**44-542 Object-Oriented Programming**

**Railways Project**

**Objective:** Covers all concepts in this course.

**Instructions:**

1. Read every instruction carefully and follow them.
2. Do not hard code any values unless specified and you must follow the given naming conventions.
3. Check the given sample output to understand how the results need to be printed.
4. Use **@author**, **@param,** and **@return** annotations wherever possible.
5. **@author** notation should contain your full name.
6. Generate proper Javadoc comments for all the classes with clear and thorough descriptions of their contents.
7. Ensure your code is indented properly. Use **shift+alt+f** to indent your code.
8. Each class that you create for this assignment should include the information given below.



1. Here, **NN** is your section number.

**How to create and view Javadoc in NetBeans:**

1. Select the project that you want to generate Javadoc from the Projects pane. At the top, click on the **Run** tab and then click on **Generate Javadoc**.
2. If the project build tool is Maven: Go to the project location in the Files folder -> target -> site -> apidocs -> click on the index.html file to access the Javadoc.

**Steps for Project Creation:**

1. Create a Java project in **NetBeans** (Click on File -> New Project -> Select Categories as Java with Maven and Projects as Java Application -> Click on Next) and name the project as **S######Project,** where **S######** is your student ID, for instance, **S123456Project**.
2. Create a driver class and free feel to write the logic on your own.
3. Check the sample output to know the proper appearance of your output.
4. The input files **trains.txt, journeyPath.txt** are provided to you, use the same file to read the data.
5. Create a **package** called **edu.nwmissouri.spring24.cs44542.secNN.railways** and create the classes as per the UML diagram (Figure 1).

