Final Term Exam

Object oriented system analysis and design

Name ; Muhammad Sameed Khan
ID ; 6843

Q1: **Draw Use Case diagram**
Propose a use case diagram for a vending machine that sells beverages and snacks. Make use of inclusion and extension associations and remember that a vending machine may need technical assistance from time to time.

ANS: **Use case diagram**

In this section we give some examples of use case diagrams for various situations.

Simplified ATM machine

Propose a use case diagram for an ATM machine for withdrawing cash. Make the use case simple yet informative; only include the major features.

Solution

The use case diagram is given in Fig.1

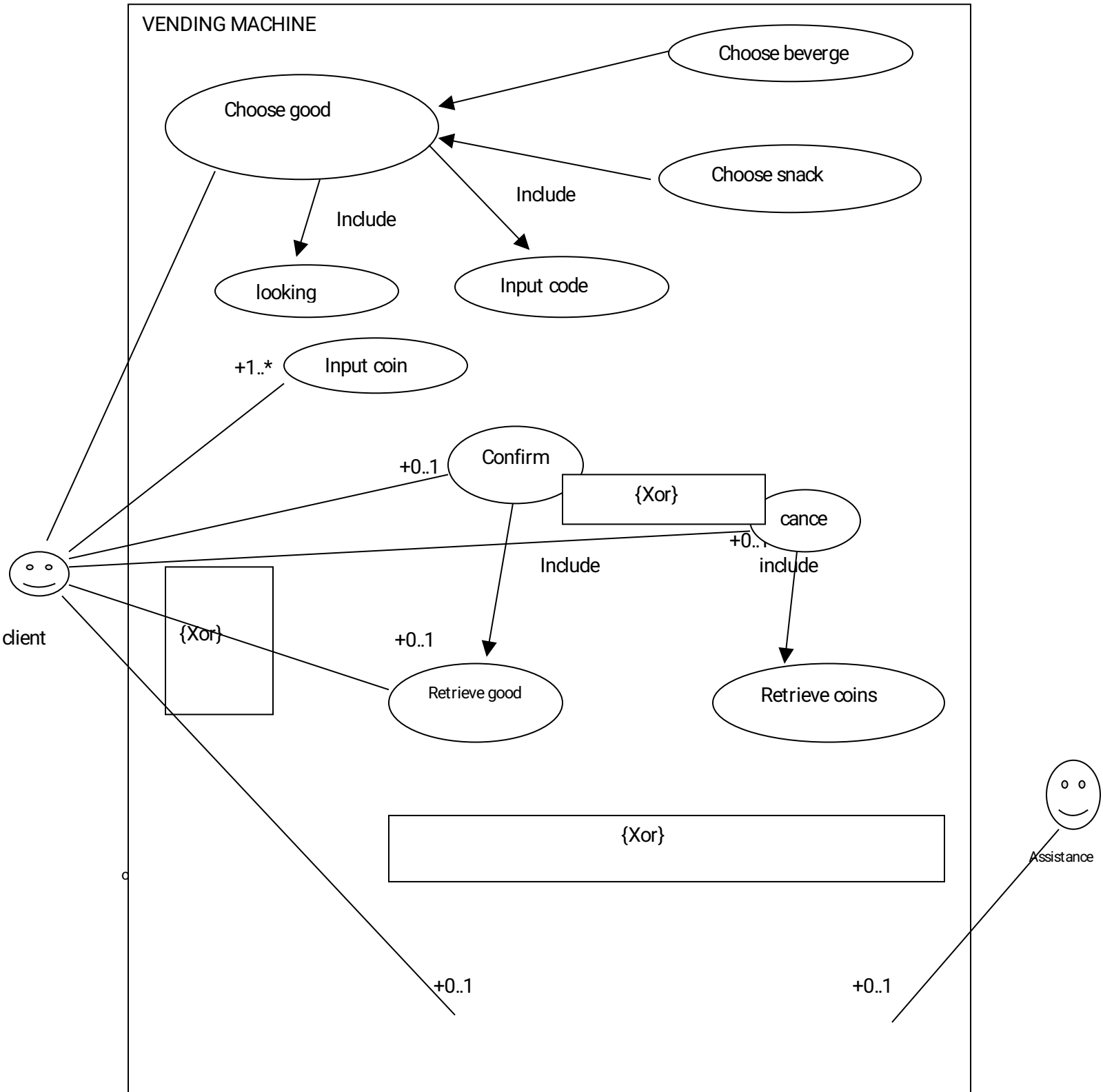
Vending machine

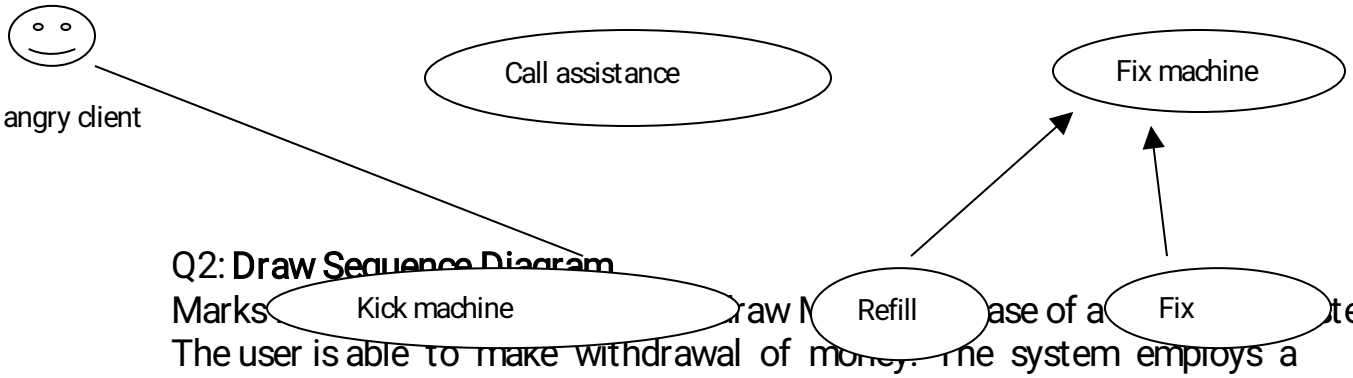
Propose a use case diagram for a vending machine that sells beverages and snacks. Make use of inclusion and extension associations, mark multiplicities and remember that a vending machine may need technical assistance from time to time.

Solution

The use case diagram is given in Fig. 2. We remark that the "+" character in front of multiplicities should not be there and that the {xor} constraint should be marked by a dashed segment rather than a box (the umbrello UML modeller that comes with the KDE linux desktop has some bugs and limitations).

