

INSTRUCTION:

Design and implement a Java program for the following description of an application. You must apply *Facade* Design Pattern.

Let us build an application that:

- Accepts customer details (account, address and credit card details)
- Validates the input data
- Saves the input data to appropriate data files

The three classes — `Account`, `Address` and `CreditCard` are available in the system, each with its own methods for validating and saving the respective data. These 3 classes work together, as part of a subsystem, providing the features of an online customer.

Account	Address
<code>firstName:String</code> <code>lastName:String</code>	<code>address:String</code> <code>city:String</code> <code>state:String</code>
<code>isValid():boolean</code> <code>save():boolean</code> <code>getFirstName():String</code> <code>getLastName():String</code>	<code>isValid():boolean</code> <code>save():boolean</code> <code>getAddress():String</code> <code>getState():String</code>

CreditCard
<code>cardType:String</code> <code>cardNumber:String</code> <code>cardExpDate:String</code>
<code>isValid():String</code> <code>save():String</code> <code>getCardType():String</code> <code>getCardNumber():String</code> <code>getCardExpDate():String</code>

```

public class Account {
    String firstName;
    String lastName;
    final String ACCOUNT_DATA_FILE = "AccountData.txt";

    public Account(String fname, String lname) {
        firstName = fname;
        lastName = lname;
    }

    public boolean isValid() {
        /*
         * Let's go with simpler validation
         * here to keep the example simpler.
         */

        if (getLastName().trim().length() < 2)
            return false;
        return true;
    }

    public boolean save() {
        FileUtility futil = new FileUtility();
        String dataLine = getLastName() + "," + getFirstName();
        return futil.writeToFile(ACCOUNT_DATA_FILE, dataLine);
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
}

```

```

public Address(String add, String cty, String st) {
    address = add;
    city = cty;
    state = st;
}

public boolean isValid() {
    /*
     * The address validation algorithm could be complex in real-world
     * applications. Let's go with simpler validation here to keep the example simpler.
     */
    if (getState().trim().length() < 2)
        return false;
    return true;
}

public boolean save() {
    FileUtility futil = new FileUtility();
    String dataLine = getAddress() + "," + getCity() + "," + getState();
    return futil.writeToFile(ADDRESS_DATA_FILE, dataLine);
}

public String getAddress() {
    return address;
}

public String getCity() {
    return city;
}

public String getState() {
    return state;
}
}

```

```

public class CreditCard {
    String cardType;
    String cardNumber;
    String cardExpDate;
    final String CC_DATA_FILE = "CC.txt";
    public static final String VISA = "Visa";
    public static final String MASTER = "Master";

    public CreditCard(String ccType, String ccNumber, String ccExpDate) {
        cardType = ccType;
        cardNumber = ccNumber;
        cardExpDate = ccExpDate;
    }

    public boolean isValid() {
        /*
         * Let's go with simpler validation here to keep the example simpler.
         */
        if (getCardType().equals(VISA) || getCardType().equals(MASTER)) {
            return (getCardNumber().trim().length() == 16);
        }
        return false;
    }

    public boolean save() {
        FileUtility futil = new FileUtility();
        String dataLine = getCardType() + "," + getCardNumber() + "," + getCardExpDate();
        return futil.writeToFile(CC_DATA_FILE, dataLine);
    }

    public String getCardType() {
        return cardType;
    }

    public String getCardNumber() {
        return cardNumber;
    }

    public String getCardExpDate() {
        return cardExpDate;
    }
}

```

A client `AccountManager` class displays the user interface to a user to input the customer data. In order to validate and save the input data, the client `AccountManager` would:

- Create `Account`, `Address` and `CreditCard` objects
- Validate the input data using these objects
- Save the input data using these objects

However, this design creates high coupling between the client `AccountManager` and the subsystem components (`Address`, `Account` and `CreditCard` classes in this case).

1. The **Façade** pattern can be used to achieve low coupling between the client `AccountManager` and the subsystem components. Define a **Façade** class `CustomerFacade` that offers a higher level, simplified interface to the subsystem consisting of customer data processing classes (`Address`, `Account` and `CreditCard`). Draw a UML class diagram to show your design for the application.

2. Implement the application in Java based on your design above.

Note: The `FileUtility` class provides the following method to write a message to a file.

```
public boolean writeToFile(String filename, String msg)
```

3. Write the statements in `AccountManager.java` to test your implementation. The output should be as shown below:

```
First customer:
First name = John, Last name = Smith
Address = 101 Jalan Bukit, City = Shah Alam, State = Selangor
Card type = Visa, Card number = 1111222233334444, Card expiry date = 01/09/2020
Valid FirstName/LastName
Valid Address/City/State
Valid CreditCard
==>Valid Customer Data: Data Saved Successfully

Second customer:
First name = Vijaya, Last name = K
Address = 1 Jalan University, City = Kuala Lumpur, State = Wilayah Persekutuan
Card type = Master, Card number = 9999888877776666, Card expiry date = 01/01/2022
Invalid FirstName/LastName
Valid Address/City/State
Valid CreditCard
==>Invalid Customer Data: Data Could Not Be Saved

Third customer:
First name = Aryati, Last name = Ahmad
Address = 35 Wisma Jaya, City = Petaling Jaya, State = Selangor
Card type = Master, Card number = 555566667777, Card expiry date = 01/05/2023
Valid FirstName/LastName
Valid Address/City/State
Invalid CreditCard Info
==>Invalid Customer Data: Data Could Not Be Saved
```