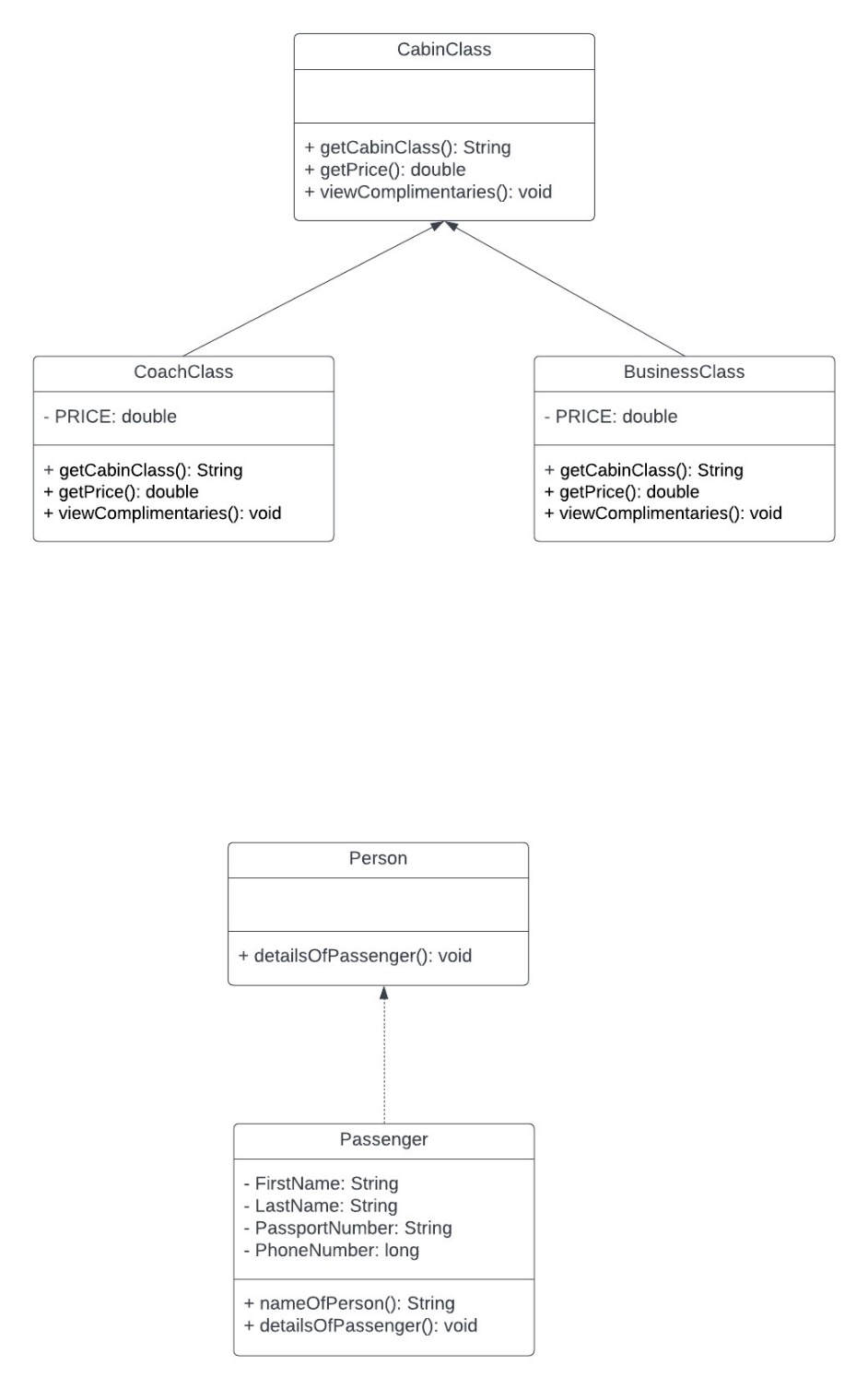


Figure 1 UML

**CabinClass:** This is an abstract class that has the methods mentioned in Table 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Return Type** | **Method Name** | **Parameters** |
| public | String | **getCabinClass()** | None |
| public | double | **getPrice()** | None |
| public | void | **viewComplimentaries()** | None |

Table 1 CabinClass methods

1. **Abstract Methods:**
   1. **getCabinClass():** This method is declared without an implementation (abstract). It is implemented by subclasses to return the name or type of the cabin class.
   2. **getPrice():** Another abstract method to be implemented by subclasses to return the price of the cabin.
   3. **viewComplimentaries():** An abstract method meant to print out the list of complimentary services or items associated with the cabin class.

**BusinessClass:** This is a concrete class that extends **CabinClass** that has the attributes and methods mentioned in Table 2 and Table 3 respectively.

**Attributes**: The **price** attribute is constant and is initialized to the value of 482.76.

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Attribute** | **Type** | **Attribute Description** |
| private | PRICE | double | Price of the class |

Table 2 BusinessClass attributes

**Methods:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Return Type** | **Method Name** | **Parameters** |
| public | String | **getCabinClass()** | None |
| public | double | **getPrice()** | None |
| public | void | **viewComplimentaries()** | None |

Table 3 CabinClass methods

1. Create a default constructor.
2. **getCabinClass():** Overrides the getCabinClass() method inherited from the CabinClass abstract class. It returns the identifier "BS" to denote the business class cabin.
3. **getPrice():** Overrides the getPrice() method inherited from the CabinClass abstract class. It returns the price of the business class cabin.
4. **viewComplimentaries():** Overrides the viewComplimentaries() method inherited from the CabinClass abstract class. It prints out a list of complimentary services offered in the business class cabin as shown below.

A white text with black text

Description automatically generated

**CoachClass:** This is a concrete class that extends **CabinClass** that has the attributes and methods mentioned in Table 4 and Table 5 respectively.

**Attributes**: The **price** attribute is constant and is initialized to the value of 298.45.

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Attribute** | **Type** | **Attribute Description** |
| private | PRICE | double | Price of the class |

Table 4 CoachClass attributes

**Methods:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Return Type** | **Method Name** | **Parameters** |
| public | String | **getCabinClass()** | None |
| public | double | **getPrice()** | None |
| public | void | **viewComplimentaries()** | None |

Table 5 CoachClass methods

1. Create a default constructor.
2. **getCabinClass():** Overrides the getCabinClass() method inherited from the CabinClass abstract class. It returns the identifier "CC" to denote the business class cabin.
3. **getPrice():** Overrides the getPrice() method inherited from the CabinClass abstract class. It returns the price of the Coach class cabin.
4. **viewComplimentaries():** Overrides the viewComplimentaries() method inherited from the CabinClass abstract class. It prints out a list of complimentary services offered in the Coach class cabin as shown below.