The usecase diagram oF a vending machine fiig 2

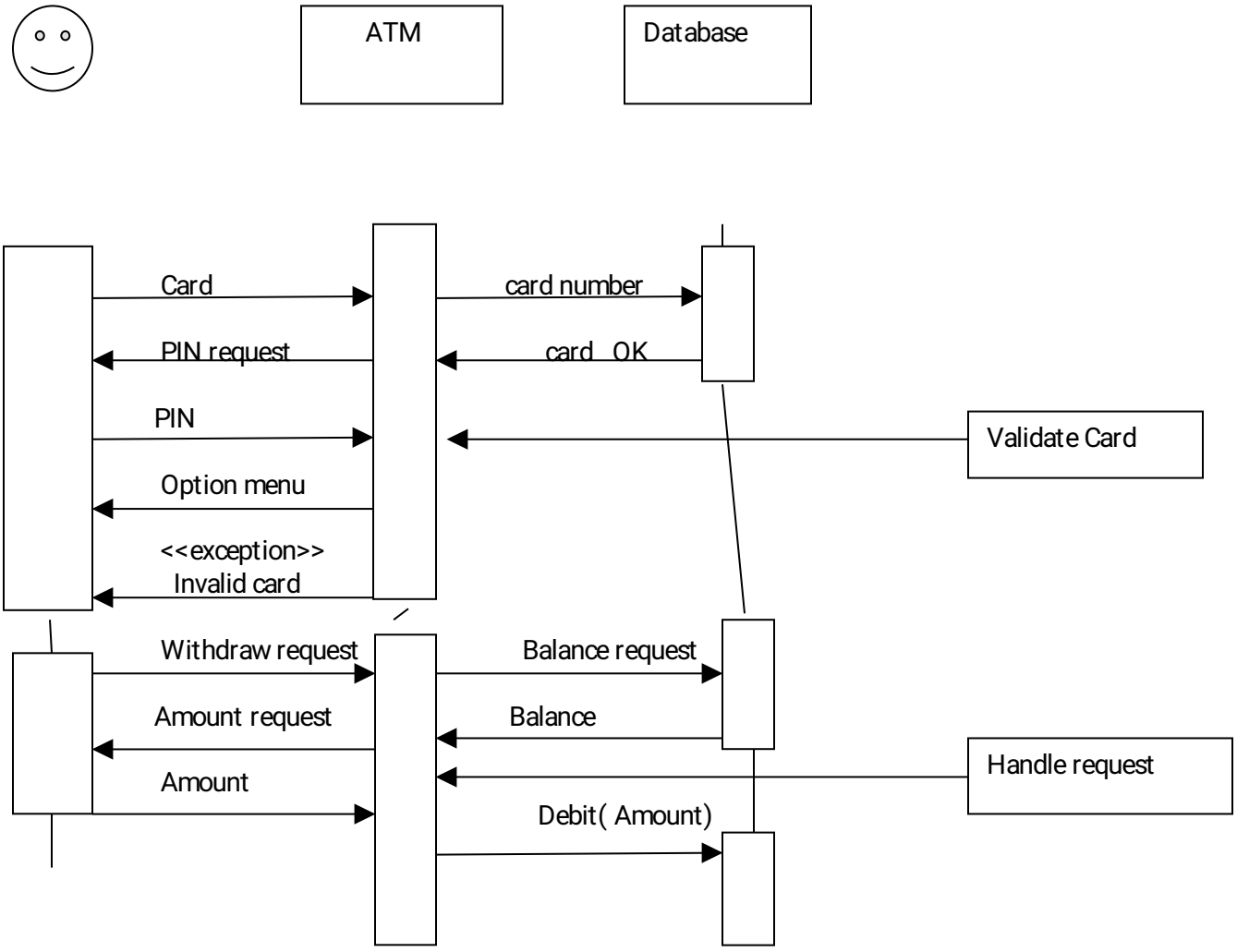


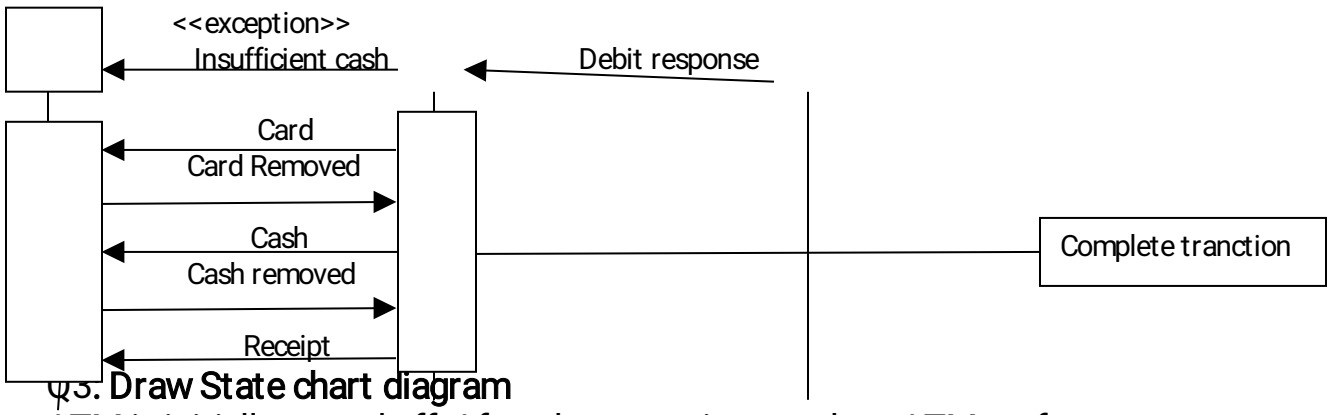


Q2: Draw Sequence Diagram
Marks: raw M ase of a tem.
The user is able to make withdrawal of money. The system employs a standard procedure of validating the card and account holder's password

