**Problem Statement**

You are tasked with developing a basic tourist management system in Java. The system will interact with users to gather their details, validate these details, and generate a tourist ID based on the provided information.

**Classes**

You need to implement the following classes:

1. **UserInterface**:
   * Contains the main method to handle user input and interaction with the TouristManagementSystem.
2. **TouristManagementSystem**:
   * Contains methods to validate tourist details and generate a tourist ID.
3. **InvalidTouristDetailsException**:
   * A custom exception class to handle invalid tourist details.

**Class Details**

**UserInterface.java**

Implement the UserInterface class with the following requirements:

* **public static void main(String[] args)**:
  + Prompt the user to enter their name, age, phone number, and travel mode.
  + Use Scanner to read these inputs.
  + Create an instance of TouristManagementSystem.
  + Validate the details using the validateTouristDetails method.
  + If details are valid, generate a tourist ID using the getTouristId method and display it.
  + Handle exceptions of type InvalidTouristDetailsException and display the error message.

**TouristManagementSystem.java**

Implement the TouristManagementSystem class with the following methods:

* **public boolean validateTouristDetails(String name, int age, long phoneNumber, String travelMode) throws InvalidTouristDetailsException**:
  + Validates the tourist details with the following rules:
    - **Name**: Must contain only letters and spaces.
    - **Age**: Must be between 60 and 99 (inclusive).
    - **Phone Number**: Must be exactly 10 digits long.
    - **Travel Mode**: Must be either "Airway" or "Roadway".
  + Throws InvalidTouristDetailsException with an appropriate message if any detail is invalid.
* **public String getTouristId(String name, int age, long phoneNumber, String travelMode)**:
  + Generates and returns a tourist ID in the following format:
    - The first two characters of the name.
    - The age.
    - The first two digits of the phone number.
    - The first two characters of the travel mode.
  + Returns an appropriate message if the details are invalid.

**InvalidTouristDetailsException.java**

Implement the InvalidTouristDetailsException class with the following constructor:

* **public InvalidTouristDetailsException(String message)**:
  + Inherits from Exception.
  + Takes an error message as a parameter and passes it to the superclass constructor.

**Example**

Here’s an example of how the application should behave:

**Input:**

Enter your name

John Doe

Enter your age

65

Enter your phone number

1234567890

Enter your travel mode

Airway

**Output:**

Tourist details are valid your tourist Id is Jo659012Ai

**Note:**

* Implement the validateTouristDetails method to perform input validation and throw InvalidTouristDetailsException as necessary.
* The getTouristId method should only generate and return the ID if the details are valid.

Top of Form

Bottom of Form

Boiler Plate:

UserInterface.java:  
  
import java.util.Scanner;  
public class UserInterface {  
 public static void main(String[] args) {  
 Scanner sc=new Scanner(System.in);  
 //Fill the code here  
   
   
 }  
}

TouristManagementSystem.java:

public class TouristManagementSystem {  
 public boolean validateTouristDetails(String name, int age, long phoneNumber, String travelMode) throws InvalidTouristDetailsException{  
 //Fill the code here  
 return null;  
 }  
 public String getTouristId(String name, int age, long phoneNumber, String travelMode) {  
   
 //Fill the code here  
 return null;  
 }  
}

InvalidTouristDetailsException.java:  
  
public class InvalidTouristDetailsException extends Exception{  
 public InvalidTouristDetailsException(String message) {  
 super(message);  
 }  
}

**Problem Statement**

You are tasked with implementing a customer information system that manages and processes customer account details for a financial service. The system should validate user inputs and calculate the total amount based on the provided details.

**Classes**

You need to implement the following classes:

1. **UserInterface**:
   * Manages user interaction and coordinates with the CustomerInfoSystem class.
2. **CustomerInfoSystem**:
   * Contains methods to validate customer information and calculate the total amount.
3. **InvalidCustomerInfoException**:
   * A custom exception to handle invalid customer information.

**Class Details**

**UserInterface.java**

Implement the UserInterface class with the following requirements:

* **public static void main(String[] args)**:
  + Prompt the user to enter their account number, name, age, tenure, and monthly deposit.
  + Use Scanner to read these inputs.
  + Create an instance of CustomerInfoSystem.
  + Validate the details using the validateCustomerInfo method.
  + If the details are valid, calculate and display the total amount using the getTotalAmount method.
  + Handle exceptions of type InvalidCustomerInfoException and display the error message.

**CustomerInfoSystem.java**

Implement the CustomerInfoSystem class with the following methods:

* **public boolean validateCustomerInfo(String accountNumber, String name, int age, int tenure, double monthlyDeposit) throws InvalidCustomerInfoException**:
  + Validates the customer details with the following rules:
    - **Account Number**: Must be exactly 10 digits long.
    - **Name**: Must contain only letters and spaces.
    - **Age**: Must be between 18 and 100 (inclusive).
    - **Tenure**: Must be between 12 and 120 months (inclusive).
    - **Monthly Deposit**: Must be a non-negative number.
  + Throws InvalidCustomerInfoException with an appropriate message if any detail is invalid.
* **public double getTotalAmount(int age, int tenure, double monthlyDeposit)**:
  + Calculates and returns the total amount based on the age and tenure:
    - For customers aged between 18 and 59:
      * Interest Rate: 2% of the monthly deposit.
    - For customers aged 60 and above:
      * Interest Rate: 5% of the monthly deposit.
  + The total amount is calculated as:
    - Total Amount = Monthly Deposit + (Interest Rate \* Tenure)

**InvalidCustomerInfoException.java**

Implement the InvalidCustomerInfoException class with the following constructor:

* **public InvalidCustomerInfoException(String message)**:
  + Inherits from Exception.
  + Takes an error message as a parameter and passes it to the superclass constructor.

