1. Design a class named Circle containing following attributes and behavior. • One double data field named radius. The default value is 1. • A no-argument constructor that creates a default circle. • A Single argument constructor that creates a Circle with the specified radius. • A method named getArea() that returns area of the Circle. • A method named getPerimeter() that returns perimeter of it.

```
public class Circle{
    private double radius;
    public Circle(){
        radius=1.0;
    }
    public Circle(double r)
    {
        radius=r;
    }
    public double getArea(){
        return radius*radius*Math.PI;
    }
    public double getPerimeter(){
        return radius*2*Math.PI;
    }
}
public class Check{
    public static void main(String[] args)
    {
        Circle c1=new Circle();
        System.out.println("Perimeter = "+c1.getPerimeter());
        System.out.println("Area = "+c1.getArea());
        Circle c2=new Circle(2.5);
        System.out.println("Perimeter = "+c2.getPerimeter());
        System.out.println("Area = "+c2.getArea());
    }
}
```

2. Design a class named Account that contains: • A private int data field named id for the account (default 0). • A private double data field named balance for the account (default 500₹). • A private double data field named annualInterestRate that stores the current interest rate (default 7%). Assume all accounts have the same interest rate. • A private Date data field named dateCreated that stores the date when the account was

created. • A no-arg constructor that creates a default account. • A constructor that creates an account with the specified id and initial balance. • The accessor and mutator methods for id, balance, and annualInterestRate. • The accessor method for dateCreated. • A method named getMonthlyInterestRate() that returns the monthly interest rate. • A method named getMonthlyInterest() that returns the monthly interest. • A method named withdraw that withdraws a specified amount from the account. • A method named deposit that deposits a specified amount to the account.

```
class Account {
    private int id = 0;
    private double balance = 500.0;
    private static double annualInterestRate = 7.0;
    private java.util.Date dateCreated;

    public Account() {
        dateCreated = new java.util.Date();
    }

    public Account(int id2, double balance2) {
        id = id2;
        balance = balance2;
    }

    public int getId() {
        return this.id;
    }

    public double getBalance() {
        return this.balance;
    }

    public double getAnnualInterestRate() {
        return annualInterestRate;
    }

    public String getDateCreated() {
        return this.dateCreated.toString();
    }

    public void setId(int id) {
        this.id = id;
    }
    public void setBalance(double balance) {
        this.balance = balance;
    }

    public void setAnnualInterestRate(double annualInterestRate) {
        this.annualInterestRate = annualInterestRate;
    }

    public double getMonthlyInterestRate() {
        return (annualInterestRate / 100) / 12 ;
    }

    public double getMonthlyInterest() {
        return balance * getMonthlyInterestRate();
    }

    public void withdraw(double amount) {
        this.balance -= amount;
    }

    public void deposit(double amount) {
        this.balance += amount;
    }
}
```

3. Use the Account class created as above to simulate an ATM machine. Create 10 accounts with id AC001…..AC010 with initial balance 300₹. The system prompts the users to enter an id. If the id is entered incorrectly, ask the user to enter a correct id. Once an id is accepted, display menu with multiple choices. 1. Balance inquiry 2. Withdraw money [Maintain minimum balance 300₹] 3. Deposit money 4. Money Transfer 5. Create Account 6. Deactivate Account 7. Exit Hint: Use ArrayList, which is can shrink and expand with compared to Array.

```
import java.util.Scanner;
import java.util.ArrayList;
import java.util.*;
public class accountCheck {
private static final Scanner in = new Scanner(System.in);

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

```
    ArrayList<Account> list = new ArrayList<Account>();
    Account a1 = new Account("ACC01",500);
    Account a2 = new Account("ACC02",500);
    Account a3 = new Account("ACC03",500);
    Account a4 = new Account("ACC04",500);
    Account a5 = new Account("ACC05",500);
    Account a6 = new Account("ACC06",500);
    Account a7 = new Account("ACC07",500);
    Account a8 = new Account("ACC08",500);
    Account a9 = new Account("ACC09",500);
    Account a10 = new Account("ACC10",500);
for (int i = 1; i < 11; i++) {
           list.add(a[i]);
        }

    static String getAccount() {
        Scanner sc = new Scanner(System.in);
        while (true) {
            System.out.println("Enter Account Number : ");
            String ac_no = sc.next();
            if (ac_no.charAt(5) ="1"|| ac_no.charAt(5) ='2' || ac_no.charAt(5)
='3' || ac_no.charAt(5) ='4' || ac_no.charAt(5) ='5' || ac_no.charAt(5) ='6'
|| ac_no.charAt(5) ='7' || ac_no.charAt(5) ='8' || ac_no.charAt(5) ='9' ||
ac_no.charAt(4) ='1') {return ac_no;
            }
            System.out.println("Invalid  account no");
        }
    }


