| | |
|---|---|
| EX NO:01<br><br>DATE: | **Identify a software system that made to be developed** |

**Aim:**

To identify a software system that needs to be developed - Credit Card Processing System

**Introduction:**

In today's digital age, seamless and secure credit card processing is fundamental to a thriving business. A robust Credit Card Processing System is crucial for authorizing transactions, ensuring data security, and streamlining financial operations. This document outlines the key features and architecture of such a system

**Features of Credit Card Processing System:**

**Transaction Processing:**

- Support various card types (e.g., Visa, Mastercard) and payment methods (online, in-person).
- Authorize transactions in real-time using secure connections to payment networks.
- Integrate with point-of-sale (POS) systems for in-store transactions.

**Fraud Detection and Prevention:**

- Implement advanced fraud detection algorithms to identify suspicious activity.
- Utilize tokenization and encryption to protect sensitive cardholder data.
- Comply with Payment Card Industry Data Security Standard (PCI DSS) for enhanced security.

## Reporting and Reconciliation:

- Generate detailed reports on transactions, fees, and settlements.
- Facilitate reconciliation with bank statements for accurate accounting.
- Offer customizable reports for business insights and trend analysis.

## Customer Management:

- Provide a user-friendly interface for customers to manage accounts and view transactions.
- Integrate with customer relationship management (CRM) systems for improved service.
- Offer secure self-service options for account updates and dispute resolution.

## Administration and Management:

- Enable administrators to manage user accounts and access control.
- Monitor system performance and troubleshoot any issues.
- Generate system logs for auditing and compliance purposes.

## Mobile Payments and Integration:

- Enable secure mobile wallet payments (e.g., Apple Pay, Google Pay) for a frictionless customer experience.
- Integrate with mobile apps for in-app purchases and convenient payment options.
- Offer QR code or contactless payment solutions for faster checkout experiences.

## Result:

Thus the software system that is need to be developed for Credit Card Processing System was executed successfully.

| EX NO:02<br><br>DATE: | **Document the software requirements specification(SRS) for the credit card processing system** |
|---|---|

## Aim:

To document the software requirements specification(SRS) for the credit card processing system

## Problem statement:

Design and implement an efficient credit card creation process that optimizes both user experience and security measures, addressing challenges such as identity verification, fraud prevention, and seamless integration with banking systems, to ensure a streamlined and secure issuance of credit cards while minimizing potential risks and enhancing customer satisfaction.

The Credit Card Processing System is a software application designed to securely handle transactions made using credit cards. The system ensures the confidentiality, integrity, and availability of sensitive financial information while facilitating the authorization, processing, and settlement of credit card transactions.

## Table of Contents:

## 1.Introduction:

Credit card processing is interaction of the customer to the credit card processor for buying a new card of the payment of bills.The processor allows getting a new card and also to get the details of the card and makes the copy of the purchase made.

### 1.1 Purpose

The credit card makes shopping easier and the customer or purchaser need not carry cash along with him/her.It allows the customer to buy his/her requirements without a need to keep **track of his/her cash in hand especially in case of emergencies.**

### 1.2 Document Conventions

This document is written in the following style

| | | |
|---|---|---|
| Font | : | Times New Roman |
| Headings | : | 16 size |
| Sub-Headings | | :14 size |
| Description | : | 12 size |

### 1.3 Intended Audience and Reading Suggestions

This document is intended to be read by the developers and the end users. This is a technical document and the terms can be easily understood by the users.

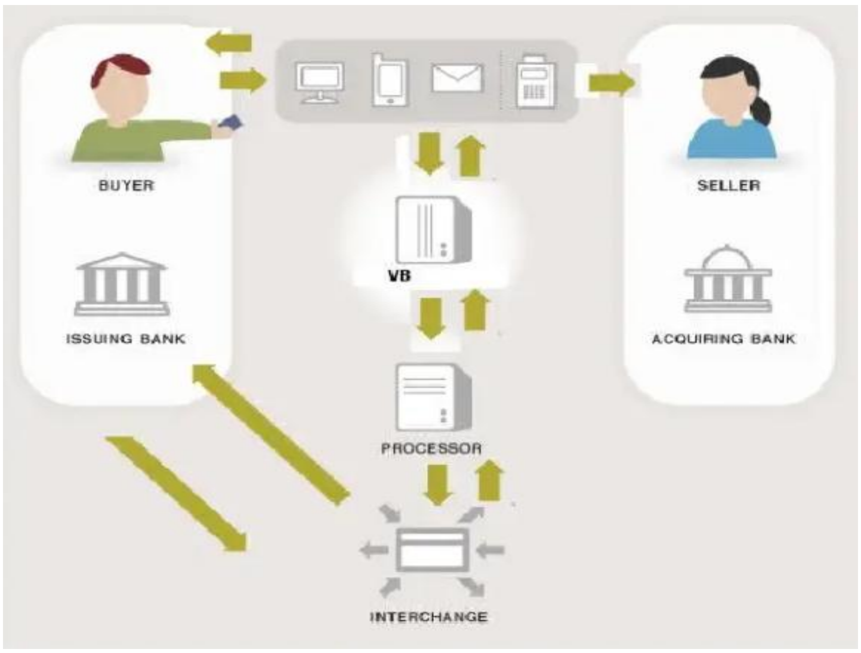
### 1.4 Product scope

The credit card processing is usually done with a help of the swiper which scans all the details of the card. after each purchase, the details are recorded and document is made by the credit card issuer. Accepting credit card is an integral part of business today for merchants who want to be competitive in their market and grow their business to its greatest potential. Software and Gateway processing helps reduce fraud losses, saves you time and money, and provides powerful features and performance, including detailed transaction records and reports. Swiping credit card ensures lower rates, resulting in potential savings of hundreds of dollars each month for your business.

## 2. Overall Description:

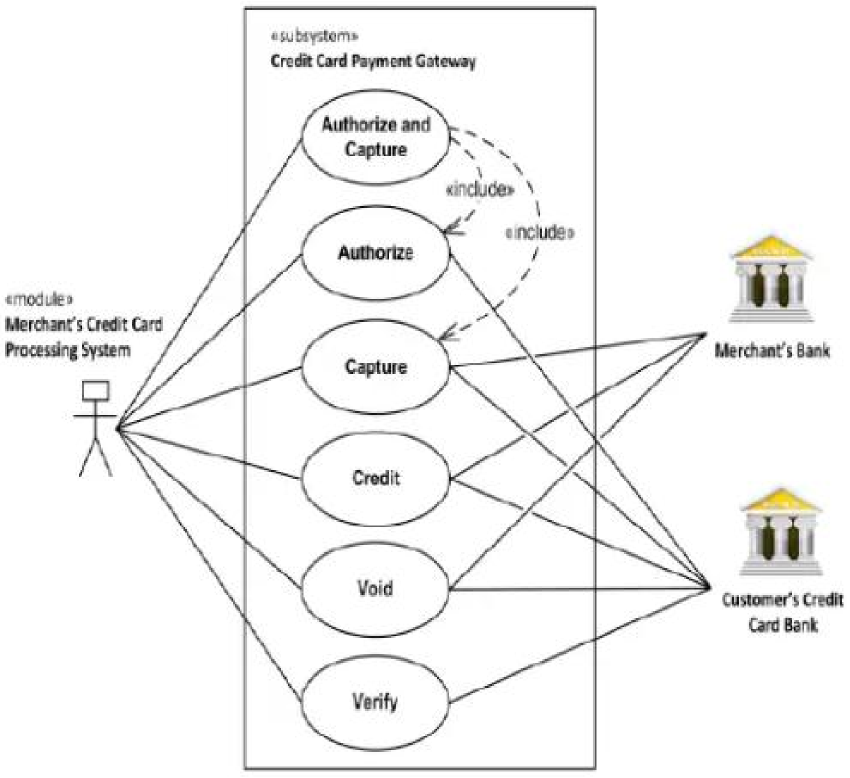
### 2.1 Product prespective

This project is a self -contained project for easy transaction using credit card.



### 2.2 Product Features:

This front end requires details that are needed to buy a new card or selecting the options for paying the bill to the credit card. In the back end, The credit card issuer records and deals the all the transaction made by the purchaser and send the statement to him/her.



## Credit Card Processing System(Credit card Payment  Gateway)

Credit card Payment Gateway is a subject, i.e system under design or consideration.

## Merchant's Credit Card Processing System

Primary actor of the system is the Merchant's  Credit Card Processing System. The merchant submits a credit  card transaction request to the credit card payment gateway on behalf of a customer. Bank which issued customer's credit card is actor which could approve or reject the transaction. If transaction is approved, funds will be transferred to merchant bank account.

## Customer's Credit Card Bank and Merchant's Bank

The requested amount of money should be first authorized by Customer's Credit Card Bank, and if approved, is further submitted for settlement.

During the settlement funds approved for the credit card transaction are deposited into the Merchant's  Bank account.

## Authorize and Capture

Authorize and Capture use case is the most common type of credit card transaction.

In this case usually if no further action is taken within some number of days, the authorization expires. Merchants can submit this request is they want to verify the availability of funds on the customer's credit card, If items is not currently in stock, or if merchant wants to review orders before shipping.

## Capture

Capture (request to capture funds that were previously authorized use case describes several scenarios when merchant needs to complete some previously authorized transaction-either submitted through the payment gateway or requested without using the system, e.g: using voice authorization.

**Credit**

Credit use case describes situations when customers should receive a refund for a transaction that was either successfully processed and settled through the system or for some transaction that was not originally submitted through the payment gateway.

**Void**

Void use case describes cases when it is needed to cancel one or several related transactions that were not yet settled. If possible, the transactions will not be sent for settlement. If  the void transaction fails, the original transaction is likely already settled.

**Verify**

Verify use case describes zero or small amount verification transactions which could also include verification of some client's data such as address.

## 2.3 User Classes and Characteristics

The credit card transaction is used by the clients to do the crediting features that are available in bank and do the payment back. the account has to be updated with the balance every time when the crediting and the payback are done.