**Example**

Here’s an example of how the application should behave:

**Input:**

Enter the account number

1234567890

Enter the name

John Doe

Enter the age

30

Enter the tenure

24

Enter the monthly deposit

1000

**Output:**

Total amount is 22000.0

**Notes:**

* Implement the validateCustomerInfo method to perform input validation and throw InvalidCustomerInfoException as necessary.
* The getTotalAmount method should calculate the total amount based on the given rules and return it.

Boiler Code:  
  
UserInterface.java:  
  
import java.util.Scanner;  
  
public class UserInterface {  
 public static void main(String[] args) {  
 Scanner sc=new Scanner(System.in);  
 //Fill the code here  
   
 }  
}

InvalidCustomerInfoException.java:  
  
public class InvalidCustomerInfoException extends Exception{  
 public InvalidCustomerInfoException(String message) {  
 super(message);  
 }  
}  
  
CustomerInfoSystem.java:

public class CustomerInfoSystem {  
 public boolean validateCustomerInfo(String accountNumber, String name, int age, int tenure, double monthlyDeposit) throws InvalidCustomerInfoException{  
 //Fill the code here  
   
 return null;  
   
   
 }  
   
 public double getTotalAmount(int age, int tenure, double monthlyDeposit) {  
 //Fill the code here  
   
 return null;  
 }  
}

**Problem Statement**

You are tasked with creating a vehicle registration validation system. This system will validate the details of vehicle and driver licenses based on specific formats and return appropriate messages if the details are valid or throw an exception if they are not.

**Classes**

You need to implement the following classes:

1. **UserInterface**:
   * Handles user input and interacts with the VehicleRegistrationSystem.
2. **VehicleRegistrationSystem**:
   * Contains methods to validate the details of vehicle and driver licenses.
3. **InvalidVehicleDetailsException**:
   * A custom exception for handling invalid vehicle details.

**Class Details**

**UserInterface.java**

Implement the UserInterface class with the following requirements:

* **public static void main(String[] args) throws InvalidVehicleDetailsException**:
  + Prompt the user to enter:
    - Registration ID
    - Vehicle License Number
    - Driver License Number
  + Use Scanner to read these inputs.
  + Create an instance of VehicleRegistrationSystem.
  + Validate the details using the validateFleetDetails method.
  + Display the validation result or handle exceptions of type InvalidVehicleDetailsException.

**VehicleRegistrationSystem.java**

Implement the VehicleRegistrationSystem class with the following method:

* **public String validateFleetDetails(String registrationId, String vehicleLicenseNumber, String driverLicenseNumber) throws InvalidVehicleDetailsException**:
  + Validate the details with the following rules:
    - **Registration ID**: Must match the format VR-#### where #### is a 4-digit number.
    - **Vehicle License Number**: Must match the format LL00LL0000 where LL are uppercase letters, 00 are digits, and 0000 are digits.
    - **Driver License Number**: Must match the format LL0000000000000 where LL are uppercase letters and 0000000000000 is a 13-digit number.
  + If any detail does not meet these criteria, throw an InvalidVehicleDetailsException with an appropriate message.
  + If all details are valid, return a message indicating that the details can be added to the system.

**InvalidVehicleDetailsException.java**

Implement the InvalidVehicleDetailsException class with the following constructor:

* **public InvalidVehicleDetailsException(String message)**:
  + Inherits from Exception.
  + Takes an error message as a parameter and passes it to the superclass constructor.

**Example**

Here’s an example of how the application should behave:

**Input:**

Enter Registration ID:

VR-1234

Enter Vehicle License Number:

AB12CD3456

Enter Driver License Number:

AB1234567890123

**Output:**

Details for Registration ID:VR-1234 are valid and can be added to the system

**Notes:**

* Implement the validateFleetDetails method to check the validity of each input using regular expressions.
* Ensure that the InvalidVehicleDetailsException is thrown with the correct error message if any detail is invalid.

Boiler Plate:  
  
InvalidVehicleDetailsException.java:

public class InvalidVehicleDetailsException extends Exception {  
 public InvalidVehicleDetailsException(String message) {  
 super(message);  
 }  
}

UserInterface.java

import java.util.Scanner;  
  
public class UserInterface {  
 public static void main(String[] args) throws InvalidVehicleDetailsException {  
 Scanner sc = new Scanner(System.in);  
  
 // Fill the code here  
   
 }  
}  
  
VehicleRegistrationSystem.java  
public class VehicleRegistrationSystem {  
  
 public String validateFleetDetails(String registrationId, String vehicleLicenseNumber, String driverLicenseNumber) throws InvalidVehicleDetailsException {  
  
 // Fill the code here  
 }  
}