Ans

Sequence diagram of ATM withdrawal

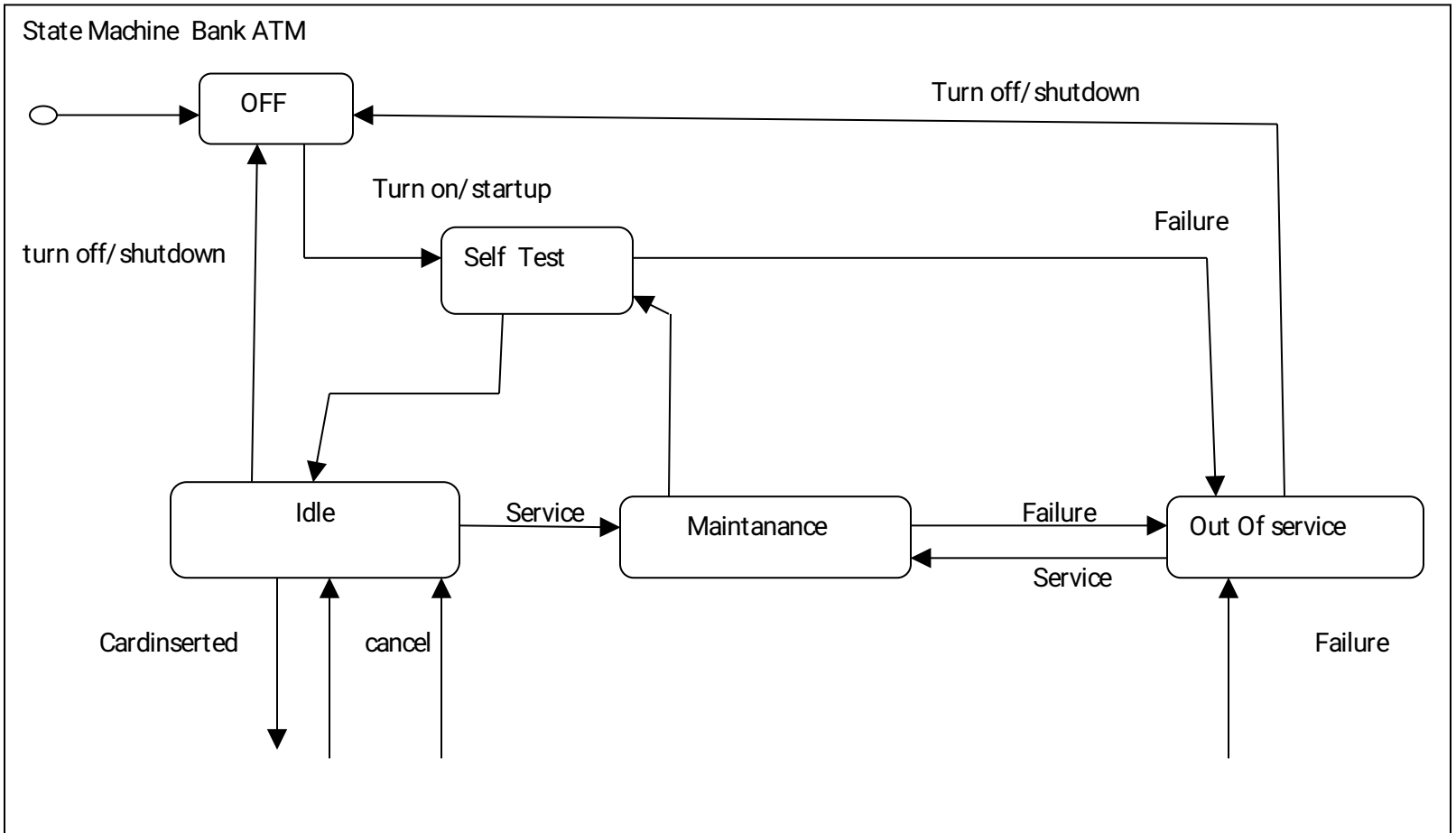


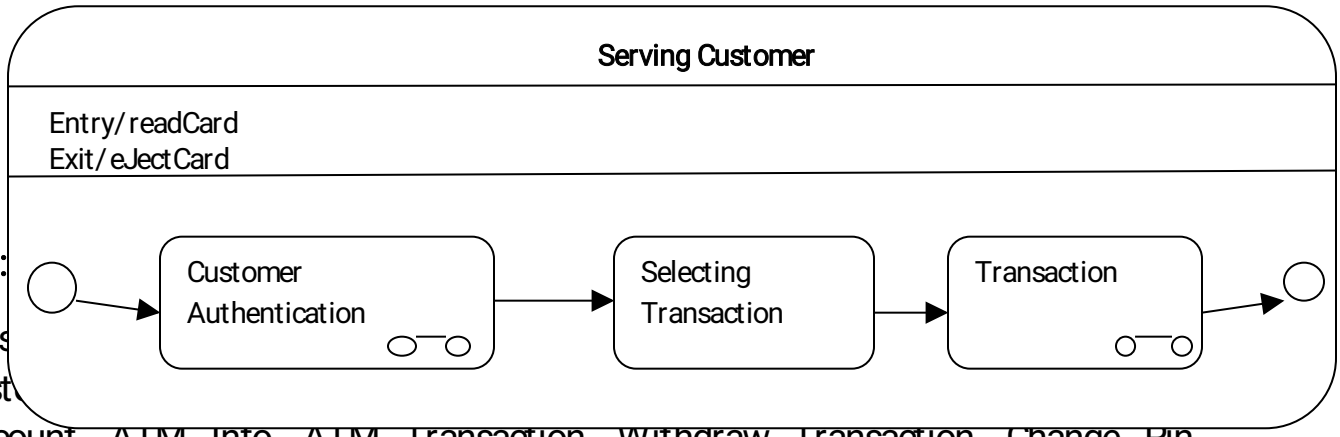


Q3. Draw State chart diagram

ATM is initially turned off. After the power is turned on, ATM performs startup action and enters Self Test state. If the test fails, ATM goes into Out of Service state, otherwise transition to the Idle state. In this state ATM waits for customer interaction. The ATM state changes from Idle to Serving Customer when the customer inserts banking or credit card in the ATM's card reader. On entering the Serving Customer state that is composed of basic ATM functions i.e authentication, money withdrawal etc

ANS:





Q4:

Illus
syst
Account, ATM Info, ATM Transaction, Withdraw Transaction, Change Pin, Transfer Money, Check Balance. The Bank maintains personal and ATM information of each customer. The customer can access their account using Debit Card issued by the Bank. In this system there could be two types of Account: Current Account and Saving Account. Both use to share many of the properties and methods. The ATM Machine can perform multiple transactions such as Withdrawing cash, change pin, check balance and Transfer Money to each account.

