ATM Case Study

1. **Classes Used in the ATM**

   1)ATM

   2)Account (for users)

2. **Interfaces Used in the ATM**

   Services

   Features Of My ATM

   I. my code contains 2 roles user and admin
   II. User has to create a new account at first and then use the features available for an existing user
   III. While creating an account in ATM user needs to note the account number generated at first then he needs to create a password which is of length greater than 5 and should only contain digits and pin should not contain 0 as starting number
   IV. Then he should enter his name which should only contain alphabets
   V. Then he needs to enter his phone number and should only contain numbers and +country code
   VI. Later ATM generates the stored information he entered so that user can verify and can change them Later(if they are wrong)
   VII. If any of the above rules are violated then the atm warns him
   VIII. Rules for Using the existing Account by user
   IX. First he needs to enter his account number and then he needs to enter the pin kept by the user
   X. He have many features like checking his balance ,adding money to his account ,withdrawing money from his account, getting his las three transactions, getting all the transactions made by him till now and changing his credentials
   XI. After using the atm he needs to exit so that other user can use the atm
   XII. For Admin
   XIII. Admin can see the amount present in the atm
   XIV. Admin can add Money to atm if money in atm is insufficient so that users can take money from atm

XV. Admin can also keep atm under maintenance so that any hardware repairs can be done without losing data

XVI. After using atm admin needs to exit so that user can use the ATM

3. Main Methods used in ATM

1) Methods in Account class

Setter and getters for various fields

i. PrintQ(); //prints last 3 transactions made by user
ii. PrintQ(int d); //prints whole transactions made by user
iii. SetQ(String s); //adds elements to queue
iv. getAccountNo();//Getter for field Account No
v. setAccountNo();//Setter for field Account No
vi. GetPassword(); //Getter for field Password
vii. setPassword(); //Setter for field Password
viii. getName(); //getter for field Name
ix. setName(); //setter for field Name
x. setPhNo(); //setter for Phone number of user
xi. getPhNo(); //getter for Phone number of user
xii. setBalence(); //it sets balance of the user
xiii. getBalence(); //it returns the balance of the user

Ex: for Queue q, Account No., Password, Name, Phone No etc

2)Methods in ATM class

i. Getter to get Password for admin
ii. Boolean **IsPossible()**

It checks possibility whether the atm has enough money for the transaction and invokes the remove function and gives money to user

iii. **Print()**

It prints all the denominations supported by the atm and respective amount present in the atm.

It is an admin specific function it helps him whether the atm has enough money to work

iv. **addAmt()**

It adds amount from either user or from admin/Bank manager to atm and adds user balance if user has deposited the money

v. **remove()**

It removes certain amount of notes required to make the transaction from the atm and gives the amount to user

4. **Methods in services Interface**

These are same as in ATM class

As there are sensitive they need a abstraction (DATA HIDING)

5. **Functions in Main class**

Test();

It is a menu driven program and is called from main function

Main();

It is driver code that contains exception handling and calls the test function.

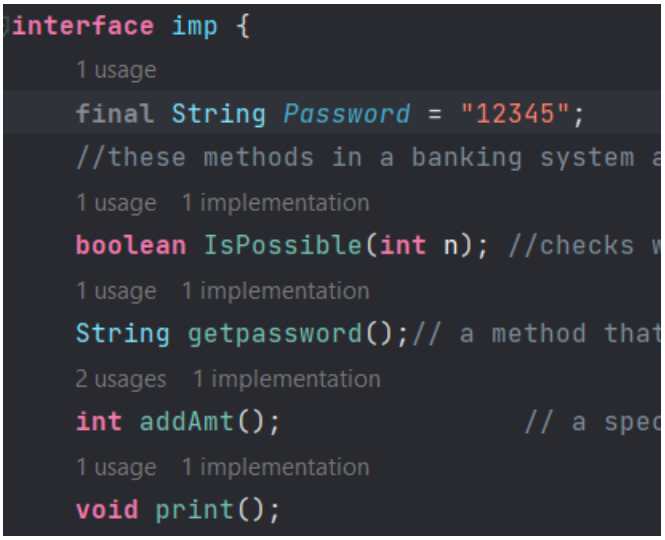
OOPS CONCEPTS USED TO MAKE THE ATM

Inheritance: I have used inheritance to implement some of the methods from services to ATM as a child class can have methods from parent class

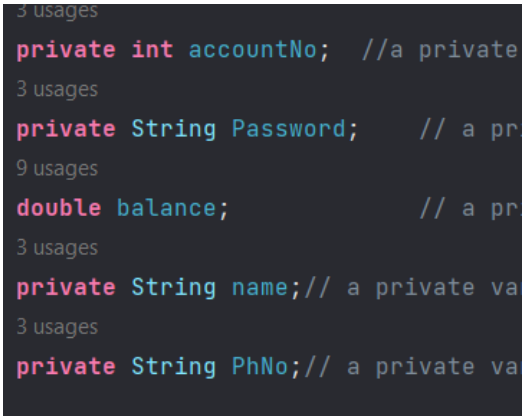
```
class Account extends ATM {      //Account class which stores d
    8 usages
    Queue<String> q1=new LinkedList<>();
    10 usages
    Queue<String> q = new LinkedList<>();       // This Queue h
    Account(int accNO,String pwd,String name,String phNumber){
        this.accountNo=accNO;
        this.Password=pwd;
        this.name=name;
        this.PhNo=phNumber;
    }
```

```
class ATM implements imp {

    int one = 100, two = 100, five = 100, ten = 100, twenty = 1

    //initially our atm is filled with 100 each 1,2,5(coins) :

    public String getpassword() {
        return Password;
    }//this method returns password to check whether admin inpu


    public boolean IsPossible(int n) {...}

    public void print() {...}

    public int addAmt() {
        Scanner sc = new Scanner(System.in);
```