A white background with black text

Description automatically generated

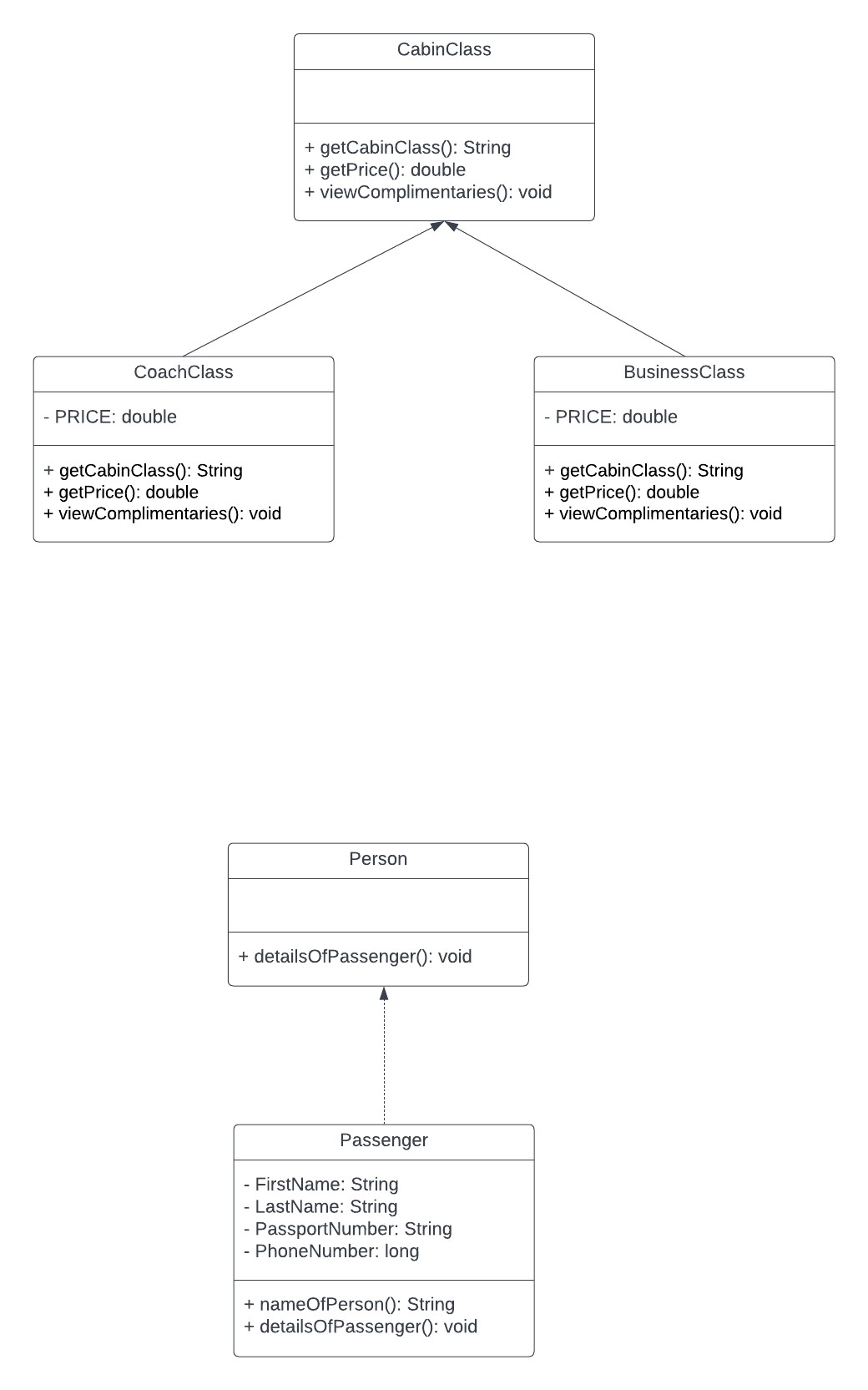


Figure 2 UML

**Person:** This is an Interface that has the methods mentioned in Table 6.

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Return Type** | **Method Name** | **Parameters** |
| public | void | **detailsOfPassenger()** | None |

Table 6 Person methods

**Passenger:** This is a concrete class that implements **Person** that has the attributes and methods mentioned in Table 7 and Table 8 respectively.