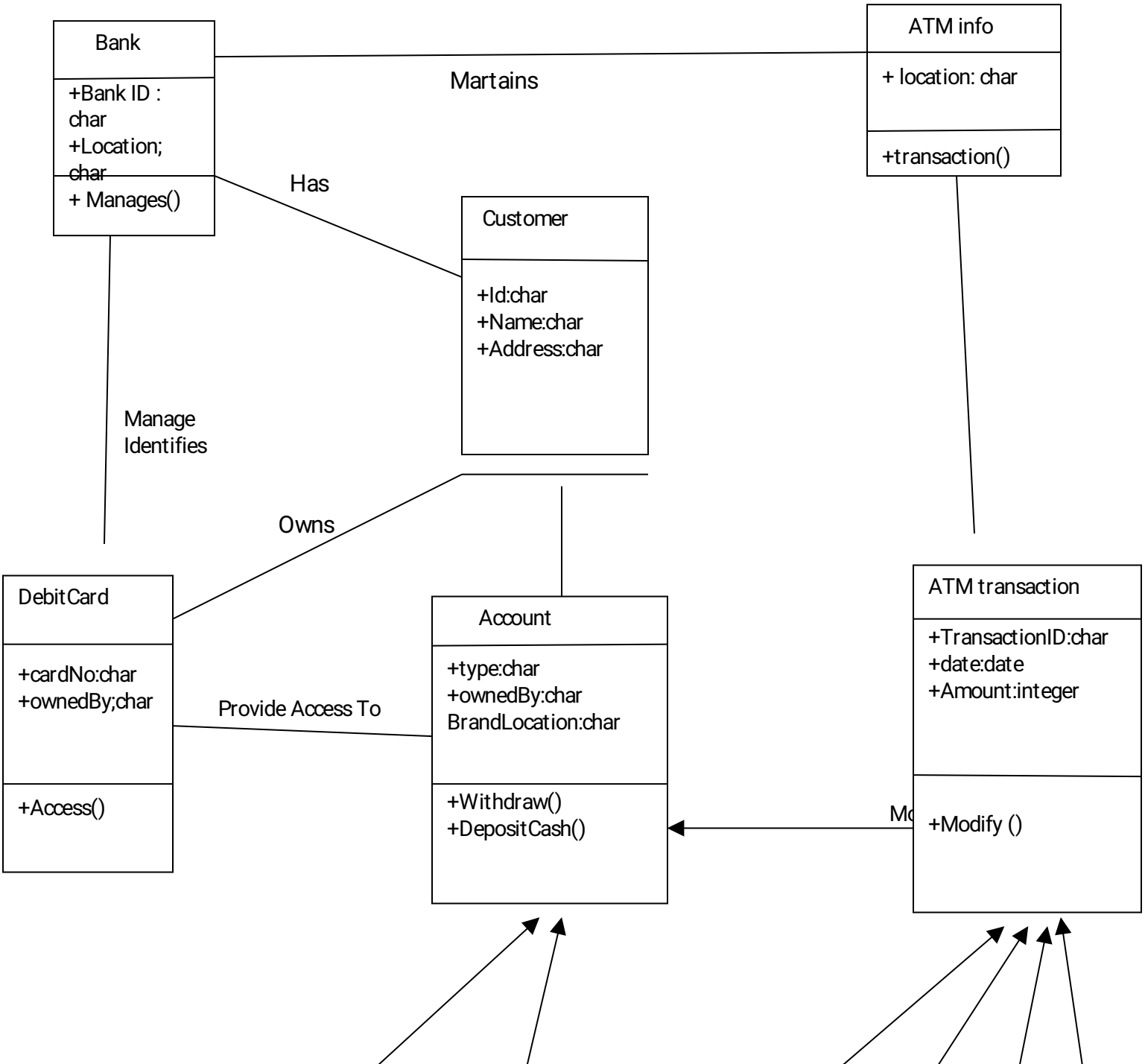
ANS :

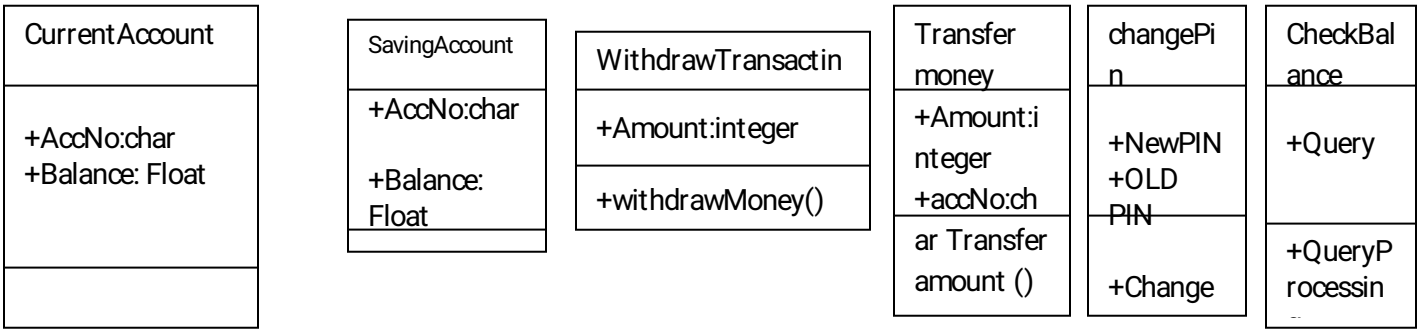
ILLUSTRATE UML CLASS DIAGRAM FOR ATM MACHINES