## 2.4 Operating Environment

This tool will function on MICROSOFT WINDOWS based operating system

| | | |
|---|---|---|
| Programming Language | : | VB |
| Database | : | SQL SERVER 2008 R2 |

## 2.5 Design and Implementation constraints

This will be an opensource project and thus will not be able to operate and function on most propriety operating systems due to the lack of support systems provided by these systems. In the current version reports can be generated only in SQL SERVER 2008 R32 dbms  and source database are also restricted to few sources mentioned above. furthermore it will also require VB language compilers for further enhancing of features.

## 2.6 User Documentation

The user documentation can be found in this SRS.

## 2.7 Assumptions and Dependencies

We assure that the extra documentation beyond this SRS would not be necessary in order for the user to utilize this project.

## 3.System features:

## 3.1 Trainer

### 3.1.1 Description and priority

The process of swiper is to scan the information in the magnetic tape and inputs the PIN number of the credit card to make a successful transaction.

### 3.1.2 Stimulus/Response Sequences

When the card is swiped, a remote connection is made with the issuer and the amount is retrieved.

### 3.1.3 Functional Requirements

REQ 1: The PIN number must be genuine

REQ 2: The credit card holder must pay his debts genuinely

## 4.External Interface Requirements:

### 4.1 User interfaces

This tool will come along with a basic user interface tool with GUI along with simple and easy to navigate icons thus avoiding any complexity which may not be understandable for novoice users. The interfacing will contain arrow marks, text boxes and some different shaped boxes reduce complexity.

### 4.2 Hardware interfaces

| | | |
|---|---|---|
| Processor | : | Intel Pentium IV 1.83 |
| Memory | : | 256 MB RAM |
| Hard disk drive | : | 40 GB |

Credit card processing system requires a Swiper to be connected to the computer.

### 4.3 Software Interfaces

| | | |
|---|---|---|
| Operating system | : | MICROSOFT WINDOWS |
| Programming Language | : | VB |
| DBMS | : | MICROSOFT ACCESS 2007 |

## 5.Other non-functional requirements:

### 5.1 Performance requirements

The swiper should work effectively during the transaction for good maintainence of the system.

### 5.2 Safety requirements

The card issuer must update the record of the card holders prevent fraudulent actions and other errors that may occur during transactions.

### 5.3 Security requirements

The card holder must not give details to others about the PIN number of the credit card.

**5.4 Software quality attributes**

The code will be free to in its accuracy, reusability to modify the code, flexibility in its usage and more over simple and informative.

**5.5 Business rules**

There are no such business rules. the tool can be used by any users.

**6.Other requirements:**

There are no other requirement

**Result:**

Thus the documentation of the software requirements specification(SRS) for the credit card processing system was executed successfully.

| | |
|---|---|
| EX NO:03<br><br>DATE: | **Identify use cases and develop the use case model** |

**Aim:**

To identify use cases and develop the use case model for credit card processing system.

## Introduction of use case diagram:

Use case diagram are graphical representations that depict the interactions between users (actors) and a system to achieve specific goals or tasks. They are an essential tool in software development for capturing and communicating system requirements in a clear and visual manner.

## Purpose:

**Requirement Analysis:** Use case diagrams help stakeholders, including developers, designers, and clients, understand the system's functionality from a user's prespective. They provide a high-level overview of the system's features and interactions.

**Communications:** Use case Diagrams facilitate communication between stakeholders by providing a common visual language to discuss system requirement and functionalities. They help ensure that everyone involved in the project has a shared understanding of the systems scope and objectives.

**Design validation:** Use case diagrams assist in validating the system design by identifying potential gaps inconsistency or missing functionalities. They allow stakeholders review and refine the system requirements before proceeding with the implementation phase.

## Components:

**Actors:** Actors represent entities example user’s system that interact with the system to accomplish specific goal actors or depicted as stick figures and are connected to use causes by associate

**Use cases:** Use case represent the specific functionalities or tasks that the system performs to fulfill user goals each use case describes a sequence of interactions between the system and actors to achieve a particular outcome. Use case or depicted as ovals within the diagrams

**Relationship:** Relationships between actors and use case are depicted using association lines. The relationships indicate the interactions between actors and the system’s to accomplish various tasks. There are different types of relationships including include extend and generalization.

## Benefits:

**Clarity:** Use case diagrams provide a visual representation of system requirements making it easier for stakeholders. To understand and visualize the systems behaviors and functionality.

**Scope definition:** Use case diagrams help define the scope of the system by identifying the primary actors and their interactions with the system. They provide a clear boundary of what functionalities the system should include.

**Requirements prioritization:** Use case diagrams allow stakeholders to prioritize a system based on goals and objectives by identifying critical use case stakeholders can focus on implementing essentials functionalities first.

**Change management:** Use case diagrams facilitate change management by providing a structured framework for evaluating and incorporating new requirement or modification to existing functionalities. They help assess the impact of the overall system.

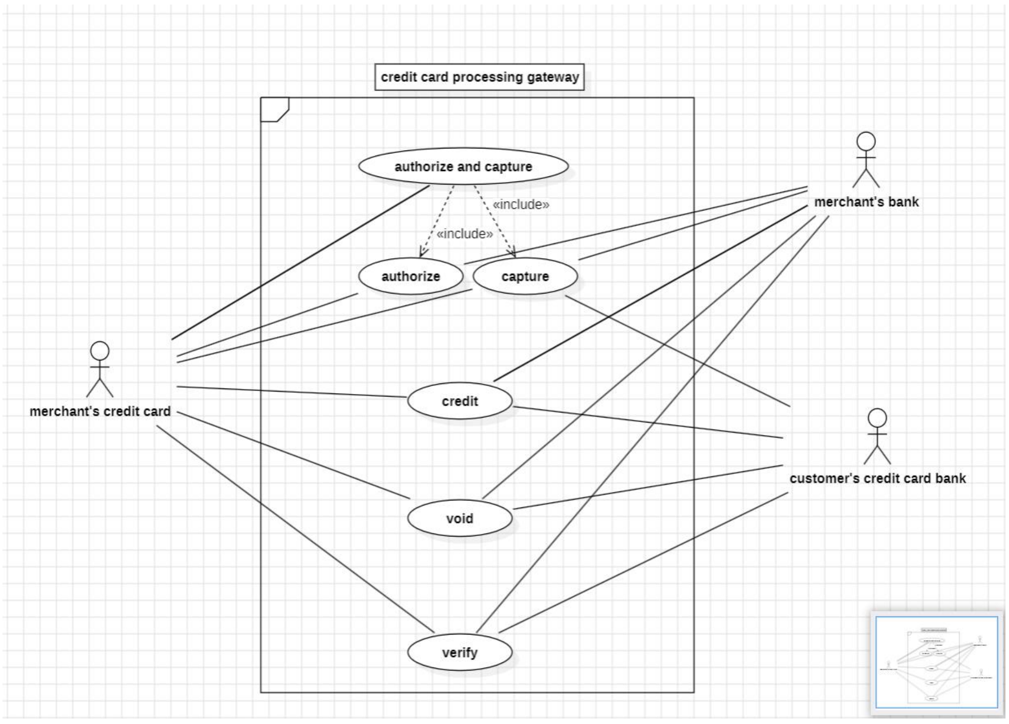
## Use case diagram:

### Credit Card Processing System(Credit card Payment Gateway)

Credit card Payment Gateway is a subject, i.e system under design or consideration.

### Merchant's Credit Card Processing System

Primary actor of the system is the Merchant's Credit Card Processing System. The merchant submits a credit card transaction request to the credit card payment gateway on behalf of a customer. Bank which issued customer's credit card is actor which could approve or reject the transaction. If transaction is approved, funds will be transferred to merchant bank account.



**Customer’s Credit Card Bank and Merchant’s Bank**

The requested amount of money should be first authorized by Customer’s Credit Card Bank, and if approved, is further submitted for settlement.

During the settlement funds approved for the credit card transaction are deposited into the Merchant’s  Bank account.

**Authorize and Capture**

Authorize and Capture use case is the most common type of credit card transaction.

In this case usually if no further action is taken within some number of days, the authorization expires. Merchants can submit this request is they want to verify the availability of funds on the customer’s credit card, If items is not currently in stock, or if merchant wants to review orders before shipping.

**Capture**

Capture request to capture funds that were previously authorized use case describes several scenarios when merchant needs to complete some previously authorized transaction-either submitted through the payment

gateway or requested without using the system, e.g: using voice authorization.

**Credit**

Credit use case describes situations when customers should receive a refund for a transaction that was either successfully processed and settled through the system or for some transaction that was not originally submitted through the payment gateway.

**Void**

Void use case describes cases when it is needed to cancel one or several related transactions that were not yet settled. If possible, the transactions will not be sent for settlement. If the void transaction fails, the original transaction is likely already settled.

**Verify**

Verify use case describes zero or small amount verification transactions which could also include verification of some client's data such as address.

## Result:

Thus the use case diagram for credit card processing system is implemented and executed successfully.

| | |
|---|---|
| EX NO:04<br><br>DATE: | **Identify the Conceptual classes and Develop a Domain model and Derive Class diagram for Credit Card Processing system.** |

## Aim:

To Identify the conceptual classes and develop domain model and also device a class diagram from that

## Conceptual classes:

In the credit card processing system several conceptual classes can be identified to represent the various entities and functionalities of the system. here are some key conceptual classes for the client management system

**Customer:** Represents customer of the system who request for the credit card to merchant's bank.

**Verification_Officer:** Represents the verification officer who validate the application of customer for eligibility of credit card processing.

**Bank:** Represents the bank which is responsible for the allocation of the credit card or cancelling the credit card.

**Bank branch:** Represents the bank branch which is responsible for credit and debit a payment.

**Credit card:** Represents the credit card which is secured by a secret pin and contains expiry date, limit, type.

**Application:** Represents the application of the customer which contains the data of the customer to request for the credit card processing.