**Attributes**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Attribute** | **Type** | **Attribute Description** |
| private | firstName | String | firstName of the passenger |
| private | lastName | String | LastName of the passenger |
| private | phoneNumber | long | phoneNumber of the passenger |

Table 7 Passenger class attributes

**Methods:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Return Type** | **Method Name** | **Parameters** |
| Public | String | **nameOfPerson()** | None |
| Public | void | **detailsOfPassenger()** | None |

Table 8 Passenger class methods

1. Create a parametrized constructor with all attributes.
2. Create getter and setter methods for all the attributes.
3. **nameOfPerson() -** This method does not take any parameters and the return type is String. It will concat the firstName & lastName of the Passenger.
4. **detailsOfPassenger() –** This method does not take any parameters and the return type is void. It prints the Passenger’s details as shown below.

A close-up of a passport

Description automatically generated

**TrainDetails:** This is a concrete class that has the attributes and methods mentioned in Table 9 and Table 10 respectively.

**Attributes**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Attribute** | **Type** | **Attribute Description** |
| private | trainNumber | String | TrainNumber of the passenger |
| private | trains | String | trains of the passenger |
| private | from | String | From location of the passenger |
| private | to | String | To location of the passenger |
| private | seatsAvailable | String | seatsAvailable in the trains |
| private | hoursJourney | String | train Journey hours |
| private | date | LocalDate | Journey Date |
| private | time | LocalTime | Journey Time |

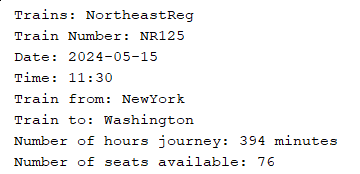
Table 9 TrainDetails class attributes

**Methods:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Return Type** | **Method Name** | **Parameters** |
| Public | void | **viewTrainDetails()** | None |

Table 10 TrainDetails class methods

1. Create a no-arg & parametrized constructor with all attributes.
2. Create getter and setter methods for all the attributes.
3. **viewTrainDetails() -** This method does not take any parameters and the return type is void. It prints the Train Details as shown below.



1. Implement the **Comparable** interface and sort the train details based on time in ascending order.

**TicketDetails:** This is a concrete class that has the attributes and methods mentioned in Table 11 and Table 12 respectively.

**Attributes**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Attribute** | **Type** | **Attribute Description** |
| private | trains | TrainDetails | TrainDetails of the passenger |
| private | passenger | Passenger | passenger |
| private | cabinClass | CabinClass | Cabin class |
| private | discount | double | Discount of the passenger |
| private | foodType | String | Foodtype |

Table 11 TicketDetails class attributes

**Methods:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Return Type** | **Method Name** | **Parameters** |
| public | int | **generatePlatformNumber()** | None |
| public | String | **generateSeatNumber()** | None |
| public | double | **calculateTax()** | None |
| public | double | **calculateAmount()** | None |
| public | void | **displayTicket()** | None |

Table 12 TicketDetails class methods

1. Create a no-arg & parametrized constructor with all attributes.
2. Create getter and setter methods for all the attributes.
3. **generatePlatformNumber()-** This method generates a random integer between 1-9 and returns it.
4. **generateSeatNumber()**- This method generates a random number between 1-36, and a random character between A-F and returns them, for example, 12D.
5. **calculateTax()–** This method does not take any parameters and the return type is double. It calculates the tax for the ticket cost. **Note:** Tax percentage is 7.5%
6. **calculateAmount() –** This method does not take any parameters and the return type is double. This method returns the total cost of the ticket.
7. **displayTicket()-** This method does not take any parameters and the return type is void. This method displays the ticket Details as shown below.
8. Override **equals() && hashcode()** methods for checking whether the data is duplicate or not in the **trains.txt.** Please refer to sample output 3where data is loaded and has duplicates in the train details. So, you must remove duplicates and load unique data. (for example, trains.txt has 9 train details but we found a duplicate that should be removed and shown in the available trains option in the sample output 3)

A close up of a number

Description automatically generated

**TrainsAvailable:** This is a concrete class that has the attributes and methods mentioned in Table 13 and Table 14 respectively.