        String ac_no;
        System.out.print("""
                1. Balance inquiry\n
                2. Withdraw money\n
                3. Deposit money\n
                4. Money Transfer\n
                5. Create Account\n
                6. Deactivate Account\n
                7. Exit\s
                Enter your Choice :\r""");
        int choice = sc.nextInt();
        switch (choice) {
            case 1:
                ac_no=getAccount();
                list.get(ac_no).BalanceInquiry();
                break;
            case 2:
                ac_no=getAccount();
                int amo;
                System.out.println("Enter amount want to withdraw : ");
                amo = sc.nextInt();
                list.get(ac_no).withdraw(amo);
                break;
            case 3:
                ac_no=getAccount();
                int amo2;
                System.out.println("Enter amount want to Deposit : ");
                amo2 = sc.nextInt();
                list.get(ac_no).deposit(amo2);
                break;
            case 4:
                int amo3;
                ac_no = getAccount();
                System.out.println("Enter amount want to Transfer : ");
                amo3 = sc.nextInt();
                System.out.println("Enter account no : ");
                string ac_no2 = sc.next();
            case 5:
                System.out.println("Enter amount you want to deposit & minimum
300 required :");
                int amo4 = sc.nextInt();
                Account anew = new Account(i + 1, amo4);
                list.add(anew);
                System.out.println("Your account number is "+ (i+1));
                list.get(i).BalanceInquiry();
                break;
            case 6 :
                ac_no = getAccount();
                list.remove(ac_no);
                System.out.println("Your account is removed successfully");
                break;
            default:
                break;
        }

public class Account {
private String id = "ACC00";
    private double balance = 500.0;
    private static double annualInterestRate = 7.0;
    private java.util.DatedateCreated;

    public Account() {
        dateCreated = new java.util.Date();
    }

    public Account(String id2, double balance2) {
        id = id2;
        balance = balance2;
    }

    public String getId() {
        return this.id;
    }

    public double getBalance() {
        return this.balance;
    }

    public double getAnnualInterestRate() {
        return annualInterestRate;
    }

    public String getDateCreated() {
        return this.dateCreated.toString();
    }

    public void setId(String id) {
        this.id = id;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }

    public void setAnnualInterestRate(double annualInterestRate) {
        this.annualInterestRate = annualInterestRate;
    }

    public double getMonthlyInterestRate() {
        return (annualInterestRate / 100) / 12 ;
    }

    public double getMonthlyInterest() {
        return balance * getMonthlyInterestRate();
    }

    public void withdraw(double amount) {
        this.balance -= amount;
    }

    public void deposit(double amount) {
        this.balance += amount;
    }
}
```

4. (Subclasses of Account) In Programming Exercise 2, the Account class was defined to model a bank account. An account has the properties account number, balance, annual interest rate, and date created, and methods to deposit and withdraw funds. Create two subclasses for checking and saving accounts. A checking account has an overdraft limit, but a savings account cannot be overdrawn. Draw the UML diagram for the classes and then implement them. Write a test program that creates objects of Account, SavingsAccount, and CheckingAccount and invokes their toString() methods.

```java
class Account {
    private int id;
    double balance;
    private static double annualInterestRate;
    private java.util.Date dateCreated;

    public Account() {
        dateCreated = new java.util.Date();
    }

    public Account(int newId, double newBalance) {
        id = newId;
        balance = newBalance;
        dateCreated = new java.util.Date();
    }

    public int getId() {
        return this.id;
    }

    public double getBalance() {
        return balance;
    }

    public static double getAnnualInterestRate() {
        return annualInterestRate;
    }

    public void setId(int newId) {
        id = newId;
    }

    public void setBalance(double newBalance) {
        balance = newBalance;
    }

    public static void setAnnualInterestRate(double newAnnualInterestRate) {
        annualInterestRate = newAnnualInterestRate;
    }

    public double getMonthlyInterest() {
        return balance * (annualInterestRate / 1200);
    }

    public java.util.Date getDateCreated() {
        return dateCreated;
    }

    public void withdraw(double amount) {
        balance -= amount;
    }

    public void deposit(double amount) {
        balance += amount;
    }
}
public class checkAcc{



    public static void main (String[] args) {
        SavingAccount savings = new SavingAccount(5,1500);
        CheckingAccount checking = new CheckingAccount(4,1500);
        Account account = new Account();
        savings.withdraw(2000);
        checking.checking(3000);
        System.out.println(savings.getBalance());
        System.out.println(checking.getBalance());

    }

    public static void Print (double x){
        System.out.println("The current balance is "+" Rs "+x);
    }
}
public class CheckingAccount extends Account {
    double overDraft = -1000;

    public CheckingAccount(int newId, double newBalance) {
        super(newId, newBalance);
    }

    public void checking(double i) {

        if (balance - i < overDraft){
            System.out.println("Failure: Can't overdraft more than Rs1,000"
                    + "A Rs 25 overdraft fee will be issued to your account.
");
            balance = balance - 25; }
        else
            balance = balance - i;
    }
}
public class SavingAccount extends Account{
    double overdraftLimit = 0;

    public SavingAccount(int newId, double newBalance) {
        super(newId, newBalance);
    }

    public void withdraw (double w) {
        if (balance - w < overdraftLimit)
            System.out.println("Insufficient Funds");
        else
            balance = balance - w;
    }
}
```

5. Develop a Program that illustrate method overloading concept.

```
class Adding {
    static int add(int a, int b) {
        return a + b;
    }

    static double add(double a, double b) {
        return a + b;
    }
}

class Overloading {
    public static void main(String[] args) {
        System.out.println(Adding.add(67, 67));
        System.out.println(Adding.add(32.4, 12.6));
    }
}
```

GITHUB LINK:

https://github.com/ocmodi21/JAVA/tree/master/Practicals