## Domain model for Credit card processing system:

**Customer:**

**Attribute:**

- cus_id
- cus_name
- address
- phone_no

**Application:**

**Attribute:**

- application_no
- type
- date&time

**Verification_officer:**

**Attribute:**

- verification_type
- name

**Behaviours:**

- Verifyuserdata()

**Bankbranch:**

**Attribute:**

- Name
- address

**Behaviours:**

- allocatecreditcard()
- cancelcreditcard()

**Bank:**

**Attribute:**

- Name
- address

**Behaviours:**

- allocatecreditcard()

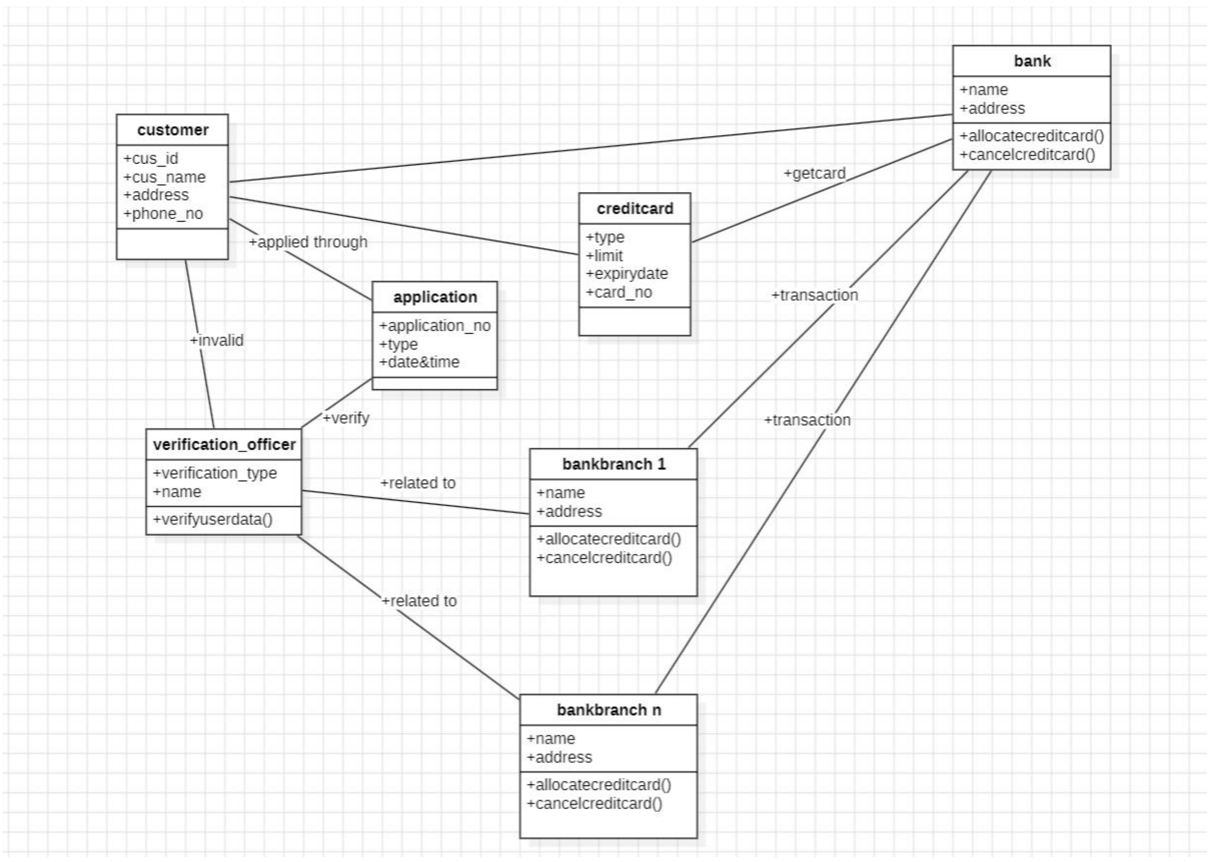
- cancelcreditcard()

**Creditcard:**

**Attributes:**

- type
- limit
- expirydate
- card_no

## Class Diagram for Credit card processing system:



## Result:

Thus the conceptual classes, domain model and class diagram for credit card processing system was identified and developed successfully.

| EX NO:05<br><br>DATE: | **Using the identified scenarios. find the interaction between objects and represent them using UML sequences and collaboration diagram.** |
|---|---|

## Aim:

Using the identified scenarios. To find the interaction between objects and represent them using UML sequences and collaboration diagram.

## Introduction of interaction diagram:

Interaction diagrams are graphical representations used in software engineering and systems design to illustrate how different components or objects within a system interact with each other to achieve certain functionalities or behaviours. These diagrams are essential for understanding the dynamic aspects of the system, showing the flow of messages, data, or control between various elements.

There are two main types of interaction diagrams commonly used in software development.

Sequence diagram

- Communication or collaboration diagram
- Sequence Diagram:

Sequence diagrams depict interaction between different objects or components in a chronological sequence. They show the order in which messages or exchanged between objects over time.

This helps developers visualize the run time behaviour of the system including how objects collaborate to accomplish specific tasks.

In a sequence diagram several components are used to represent different elements and interaction within a system. here are the main components typically found in sequence diagram.

## Sequence diagram:

### User:

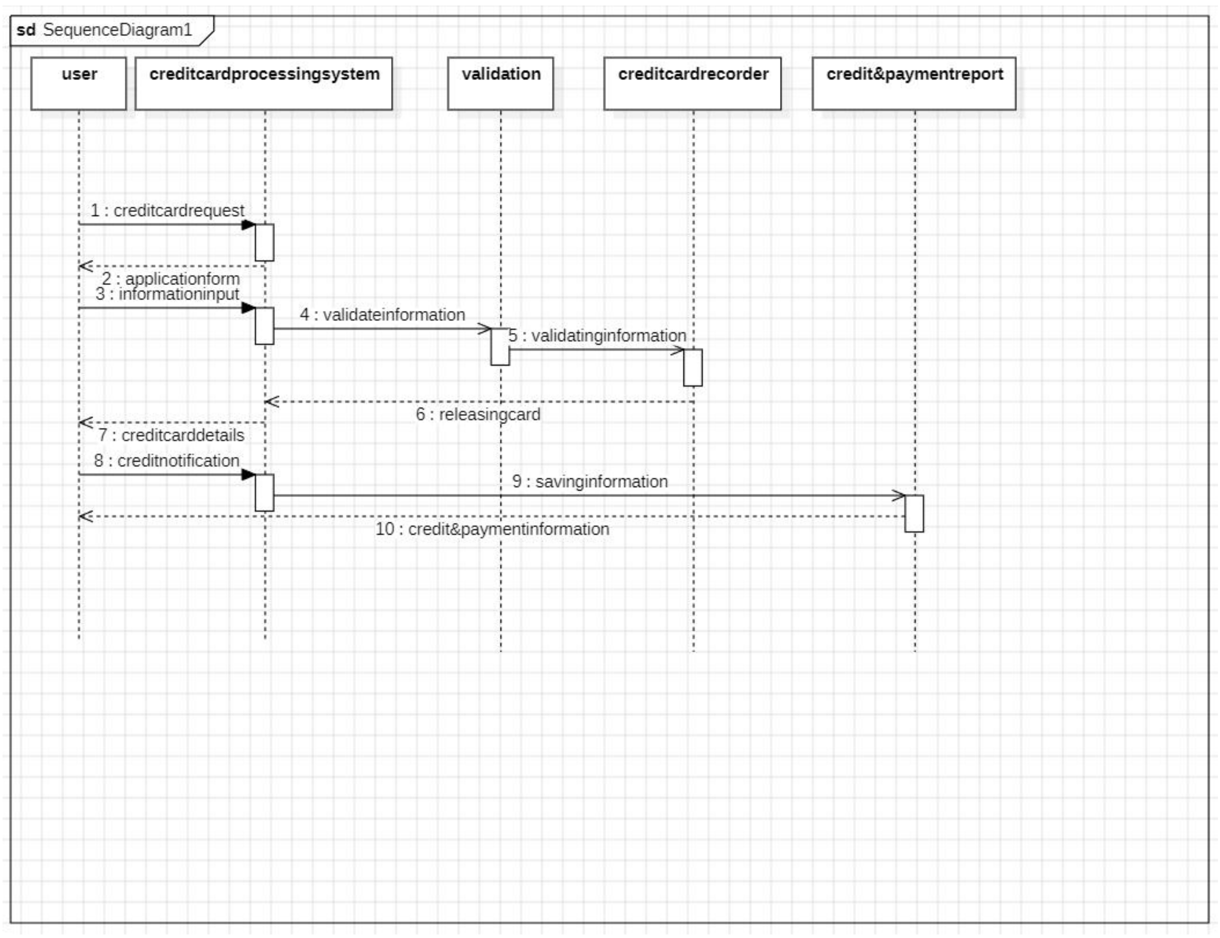
User represents the customer who request / apply for credit card. The messages done

by the user are request message, information input, credit notifications.

### Credit Card Processing System:

It represents credit card processing which handles the application form, credit card details and compute the credit notification.

### Sequence Diagram for credit card processing system:



**Validation:**

It represents the validation_officer who validate the application information whether the customer is eligible or not.

**Credit Card Recorder**:

It represents the recorder or database which contains the details of the customer and credit card.

**Credit & Payment Report:**

It represents the report which sends the credit and debit payment data to the customer.