**Attributes**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Attribute** | **Type** | **Attribute Description** |
| private | trains | **ArrayList<TrainDetails>** | TrainDetails of the passenger |

Table 13 TicketAvailable class attributes

**Methods:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Return Type** | **Method Name** | **Parameters** |
| public | void | **addTrains()** | None |
| public | void | **viewAvailableTrains()** | None |

Table 14 TicketAvailable class methods

1. Create a no-arg constructor that initializes the attribute.
2. Create getter and setter methods for all the Attributes.
3. **addTrains() -** This method does not take any parameters and the return type is void. It adds the passenger train details into the ArrayList trains.
4. **viewAvailableTrains()** **-** This method does not take any parameters and the return type is void. It prints all the available trains as shown below.

A close-up of a computer screen

Description automatically generated

**TrainStatus:** This is a concrete class that has the attributes and methods mentioned in Table 15 and Table 16 respectively.

**Attributes**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Attribute** | **Type** | **Attribute Description** |
| private | from | **String** | From the location of the passenger |
| private | to | **String** | To the location of the passenger |

Table 15 TrainStatus class attributes

**Methods:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Specifier** | **Return Type** | **Method Name** | **Parameters** |
| public | ArrayList<String> | **trainStatus()** | ArrayList<ArrayList<String>> in |

Table 16 TrainStatus class methods

1. Create a parametrized constructor with all attributes.
2. Create getter and setter methods for all the attributes.
3. **trainStatus() -** This method take the ArrayList<ArrayList<String>> as parameter and the return type is ArrayList<String>. This method returns all the stations between the from and to the location of the passenger.
4. Create a **package** called **edu.nwmissouri.spring24.cs44542.secNN.enums** and create the below Java enum in it.

**Discount:** This enum contains discounts values as shown in Table 15.

|  |  |
| --- | --- |
| **Constant** | **Discount Percentage** |
| **STUDENTS** | **13.5** |
| **COUPONS** | **7.0** |
| **HALLOWEEN** | **9.3** |
| **CHRISTMAS** | **6.7** |
| **NONE** | **0** |

Table 15 Discount

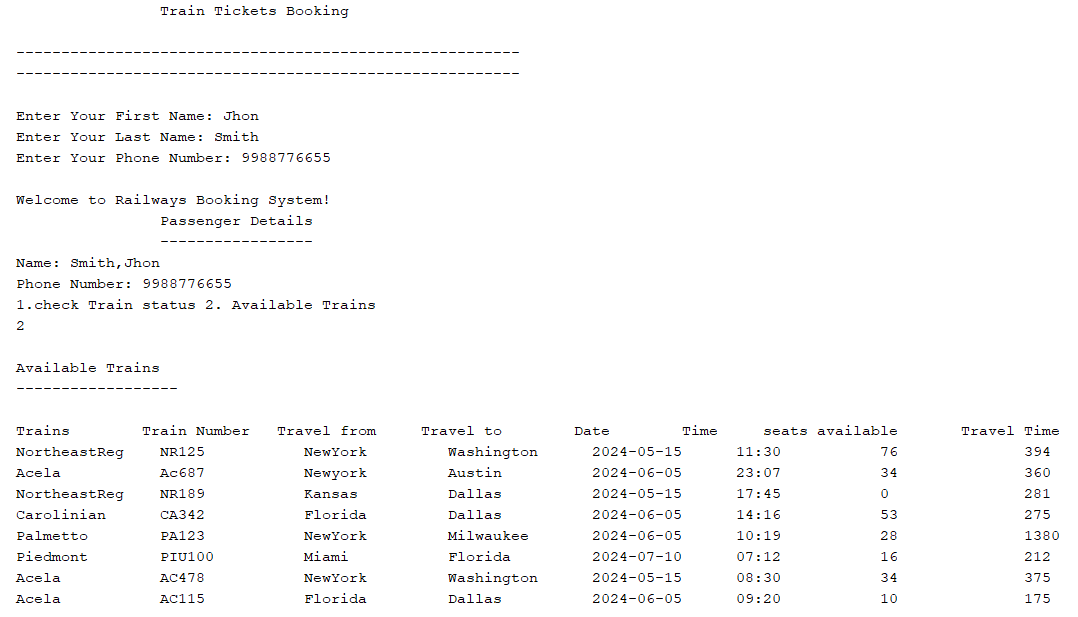
This Enum contains the attributes specified in Table 16. All the attributes of this Enum are **private final**.