Bank, Account,Customer Ino ,DebitCard CurrentACCOUNT Saving Account ,ATM ino, ATM Transactions, Withdraw Transactions ,Change Pin Transer Money, checkBalance

The Bank Maintains personal and ATM information o each customer. The customer can access their account using Debit card issurd by the Bank. In this system there could be two types o

Account: Current Account and Saving Account. Both use to share many o the properties and methods . The ATM Machines can perorm multiple transactions such as Withdraw cash, change pin, checkbalance and transfer money to each account





Q5: Design Pattern

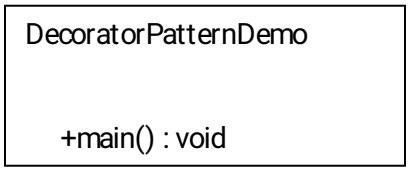
Suppose we have the following java files. Identify the pattern also Considering the java files draw class diagram.

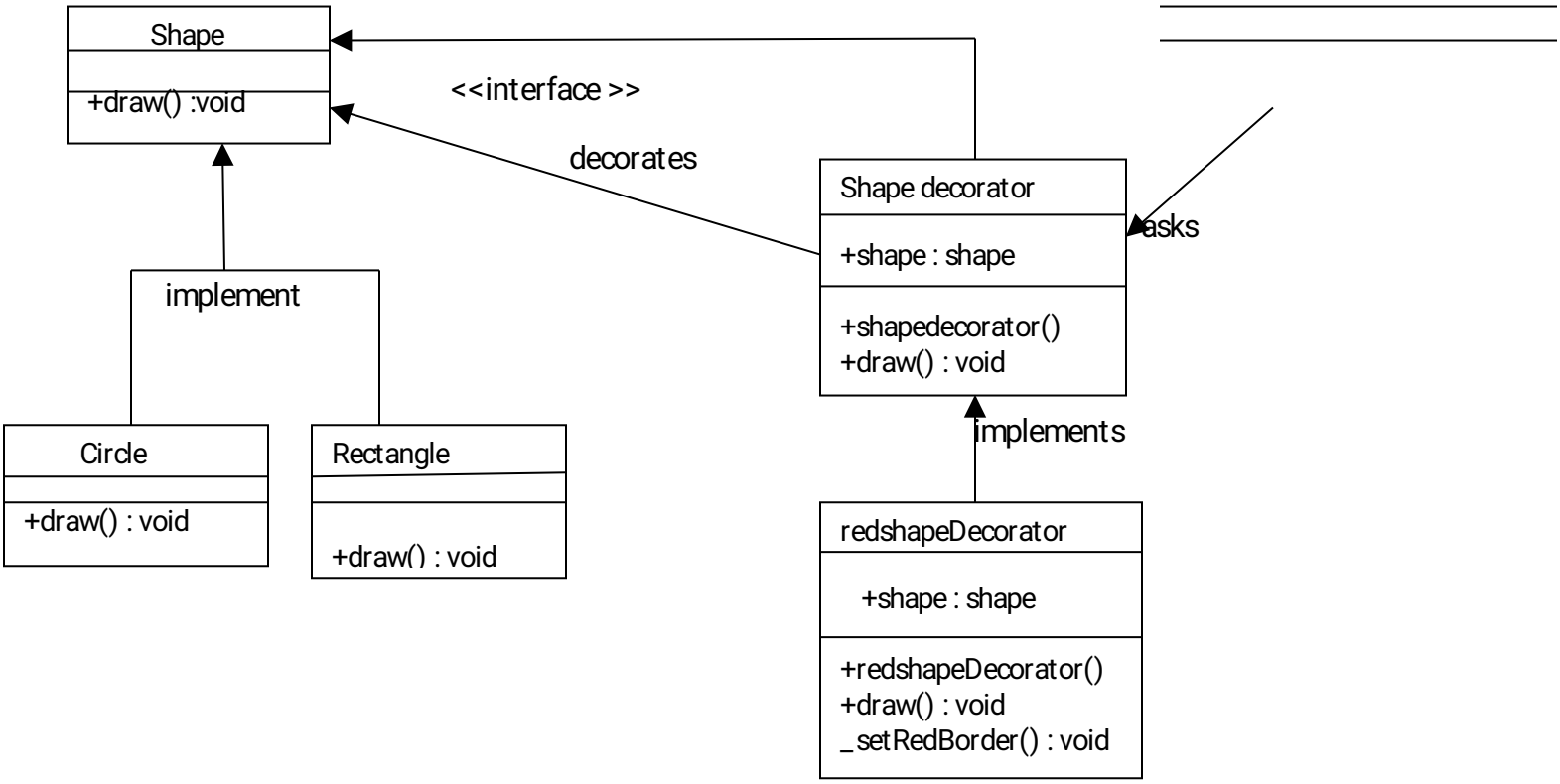
ANS :