## Collaboration diagram:

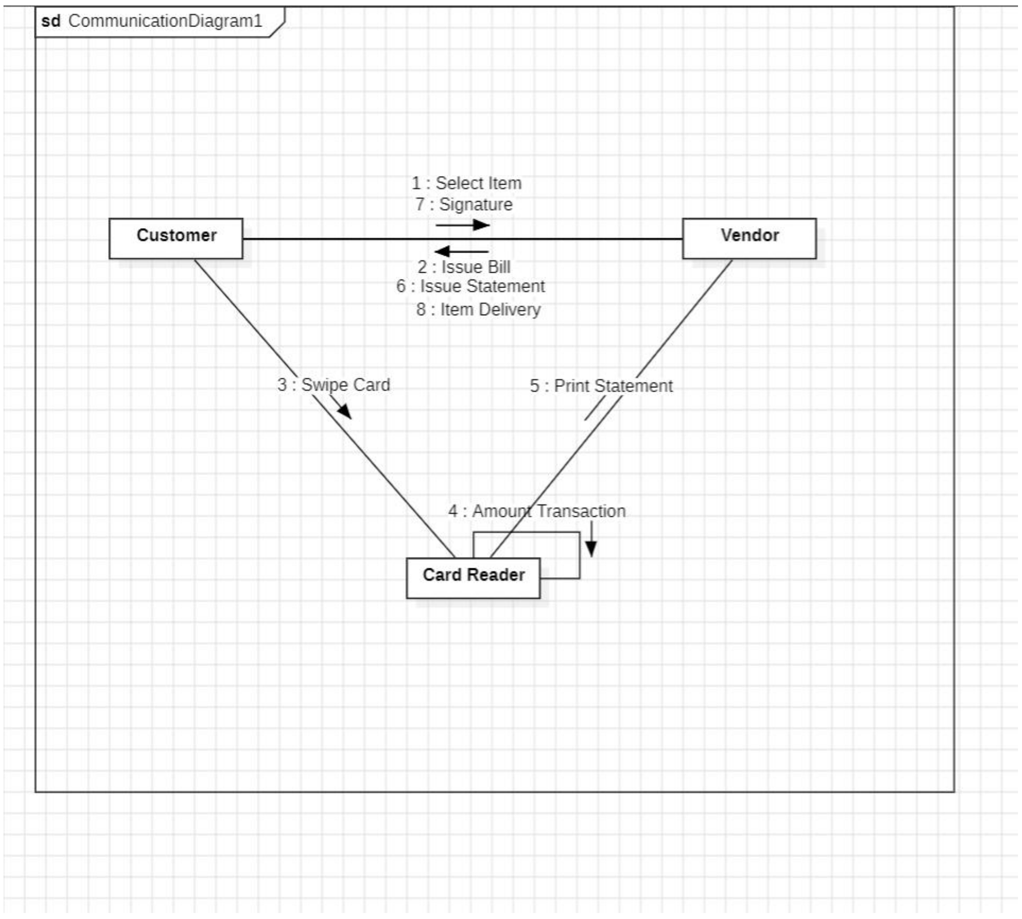
Communication diagram focus on illustrating the relationships and interactions between objects or components within a system.

Unlike sequence diagram communication diagrams emphasize the structural organisation of objects and their associations rather than the precise sequence of messages exchanged.

They provide a more holistic view of the system's architecture and the relationships between the components

In a communication diagram ,several components are used to represent different elements and interactions within a system. Here are the main components typically found in a communication diagram.

## Collaboration diagram for Credit Card Processing System:



In this diagram there is sequence of ordered relationship should performing in the credit card processing , then customer will performed a selecting item, putting signature and deliver the item , vendor should perform the swap the card , issue the statement and card reader should perform amount transaction and print the balance statement

## Result:

Thus the interaction diagram such as sequence diagram and communication diagram for credit card processing system was developed successfully.

| | |
|---|---|
| EX NO:06<br><br>DATE: | **Draw the relevant state chart and activity diagram for credit card processing system** |

**Aim:**

To draw the relevant state chart and activity diagram for credit card processing system.

## Activity Diagram:

Activity Diagrams are graphical representations used in UML to model the workflow or flow of control within a system. They illustrate the sequence of activities or actions that need to be performed to achieve a specific goal or complete a process. Here are the main components typically found in an activity diagram.

**Initial Node:**

An initial node represents the starting point of the activity diagram. It indicates where the process begins and is depicted as a solid-filled circle.

**Activity Nodes:**

Activity nodes represents specific actions or tasks to be performed within the system. They can represent both simple and complex activities. Activity nodes are depicted as rounded rectangles with the activity name written inside.

**Control Flow Arrows:**

Control Flow Arrows depict the flow of control or sequence of activities within the diagram. They connect activity nodes and indicate the order in which activities are performed. Control flow arrows are represented by arrows pointing from one activity node to another.

**Decision Nodes(Decision Points):**

Decision nodes represent points in the workflow where a decision needs to be made based on some condition or criteria. They are depicted as diamonds and are used to model branching behavior within the diagram. Depending on the outcome of the decision, the control flow may follow different paths.

**Merge Nodes:**

Merge Nodes represent points in the workflow where multiple paths of control flow converge back into a single path. They are depicted as small circles and are used to merge alternative paths back into a single flow of control.

**Fork Nodes:**

Fork Nodes represent points in the workflow where a single path of control flow splits into multiple concurrent paths. They are depicted as short bars and are used to model parallel or concurrent activities within the diagram.

**Join Nodes:**

Join nodes represent points in the workflow where multiple concurrent paths of control flow converge back into a single path. They are depicted as short bars with multiple incoming control flow arrows and are used to synchronize parallel activities back into a single flow of control.

**Final node:**

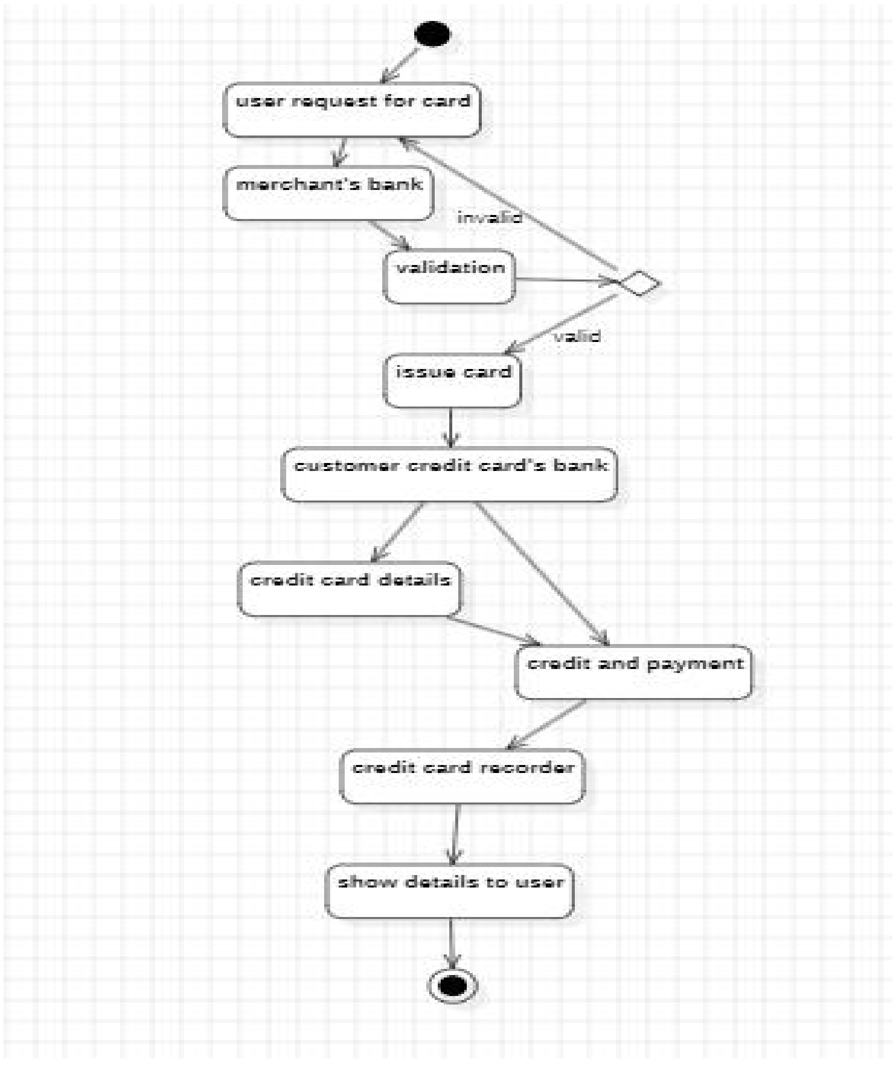
Final node represents the end point of activity diagram. It indicates where the workflow or process ends and is depicted as a solid-filled circle with a border.

## Activity diagram for credit card processing:

Activity diagram model the flow of activities or actions within a system, including decision points and parallel activities. For a credit card processing system, let's consider the process of requesting of credit card and processing of issuing credit card.

## Activities:

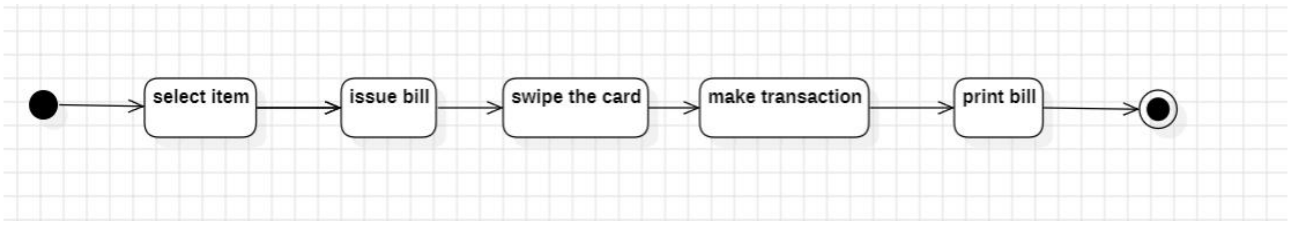
- **Customer:** customer initiates the process by request the credit card by the application with necessary details by merchant's bank.
- **Credit card's bank:** It is responsible for issuing credit card and also give the information of credit card to the customer. It also responsible for the credit and debit payments.
- **Credit card recorder:** It is responsible to record the credit and debit payment data and show the status to the customer.



## State chart diagram:

State chart diagram define different states of an object during its lifetime and these states are changed by events. State chart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

## State chart diagram for credit card processing:



The state chart of the credit card processing system defines the process of using the credit card while purchasing.