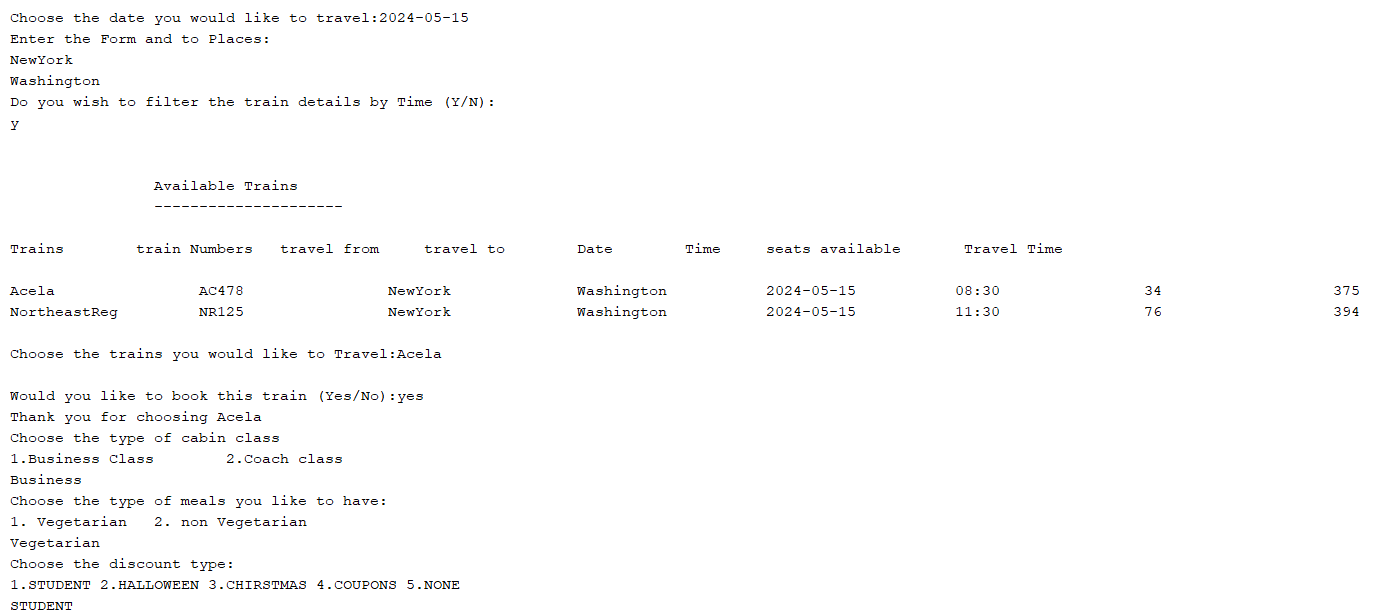
|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| **value** | **double** | Percentage value of the discount |