We're going to create a Shape interface and concrete classes implementing the Shape interface. We will then create an abstract decorator class ShapeDecorator implementing the Shape interface and having Shape object as its instance variable.

RedShapeDecorator is concrete class implementing ShapeDecorator.

DecoratorPatternDemo, our demo class will use RedShapeDecorator to decorate Shape objects.





Decorator Pattern UML Diagram

Step 1

Create an interface.

Shape.java

```
public interface Shape {
   void draw();
}
```

Step 2

Create concrete classes implementing the same interface.

Rectangle.java

```
public class Rectangle implements Shape {

   @Override
   public void draw() {
      System.out.println("Shape: Rectangle");
   }
}
```

Circle.java

```
public class Circle implements Shape {

   @Override
   public void draw() {
      System.out.println("Shape: Circle");
   }
}
```

Step 3

Create abstract decorator class implementing the Shape interface.

ShapeDecorator.java

```
public abstract class ShapeDecorator implements Shape {
```

```
   protected Shape decoratedShape;

   public ShapeDecorator(Shape decoratedShape){
      this.decoratedShape = decoratedShape;
   }

   public void draw(){

      decoratedShape.draw();
   }
}
```

Step 4

Create concrete decorator class extending the ShapeDecorator class.

RedShapeDecorator.java

```
public class RedShapeDecorator extends ShapeDecorator {

   public RedShapeDecorator(Shape decoratedShape) {
      super(decoratedShape);
   }
```

```
   @Override
   public void draw() {
      decoratedShape.draw();
      setRedBorder(decoratedShape);
   }

   private void setRedBorder(Shape decoratedShape){
      System.out.println("Border Color: Red");




   }
}
```

Step 5

Use the RedShapeDecorator to decorate Shape objects.

DecoratorPatternDemo.java

```
public class DecoratorPatternDemo {
   public static void main(String[] args) {

      Shape circle = new Circle();

      Shape redCircle = new RedShapeDecorator(new Circle());
```

```
      Shape redRectangle = new RedShapeDecorator(new Rectangle());
      System.out.println("Circle with normal border");
      circle.draw();

      System.out.println("\nCircle of red border");
      redCircle.draw();

      System.out.println("\nRectangle of red border");
      redRectangle.draw();
   }
}
```

Step 6

Verify the output.

```
Circle with normal border
Shape: Circle

Circle of red border
```

Shape: Circle

Border Color: Red

Rectangle of red border

Shape: Rectangle

Border Color: Red