- The customer initially select the product for purchase and issue the bill for that product
- Swipe the card in the card reader and get the card and customer information by using the secret pin.
- Make the transaction the credit amount by the credit card's bank. The credit card recorder records all the payment and show the current status to the customer.
- Print the bill for using the credit card to make the payments and save it to the database.

## Result:

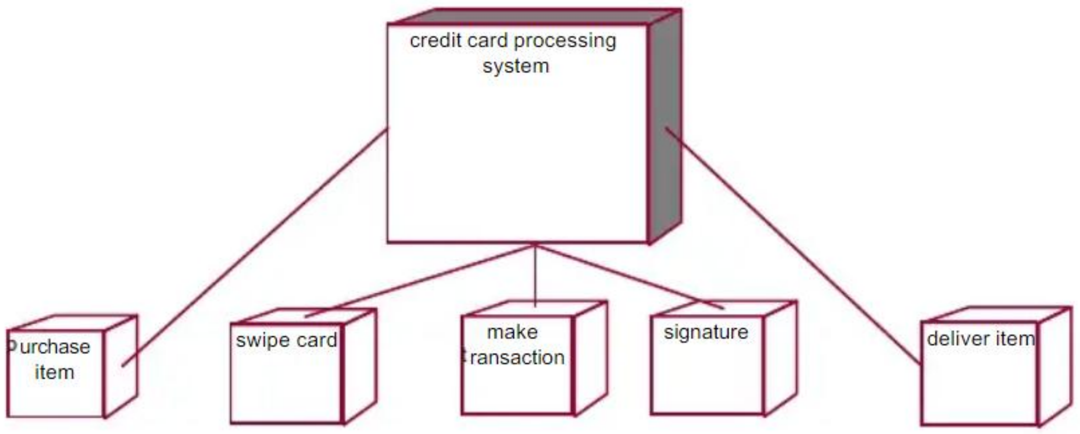
Thus the activity diagram and state chart for the credit card processing system was developed successfully.

| EX NO:07<br><br>DATE: | **Implement the system as per the detailed design** |
|---|---|

**Aim:**

To implement the credit card processing system as per the design

**Implementation diagram:**



**Coding:-**

**1.Credit Card Processing System**

```
Public Class  Form1

Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)Handles Button3.Click

Form2.Show()End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)Handles Button2.Click

Form3.Show()

End Sub
```

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)Handles Button1.Click

Form4.Show()

End Class
```

**2.Signup Form:**

```
Public Class Form2

Dim drAsOdbc.OdbcDataReader

Dim sql As String

Dim sql1 As String

Dim count As Integer

Dim bs As Integer

Dim conn As Odbc.OdbcConnection

Dim comm As Odbc.OdbcCommand

Dim ConnectionStringAs String

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)Handles Button1.Click

Sql1=”inert into SIGNUP2 values(“& TextBox1.Text & “,” & TextBox2.Text &
“,” & TextBox3.Text & “)”

MsgBox(sql1)

ConnectionString=”DSN=ADHARSH;UID=SCOTT;PWD=TIGER”

conn=New Odbc.OdbcConnection(ConnectionString)
```

```
conn.Open()

comm=New Odbc.OdbcCommand(sql1.conn)

Try

dr=comm.ExecuteReader()

Catch ex As Exception

End Try

MsgBox(“INSERTED SUCCESSFULLY”)

conn.Close()

dr.Close()

comm.Dispose()

conn.Dispose()

End Sub

End Class
```

**3.ADMIN LOGIN FORM:**

```
Public Class Form3

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)Handles Button1.Click

If TextBox1.Text=”a”And Textbox2.Text=”a” Then

Form5.Show()

End If

End Sub

End Class
```

**4.User login:**

```
Public Class Form4

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)Handles Button1.Click

Dim drAsOdbc.OdbcDataReader

Dim sql As String

Dim count As Integer

Dim aid As Integer

Dim conn As Odbc.OdbcConnection

Dim comm As Odbc.OdbcCommand

Dim ConnectionStringAs String

sql=”select*from signup2 where USERNAME=”& TextBox1.Text & “and
PASSWORD=” & TextBox2.Text & “ “

ConnectionString=”DSN=ADHARSH;UID=SCOTT;PWD=TIGER”

conn=New Odbc.OdbcConnection(ConnectionString)

conn.Open()

comm=New Odbc.OdbcCommand(sl.conn)

dr=comm.ExecuteReader()

count=0

aid=aid+1

If aid=3 Then

Me.Hide()
```

```
End If
While dr.Read()
count=count+1
End While
If count=1 Then
Me.Hide()
Form6.Show()
ElseIf count=1 Then
Me.Hide()
Form3.Show()
Else
MessageBox.Show(“invalid credentials”)
End If
conn.Close()
dr.Close()
comm.Dispose()
conn.Dispose()
End Sub
End Class
```

**5.Admin Form:**

```
Public Class Form5
```

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)Handles Button1.Click

FORM7.SHOW()

End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)Handles Button2.Click

form8.show()

End Sub

End Class
```

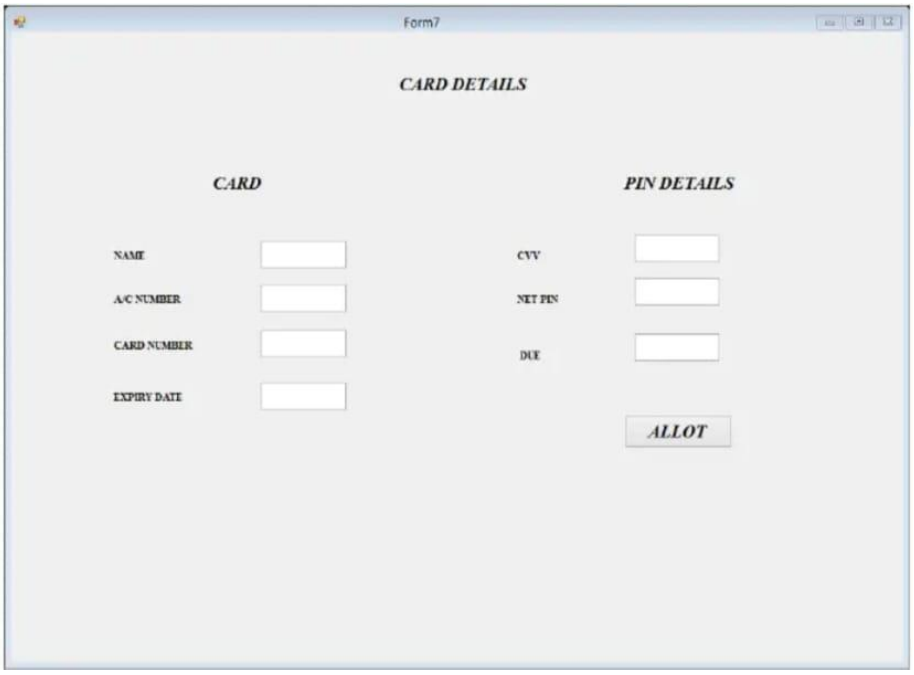
**6.User form:**

```
Public Class Form6

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)Handles Button1.Click

form9.show()

End Sub

End Class
```

**7.Allot form:**

```
Public Class Form7

Dim drAsOdbc.OdbcDataReader

Dim sql As String

Dim sql1 As String
Dim count As Integer

Dim bs As Integer

Dim conn As Odbc.OdbcConnection

Dim comm As Odbc.OdbcCommand

Dim ConnectionStringAs String

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)Handles Button1.Click

sql1=”inert into card values(“ & TextBox1.Text & “,” & TextBox2.Text & “,” &
TextBox3.Text & “,” & TextBox4.Text & “,” & TextBox5.Text & “,” &
TextBox6.Text & “,” & TextBox7.Text & “)”

MsgBox(sql1)

ConnectionString=”DSN=ADHARSH;UID=SCOTT;PWD=TIGER”

conn=New Odbc.OdbcConnection(ConnectionString)

conn.Open()

comm=New Odbc.OdbcCommand(sql1.conn)

Try

dr=comm.ExecuteReader()

Catch ex As Exception

End Try

MsgBox(“INSERTED SUCCESSFULLY”)

conn.Close()

dr.Close()
```

```
comm.Dispose()

conn.Dispose()

End Sub

End Class
```



**8.Due form:**

```
Public Class Form8

Dim drAsOdbc.OdbcDataReader

Dim sql As String

Dim sql1 As String

Dim count As Integer

Dim bs As Integer
Dim conn As Odbc.OdbcConnection

Dim comm As Odbc.OdbcCommand

Dim ConnectionStringAs String

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)Handles Button1.Click

sql1=”select due from card where CAR=”& TextBox1.Text & “”

MsgBox(sql1)

ConnectionString=”DSN=ADHARSH;UID=SCOTT;PWD=TIGER”

conn=New Odbc.OdbcConnection(ConnectionString)

conn.Open()

comm=New Odbc.OdbcCommand(sql1.conn)

Try

dr=comm.ExecuteReader()

bs=dr.GetValue(0)

TextBox2.Text=bs

Catch ex As Exception

End Try

MsgBox(“INSERTED SUCCESSFULLY”)

conn.Close()

dr.Close()

comm.Dispose()

conn.Dispose()
```

```
End Sub

End Class
```

**9.Pay form:**

```
Public Class Form9

Dim drAsOdbc.OdbcDataReader

Dim sql As String

Dim sql1 As String

Dim count As Integer

Dim bs As Integer

Dim conn As Odbc.OdbcConnection

Dim comm As Odbc.OdbcCommand

Dim ConnectionStringAs String

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)Handles Button1.Click

sql1=”select due from card where CAR=”& TextBox1.Text & “”

MsgBox(sql1)

ConnectionString=”DSN=ADHARSH;UID=SCOTT;PWD=TIGER”

conn=New Odbc.OdbcConnection(ConnectionString)

conn.Open()

comm=New Odbc.OdbcCommand(sql1.conn)

Try

dr=comm.ExecuteReader()
```

```
bs=dr.GetValue(0)

TextBox2.Text=bs

Catch ex As Exception

End Try

MsgBox("INSERTED SUCCESSFULLY")

conn.Close()

dr.Close()

comm.Dispose()

conn.Dispose()

Me.Visible=False

End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)Handles Button2.Click

FORM10.SHOW()

End Sub

End Class
```

## Result:

Thus the implementation of credit card processing system was executed successfully.

| | |
|---|---|
| EX NO:08<br><br>DATE: | **Test the Software System for All Scenarios Identified as Per the Use Case Diagram** |

**Aim:**

To test the software system for all scenarios identified in the use case diagram.

