**
 * Maximum number of carriages allowed in a train */
 public static final int MAX_NO_OF_CARRIAGES = 10;

/**
 * Menu option for Train Summary */
 public static final int TRAIN_SUMMARY_OPTION = 0;
 /**
 * Menu option for Reservation menu*/
 public static final int RESERVATION_MENU_OPTION = 1;
 /**
 * Menu option for Exit*/
 public static final int EXIT_MAIN_MENU_OPTION = 2;

/**
 * Menu option for Sleeping Car*/
 public static final int SLEEPING_CAR_OPTION = 0;
 /**
 * Menu option for Saloon car*/
 public static final int SALOON_CAR_OPTION = 1;
 /**
 * Menu option for Main Menu*/
 public static final int EXIT_RESERVATION_MENU_OPTION = 2;

/**
 * Menu option for Lower Berth*/
 public static final int LOWER_BERTH_OPTION = SleepingCar.LOWER_BERTH;
 /**
 * Menu option for Middle Berth*/
 public static final int MIDDLE_BERTH_OPTION = SleepingCar.MIDDLE_BERTH;
 /**
 * Menu option for Upper Berth*/
 public static final int UPPER_BERTH_OPTION = SleepingCar.UPPER_BERTH;
 /**
 * Menu option for Back menu Button*/
 public static final int EXIT_SLEEPING_CAR_MENU_OPTION = 3;

/**
 * Menu option Window left button*/
 public static final int WINDOW_LEFT_OPTION = SaloonCar.WINDOW_LEFT;
 /**
 * Menu option Aisle Left button*/
 public static final int AISLE_LEFT_OPTION = SaloonCar.AISLE_LEFT;
 /**
 * Menu option Aisle Right button*/
 public static final int AISLE_RIGHT_OPTION = SaloonCar.AISLE_RIGHT;
 /**
 * Menu option Window Right button*/
 public static final int WINDOW_RIGHT_OPTION = SaloonCar.WINDOW_RIGHT;
 /**
 * Menu option Back menu button*/
 public static final int EXIT_SALOON_CAR_MENU_OPTION = 4;

/**
 * Current train */
 private static Train currentTrain;

 /** Displays the Main Menu */
 private static void displayMainMenu() {
 System.out.println("\n\n");
}
System.out.println("==== Train Reservation System ==== ");
System.out.println("  " + TRAIN_SUMMARY_OPTION +": Train Summary");
System.out.println("  " + RESERVATION_MENU_OPTION +": Make a Reservation");
System.out.println("  " + EXIT_MAIN_MENU_OPTION +": Quit");
System.out.print("Make Your Selection:");
}

/** Displays Reservation menu */
private static void displayReservationMenu() {
    System.out.println("\n\n");
    System.out.println("==== Reservation Menu ==== ");
    System.out.println("  " + SLEEPING_CAR_OPTION +": Sleeping Carriage");
    System.out.println("  " + SALOON_CAR_OPTION +": Saloon Carriage");
    System.out.println("  " + EXIT_RESERVATION_MENU_OPTION +": back");
    System.out.print("Make Your Selection:");
}

/** Displays Sleeping Menu */
private static void displaySleepingMenu() {
    System.out.println("\n\n");
    System.out.println("==== Sleeping Menu ==== ");
    System.out.println("  " + LOWER_BERTH_OPTION +": lower berth");
    System.out.println("  " + MIDDLE_BERTH_OPTION +": middle berth");
    System.out.println("  " + UPPER_BERTH_OPTION +": upper berth");
    System.out.println("  " + EXIT_SLEEPING_CAR_MENU_OPTION +": back");
    System.out.print("Make Your Selection:");
}

/** Displays Saloon Menu */
private static void displaySaloonMenu() {
    System.out.println("\n\n");
    System.out.println("==== Saloon Menu ==== ");
    System.out.println("  " + WINDOW_LEFT_OPTION +": window-left");
    System.out.println("  " + AISLE_LEFT_OPTION +": aisle-left");
    System.out.println("  " + AISLE_RIGHT_OPTION +": aisle-right");
    System.out.println("  " + WINDOW_RIGHT_OPTION +": window-right");
    System.out.println("  " + EXIT_SALOON_CAR_MENU_OPTION +": back");
    System.out.print("Make Your Selection:");
}

/** Logic for the Sleeping Menu */
private static void sleepingMenu() {
    int input;
    while (true) {
        displaySleepingMenu();
        input = Stdin.getInt();
        System.out.println();
        switch(input) {
            **/
case LOWER_BERTH_OPTION:
    currentTrain.placeReservation(
        SleepingCar.SLEEPING_CAR_TYPE,
        SleepingCar.LOWER_BERTH);
    return;

case MIDDLE_BERTH_OPTION:
    currentTrain.placeReservation(
        SleepingCar.SLEEPING_CAR_TYPE,
        SleepingCar.MIDDLE_BERTH);
    return;

case UPPER_BERTH_OPTION:
    currentTrain.placeReservation(
        SleepingCar.SLEEPING_CAR_TYPE,
        SleepingCar.UPPER_BERTH);
    return;

case EXIT_SLEEPING_CAR_MENU_OPTION:
    return;

default:
    System.out.println("Invalid selection.");
}
System.out.println();

/** 4a) Logic for the Saloon Menu */
private static  void saloonMenu() { //...
}

/** 4b) Logic for the reservation menu */
private static void reservationMenu() { //...

/** Logic for the main menu. */
private static void mainMenu() {
    int input=0;
    while (input != EXIT_MAIN_MENU_OPTION) {

        displayMainMenu();
        input = Stdin.getInt();

        switch(input) {
            case TRAIN_SUMMARY_OPTION:
                currentTrain.trainSummary();
                break;
            case RESERVATION_MENU_OPTION:
                reservationMenu();
                break;
            case EXIT_MAIN_MENU_OPTION:
                break;
            default:
                System.out.println("Invalid selection.");
                break;
        }
    }
}
}
/** Initializes user input and train, and starts the system. */
public static void main( String args[] ) {
    //Create a train that can have maximum no. of carriages.
    currentTrain = new Train(MAX_NO_OF_CARRIAGES);

    mainMenu();
}

a) Write the method saloonMenu() which implements the logic of the making a reservation in a saloon car. This logic is analogous to the sleepingMenu().
private static void saloonMenu() { ... }

b) Write the method reservationMenu() which implements the logic of the making a reservation either in a sleeping car or a saloon car.
private static void reservationMenu() { ... }

**Deliverables:**

- A cover sheet with only the following information. Failure to do so will be penalized by at least 5 points.
  - The name of the course and semester: CS211, Spring 2000
  - The assignment number
  - The assignment due date
  - Your name, identification number, e-mail address and the section you attend.
  - Your partner's name, identification number, e-mail address and the section your partner attends (if you have a partner)
- A class diagram showing the relationships between all the classes.
- A sequence diagram that shows how a reservation is made either in a sleeping car or a saloon car when the method placeReservation(Carriage currentCarriage, int typeOfPlace) is invoked on an object of class Train.
- Code for all the classes. The code should be documented using javadoc comments. The generated html documentation files should not be submitted.

No hand-written documentation will be accepted.