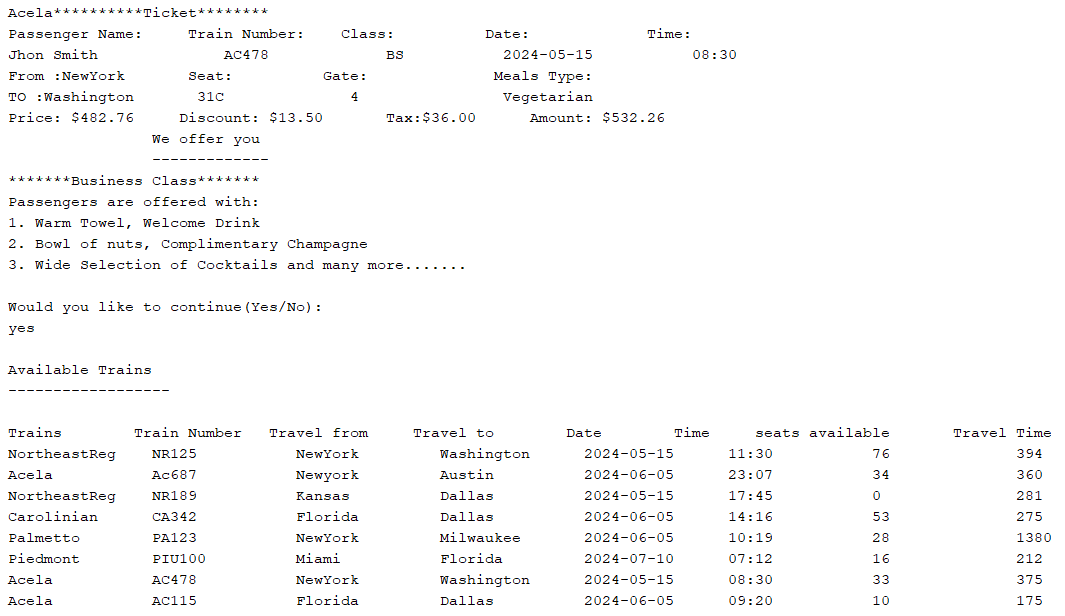
Table 16 Discount Attributes

1. Write a constructor with all the attributes in the same order as given in Table 16
2. Write getter methods for all the attributes given in Table 16.
3. Create a **package** called **edu.nwmissouri.spring24.cs44542.secNN.exceptions** and create the below Exceptions class in it.
4. **TrainNotFoundException:** When a passenger does not find a train to book a ticket this exception is raised.
5. This class extends the **Exception** class.
6. Create required constructors.
7. **NoSeatsAvailableException:** This exception is raised when a passenger tries to get a seat after all available seats are booked.
8. This class extends the **Exception** class.
9. Create required constructors.
10. **NG Testing:** Test all the methods in **TicketDetails**, **TrainsAvailable, and TrainStatus** classes.
11. Right-click on the class and select Tools and Create/Update tests.
12. Select TestNG in Create/Update tests and click Ok.

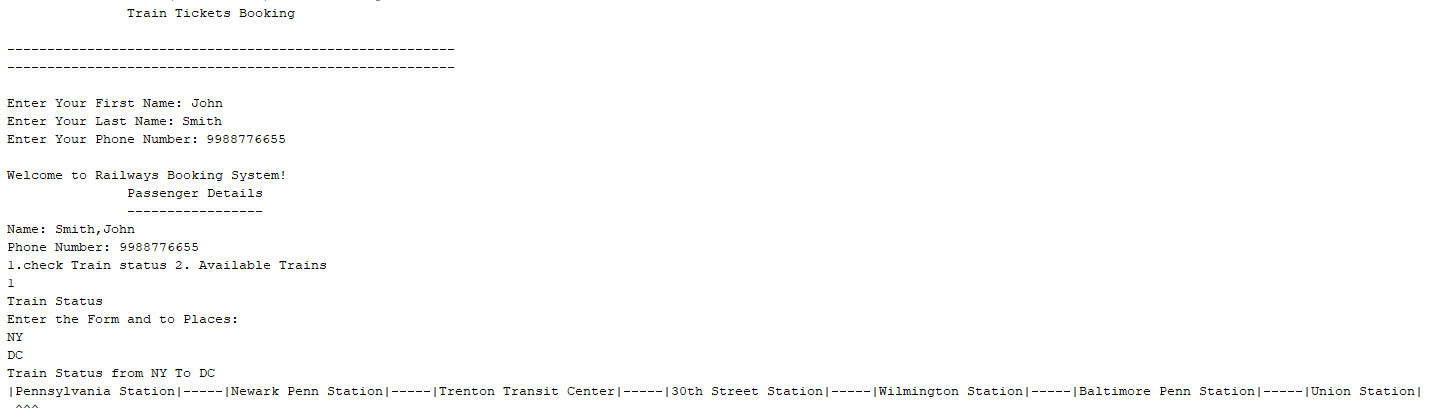
**Sample output 1:**

****





**Sample output 2:**

****

**Note**: Train status from NY To DC in Sample output 2 is shown in the Cap symbol and it is generated randomly.

**Sample output 3: (Data is loaded and Removed Duplicates Train Details)**

**A screenshot of a computer

Description automatically generated**

**Sample Output 4: (NoSeatsAvailableException)**

**A white screen with black text

Description automatically generated**

**Sample Output 5: (TrainNotFoundException)**

**A close up of a document

Description automatically generated**

**Bonus (4 points)**: Set up GitHub Actions to automatically run your unit tests for each commit.

**Submission:** Push your entire Java project to GitHub and submit your private repository link on Canvas as a Website URL before the due date.