## Credit Card Processing System:

A Credit Card Processing System is a crucial tool for financial institutions and merchants to efficiently handle credit card transactions, ensure security compliance, and provide seamless payment experiences to customers. It encompasses functionalities such as transaction authorization, payment processing, fraud detection, account management, and reporting.

## Review Use Case Diagram:

- Examine interactions and functionalities depicted in the use case diagram specific to the Credit Card Processing System.
- Identify primary actors (users) and their objectives within the system, including cardholders, merchants, financial institutions, and system administrators.
- Ensure that the use case diagram accurately reflects the system's requirements and desired behaviors for credit card processing.

## Identify Test Scenarios:

- Consider scenarios from various user perspectives, including cardholders, merchants, and financial institutions.

- Cover a wide range of functionalities, including transaction authorization, payment processing, fraud detection, account management, and reporting.
- Include both positive scenarios (expected system behavior) and negative scenarios (error handling, exception scenarios) for comprehensive testing.

## Create Test Cases:

- Creating test cases for a credit card processing system involves examining various scenarios to ensure that the system operates accurately and securely. Here are some test cases aspects of credit card processing:
  - ➢ **Valid Credit Card Transaction**
  - ➢ **Invalid Credit Card Number**
  - ➢ **Expired Credit Card**
  - ➢ **Invalid CVV**
  - ➢ **Transaction Amount Exceeds Credit Limit**
  - ➢ **Transaction with Insufficient Funds**
  - ➢ **Duplicate Transaction**
  - ➢ **Connection Timeout**
  - ➢ **Fraudulent Activity Detection**
  - ➢ **Refund Transaction**
  - ➢ **Partial Refund Transaction**
  - ➢ **Transaction in Different Currencies**
  - ➢ **Transaction Logging**
  - ➢ **System Performance Test**
  - ➢ **Security Test**
- Ensure coverage of both functional requirements (e.g., transaction processing) and non-functional requirements (e.g., security, performance).

## Prepare Test Environment:

Setting up a test environment for a credit card processing system involves several steps to ensure accuracy, security, and functionality.

- **Hardware and Software Requirements:**
  - Identify the hardware and software requirements for the credit card processing system, including servers, databases, network infrastructure, and client machines.
  - Ensure compatibility with the software components, such as operating systems, databases, web servers, and programming languages.
- **Development, Testing, and Staging Environments:**
  - Set up separate environments for development, testing, and staging to ensure proper separation of duties and to prevent unintended consequences in production.
  - **Development environment:** Used by developers to write and test code.
  - **Testing environment:** Used by quality assurance (QA) engineers to perform functional and non-functional testing.
  - **Staging environment**: Represents a replica of the production environment where final testing and user acceptance testing (UAT) occur before deployment.
- **Payment Gateway Integration:**
  - Integrate the credit card processing system with a test payment gateway provided by a payment processor.
  - Obtain test merchant accounts and payment credentials to simulate transactions without incurring actual charges.
- **Database Setup:**
  - Create a database schema for the credit card processing system.
  - Populate the database with sample data to simulate real-world scenarios.

- Implement data masking and anonymization techniques to protect sensitive information during testing.

## Execute Test Cases:

- Follow the defined test cases meticulously, recording all steps taken and observed results during test execution.
- Utilize diverse test data and scenarios to validate system behavior under various transaction conditions and user interactions.
- Document any deviations from expected outcomes and prioritize defect resolution based on severity and impact.

## Unit testing:

- **Purpose:** Unit testing focuses on testing individual units or components of the system in isolation to ensure they work correctly.
- **Process:** Developers write unit tests for functions, methods, or classes to verify their behavior against expected outcomes. These tests are typically automated and run frequently during development
- **Example:**

    1. **Validation**: Ensure that the credit card number, expiration date, and CVV are validated correctly.
    2. **Transaction Processing**: Test that transactions are processed correctly, including successful transactions, declined transactions, and transactions with insufficient funds.
    3. **Error Handling**: Verify that errors are handled appropriately, such as invalid inputs or network errors.

## 2. Black Box Testing:

- **Purpose:** Black box testing examines the functionality of the system without knowing its internal implementation details.
- **Process:** Testers interact with the system's user interface and inputs to validate outputs against expected results. They do not have access to the system's code or internal structure.
- **Example:**
  - The system should accurately process valid transactions and reject invalid ones.
  - It should handle different scenarios such as authorization failures, network issues, and concurrency gracefully without compromising security or performance.
  - The system should adhere to industry standards and regulations for credit card processing such as PCI DSS.

## 3. White Box Testing:

- **Purpose:** White box testing examines the internal logic, code structure, and paths within the system.
- **Process:** Testers have access to the system's source code and use this knowledge to design test cases that exercise specific code paths and conditions. They aim to achieve high code coverage and identify potential defects in the implementation.
- **Example:**
  White box testing for a credit card processing system involves examining the internal logic, structure, and code of the system to ensure its functionality, security, and adherence to specifications.

## 4. Integration and System Testing:

- **Integration Testing Purpose:** Integration testing verifies that individual units or components work together correctly as a whole system.
- **Integration Testing Process:** Testers combine units or components and test their interactions and interfaces to ensure they integrate seamlessly.
- **System Testing Purpose:** System testing evaluates the entire system as a whole to ensure it meets specified requirements and functions correctly in its intended environment.
- **System Testing Process:** Testers execute end-to-end scenarios and test cases to validate system behaviour, performance, security, and usability.
- **Example:**
  - Simulate a credit card transaction through the payment gateway.
  - Verify that the transaction data is correctly forwarded to the banking system.
  - Confirm that the banking system processes the transaction and sends back a response.
  - Validate that the response is received by the payment gateway and appropriately relayed to the merchant.

## Regression Testing:

- Conduct regression testing to ensure that recent changes or updates have not introduced new defects or regressions in existing system functionality.
- Re-run previously executed test cases to verify the stability and integrity of core credit card processing features.
- Automate regression test suites where feasible to streamline testing efforts and ensure consistent coverage across system releases.

- **Example:**

1. **Initial Test Case**:
    1. **Objective**: Verify that a credit card transaction is successfully processed and recorded in the system.
    2. **Steps**:
        - Input valid credit card details (card number, expiration date, CVV).
        - Input transaction amount within acceptable limits.
        - Submit the transaction.
    3. **Expected Outcome**: The system should process the transaction successfully without errors and record the transaction details accurately.
    4. **Regression Test Scenario**:
        - Change: A software update is implemented to improve the encryption algorithm used for credit card data storage.
        - Regression Test Objective: Ensure that the update hasn't introduced any regressions or issues in the credit card processing functionality.
        - Steps:
            - Repeat the initial test case with valid credit card details.
            - Verify that the transaction is processed successfully.
            - Verify that the transaction details are  stored in the database.
            - Check for any errors or anomalies in the transaction or data storage.
            - **Expected Outcome**: The transaction should be processed without errors, and the transaction details should be accurately stored in the system, demonstrating that the update hasn't impacted the core functionality of credit card processing.

- **Additional Regression Scenarios**:
    - **Scenario 1**: Change in third-party API integration for verifying card authenticity.

1. **Objective:** Ensure that the integration change hasn't affected the verification process.
2. **Steps:** Repeat the initial test case with various scenarios of card verification (valid card, expired card , etc.).
3. **Expected Outcome:** The system should correctly verify card authenticity and process transactions accordingly.

- **Scenario 2: Update in the user interface for displaying transaction details.**

1. **Objective:** Ensure that the UI update hasn't impacted the accuracy of transaction data display.
2. **Steps:** Perform transactions and verify that transaction details are correctly displayed to users.
3. **Expected Outcome:** Transaction details should be displayed in the updated UI without any inconsistencies or errors**.**

## Documentation and Reporting:

- Document all test results, including observed defects, test coverage metrics, and performance benchmarks, in comprehensive test reports.
- Prepare detailed test documentation summarizing testing activities, findings, and recommendations for stakeholders.
- Update test documentation iteratively based on feedback and lessons learned during the testing process.

## Result:

Thus testing of the software system for all scenarios identified in the use case diagram was implemented and executed successfully.

| | |
|---|---|
| EX NO:09<br><br>DATE: | **Improve the reusability and maintainability of the software by applying appropriate design patterns** |

## Aim:

To improve the reusability, maintainability, and overall quality of the software system by applying appropriate design patterns.

## Problem statement:

Design and implement an efficient credit card creation process that optimizes both user experience and security measures, addressing challenges such as identity verification, fraud prevention, and seamless integration with banking systems, to ensure a streamlined and secure issuance of credit cards while minimizing potential risks and enhancing customer satisfaction.

## Credit Card Processing System Overview:

A Credit Card Processing System is a robust solution designed to efficiently handle credit card transactions within a financial institution or merchant environment. It encompasses various functionalities such as transaction authorization, payment processing, fraud detection, account management, and reporting.