Abstraction: I used abstraction in order to hide some data from user



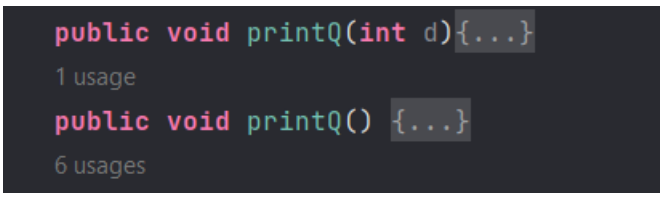
Encapsulation: I have used that to restrict the direct access to some components of the object, so users cannot access state values if the variables of a particular object



Polymorphism: I have used Polymorphism while printing the transactions made by the user

If user has asked for last 3 transactions it invokes one function

If user asked for all transactions it invokes other function

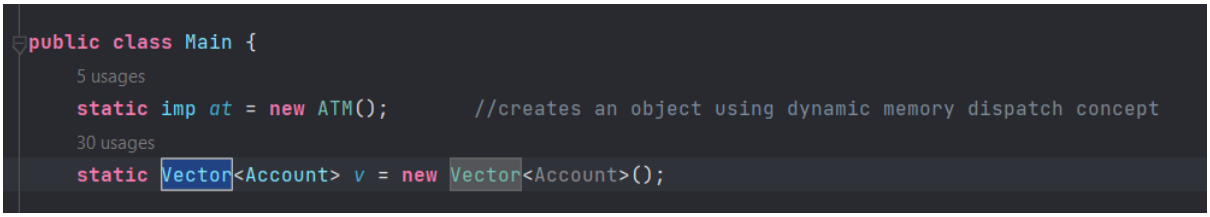


Concepts other than OOPS used in this ATM

1. Used Some **Data structures**

   Queue: To store the transactions made by the user

   Vector: To store the user data I have used it as a vector is a dynamic array and is easy to use than arrays



```
public class Main {
    5 usages
    static imp at = new ATM();        //creates an object using dynamic memory dispatch concept
    30 usages
    static Vector<Account> v = new Vector<Account>();
```

2. Used **Exception Handling**:

   As user data is very important to us giving input string in place of an int can lead us to big problems thus I used Exception handling to over come the situation

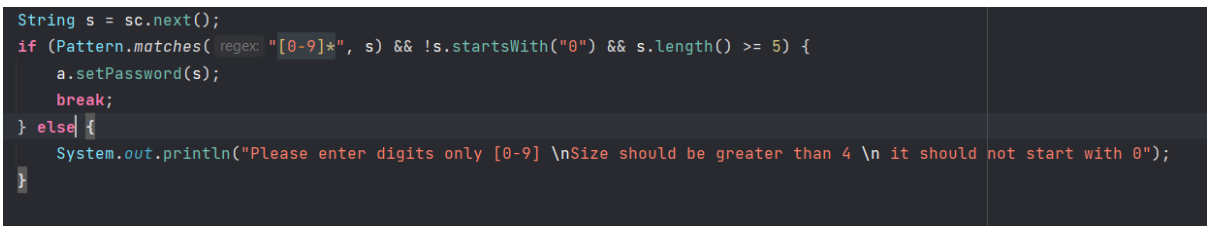
```
try {
    TEST();          //Here We Used Exception handling so that our code works even user input format
} catch (Exception e) {
    System.out.println("Please Enter in Right Format");
    //here if exception is found we tell the user to enter in right format and run the code as usually
    main(args);
}
```

   q

3. Used **Regex** so that User don't give us wrong data

   i. We know that Pin for an account must be of digit format and should be greater than 5 digits length.

   Thus, I have restricted the user to have alphabets and special characters to have them in pin and length>=5



```
String s = sc.next();
if (Pattern.matches( regex: "[0-9]*", s) && !s.startsWith("0") && s.length() >= 5) {
    a.setPassword(s);
    break;
} else {
    System.out.println("Please enter digits only [0-9] \nSize should be greater than 4 \n it should not start with 0");
}
```

   ii. We know the name of the user has to be only alphabets

   So I restricted user to enter integers in the name section

```
s = sc.next();
if (Pattern.matches( regex: "[a-z A-Z]*", s)) {
    a.setName(s);
    break;
} else {
    System.out.println("Please enter Alphabets Only(Excluding Symbols a
}
```

iii. As Phone number is always of digit format with a symbol +
I have included it and can be made of 0-9 and “+” symbol

```
if (Pattern.matches( regex: "[0-9 + ]*", s)) {
    a.setPhNo(s);
    break;
} else {
    System.out.println("Please enter digits only [0-9] \n");
    s = sc.next();
}
```