## Model-view-controller pattern:

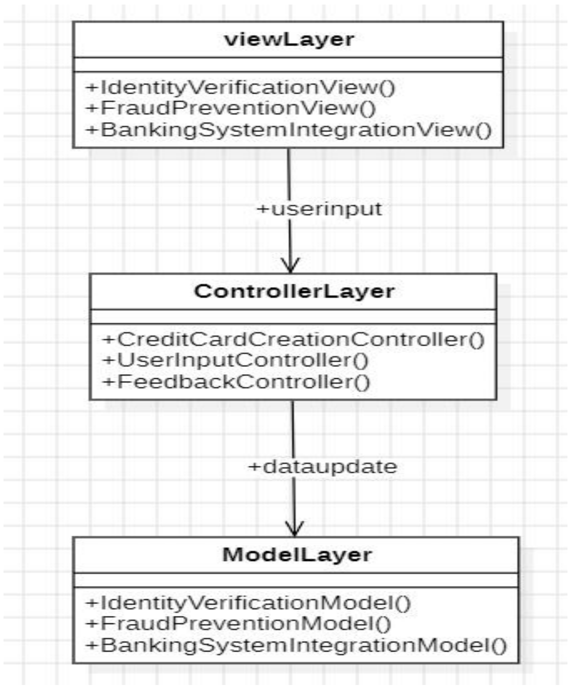
The Model-View-Controller (MVC) pattern is a software architectural pattern commonly used for developing user interfaces. It divides an application into three interconnected components:

1. **Model**: Represents the application's data and core logic. It manages the data, responds to queries about the data, and updates itself when changes

occur. The model notifies its associated views when the data changes, so they can update accordingly.

2. **View**: Displays the data to the user and sends user commands (e.g., mouse clicks, keystrokes) to the controller. The view should ideally be separated from the model and only be concerned with displaying data and interacting with the user.
3. **Controller**: Acts as an intermediary between the model and the view. It receives user input from the view, processes it (possibly modifying the model), and updates the view accordingly. The controller is responsible for interpreting user actions and invoking the appropriate methods on the model.

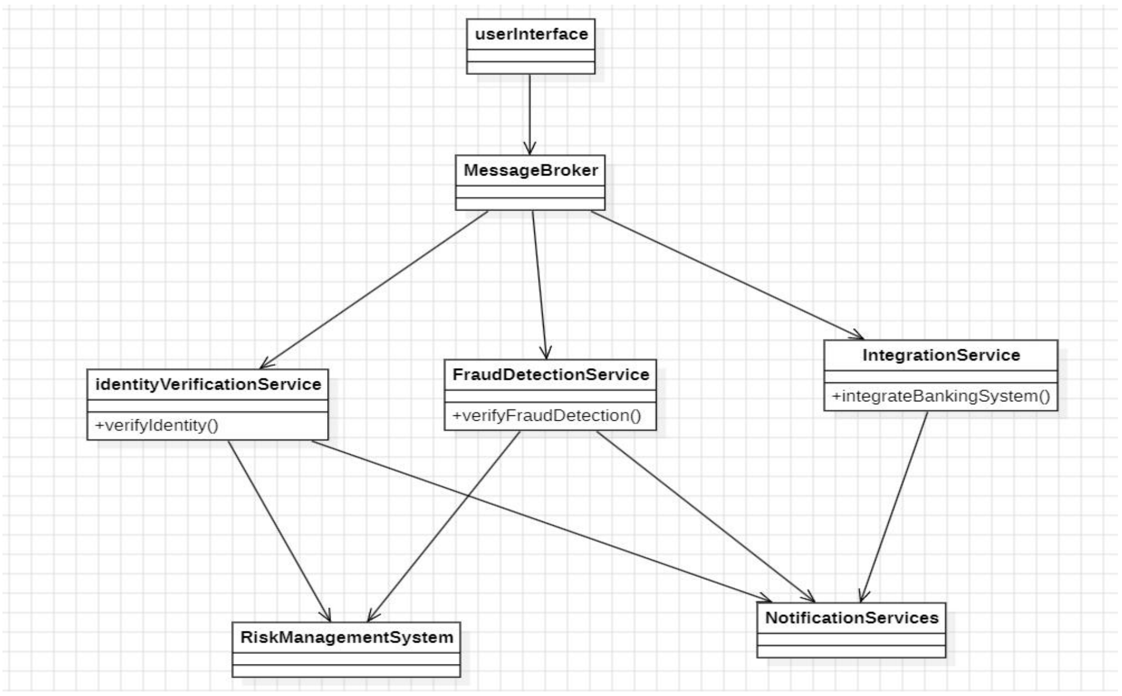
**Example:**


viewLayer
+IdentityVerificationView()
+FraudPreventionView()
+BankingSystemIntegrationView()
+userinput
ControllerLayer
+CreditCardCreationController()
+UserInputController()
+FeedbackController()
+dataupdate
ModelLayer
+IdentityVerificationModel()
+FraudPreventionModel()
+BankingSystemIntegrationModel()

## Publish-subscribe pattern:

The publish-subscribe pattern is a messaging pattern used in software architecture to decouple the sender (publisher) of a message from its receivers (subscribers). In this pattern, publishers don't send messages directly to specific subscribers.

**Example:**

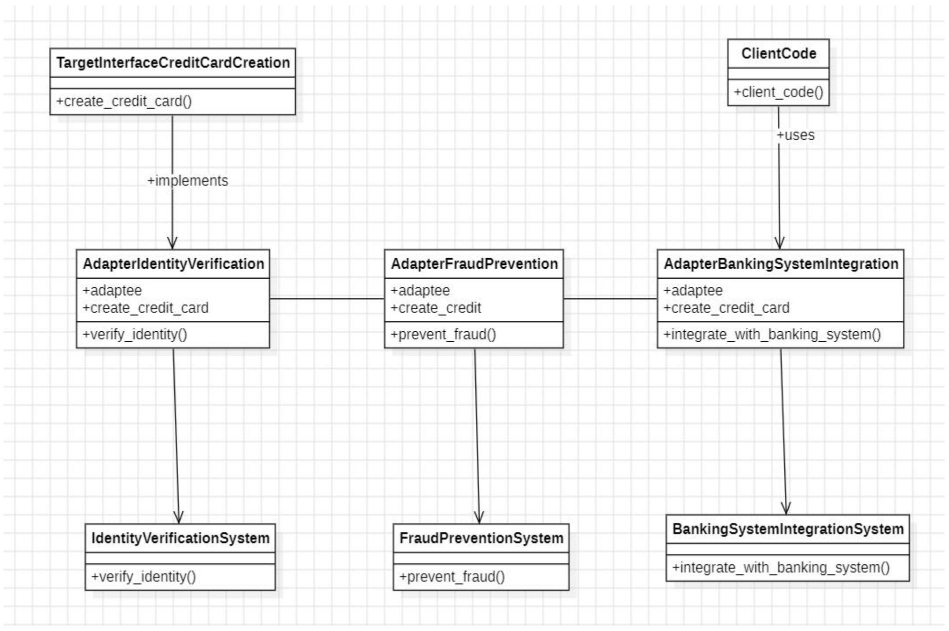


## Adapter pattern:

The Adapter Pattern is a design pattern commonly used in software engineering. It allows objects with incompatible interfaces to work together. Here's how it typically works:

1. **Target Interface**: This is the interface that the client expects. It's the interface that the client code knows and uses.
2. **Adaptee**: This is the interface that needs to be adapted to work with the client. It's the interface that the client cannot directly use.
3. **Adapter**: This is the class that bridges the gap between the Target Interface and the Adaptee. It implements the Target Interface and internally uses an instance of the Adaptee to perform the required operations.
4. **Client**: This is the code that uses the Target Interface to interact with the Adaptee through the Adapter.

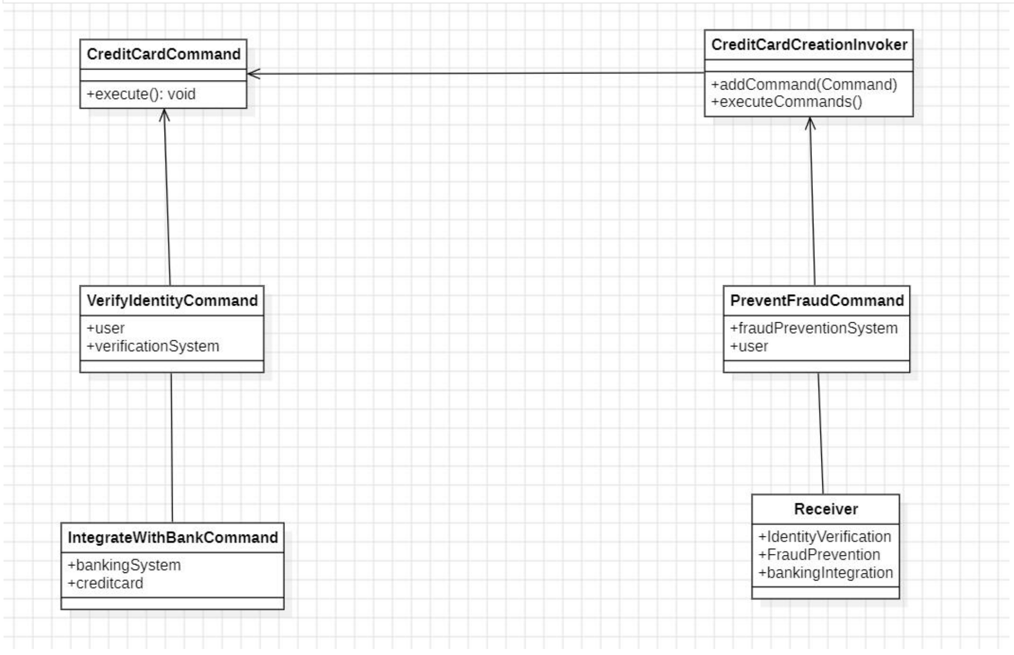
**Example:**



## Command pattern:

The Command Pattern is a behavioral design pattern that encapsulates a request as an object, thereby allowing parameterization of clients with queues, requests, and operations. In essence, it turns requests into objects, allowing you to parameterize clients with queues, requests, and operations.

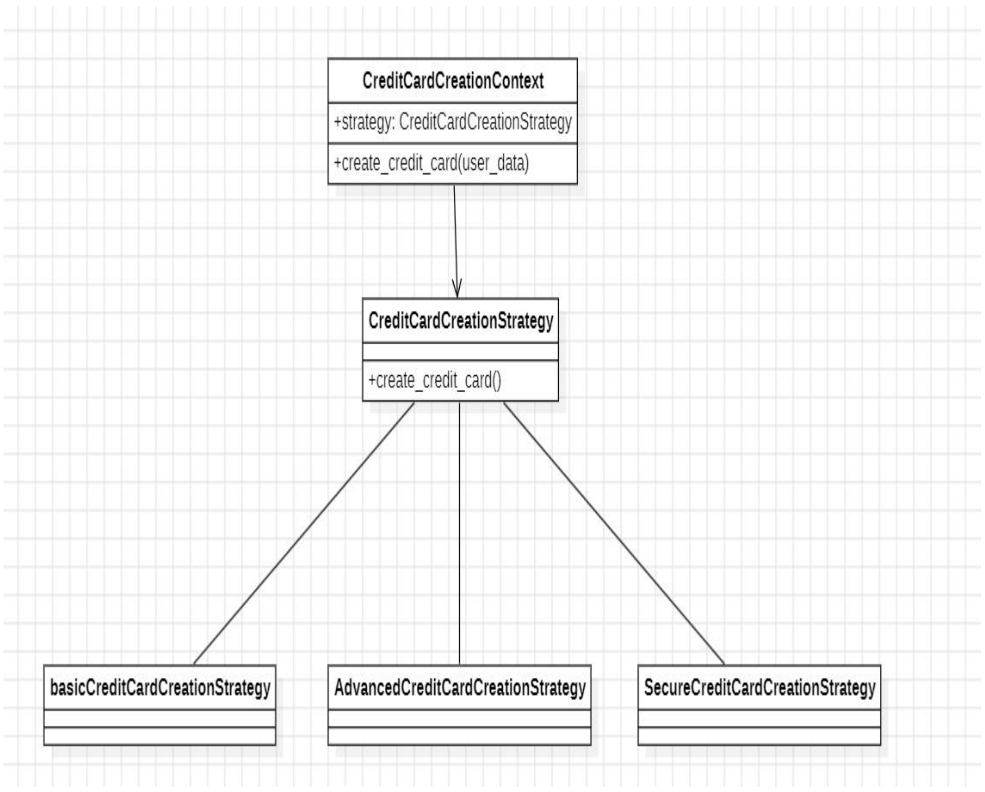
**Example:**



## Strategy Pattern:

The Strategy pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable. In the context of the credit card processing system, we can utilize the Strategy to handle business rules and algorithms related to transaction routing, fee calculation, or fraud detection strategies.

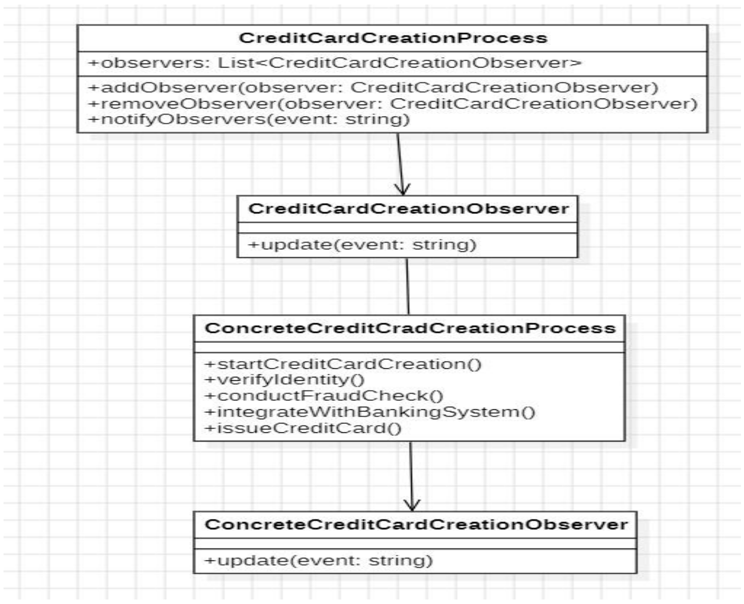
**Example:**


CreditCardCreationContext
+strategy: CreditCardCreationStrategy
+create_credit_card(user_data)
CreditCardCreationStrategy
+create_credit_card()
basicCreditCardCreationStrategy
AdvancedCreditCardCreationStrategy
SecureCreditCardCreationStrategy

## Observer Pattern:

The Observer pattern establishes a one-to-many dependency between objects, where multiple observers are notified of changes in a subject's state. Within the credit card processing system, we can leverage the Observer pattern to facilitate real-time updates and notifications regarding transaction status or account changes.

**Example:**

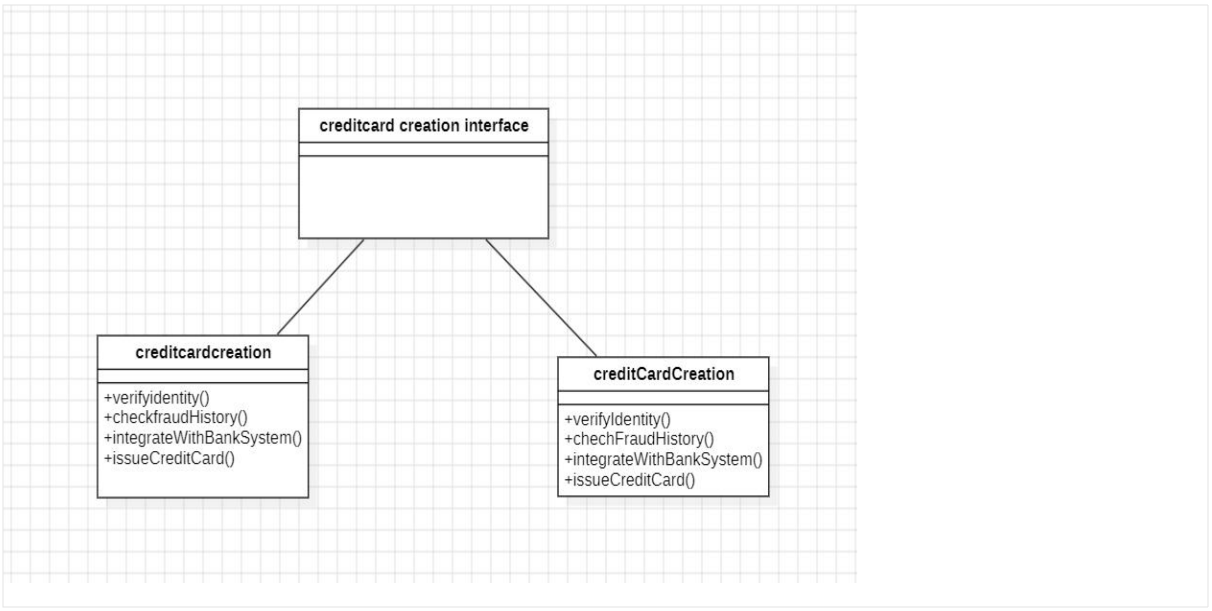


CreditCardCreationProcess
+observers: List<CreditCardCreationObserver>
+addObserver(observer: CreditCardCreationObserver)
+removeObserver(observer: CreditCardCreationObserver)
+notifyObservers(event: string)
CreditCardCreationObserver
+update(event: string)
ConcreteCreditCradCreationProcess
+startCreditCardCreation()
+verifyIdentity()
+conductFraudCheck()
+integrateWithBankingSystem()
+issueCreditCard()
ConcreteCreditCardCreationObserver
+update(event: string)

## Proxy pattern:

The Proxy Pattern is a structural design pattern that provides a surrogate or placeholder for another object to control access to it. Essentially, it acts as an intermediary or a wrapper around the actual object, intercepting requests and performing additional operations if needed before passing them to the underlying object.

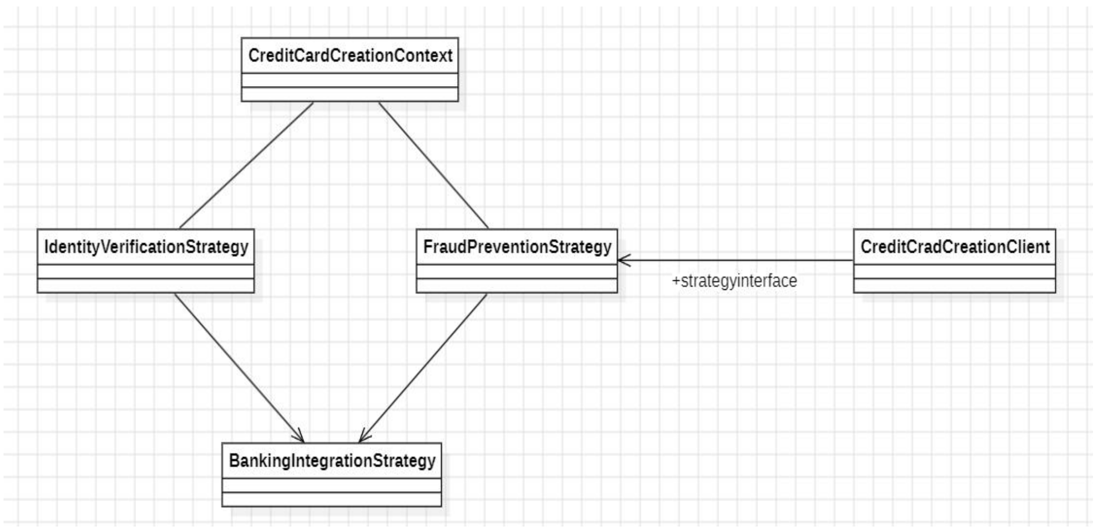
**Example:**



## Facade Pattern:

The Facade pattern provides a unified interface to a set of interfaces in a subsystem, simplifying their usage and shielding clients from their complexities. In the credit card processing system, we can employ the Facade pattern to create a unified interface for interacting with various subsystems, such as transaction processing, authorization, or reconciliation.

**Example:**


CreditCardCreationContext
IdentityVerificationStrategy
FraudPreventionStrategy
CreditCradCreationClient
+strategyinterface
BankingIntegrationStrategy

## Result:

Thus the reusability and maintainability of the software system by applying appropriate design patterns was executed successfully.