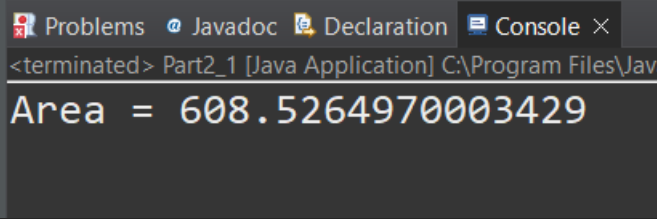


PART-2

Object Oriented Programming: Classes, Methods, Inheritance

GitHub Repository Link: <https://github.com/21ce114/JAVA-Practicals.git>

Question 1:	<p>Design a class named Circle containing following attributes and behavior.</p> <ul style="list-style-type: none"> • 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.
Answer:	<pre> /*ID: 21CE114 Name: Harsh Rana Git Repository Link: https://github.com/21ce114/JAVA-Practicals.git AIM : 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.*/ class Cylinder{ double radius; double height; Cylinder(){ radius = 1; height = 1; } Cylinder(double x){ radius = x; } Cylinder(double x, double y){ radius = x; height = y; } double getArea() { return 2*radius*Math.PI*(radius+height); } } public class Part2_1 { public static void main(String[] args) { </pre>

	<pre> Cylinder c = new Cylinder(6.5,8.4); System.out.println("Area = "+c.getArea()); } } </pre> <p>Output:</p> 
Question 2:	<p>Design a class named Account that contains:</p> <ul style="list-style-type: none"> •A private <u>int</u> data field <u>namedid</u> 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 <u>currentinterest</u> rate (default 7%). <p>Assume all accounts have the same interest rate.</p> <ul style="list-style-type: none"> •A private Date data field named dateCreated that stores the date when <u>theaccount</u> was created. •A no-<u>arg</u> constructor that creates a default account. •A constructor that creates an account with the specified id and initial balance. •The <u>accessor</u> and mutator methods for id, balance, and annualInterestRate. •The <u>accessor</u> method for dateCreated. •A method named getMonthlyInterestRate() that returns the <u>monthlyinterest</u> rate. •A method named getMonthlyInterest() that returns the monthly interest. •A method named withdraw that withdraws a specified amount from <u>theaccount</u>. •A method named deposit that deposits <u>aspecified</u> amount to the account.
Answer:	<pre> /*ID: 21CE114 Name: Harsh Rana Git Repository Link: https://github.com/21ce114/JAVA-Practicals.git AIM : Design a class named Account that contains: •A private <u>int</u> data field <u>namedid</u> 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 <u>currentinterest</u> rate (default 7%). Assume all accounts have the same interest rate. •A private Date data field named dateCreated that stores the date when <u>theaccount</u> was created. •A no-<u>arg</u> constructor that creates a default account. •A constructor that creates an account with the specified id and initial balance. •The <u>accessor</u> and mutator methods for id, balance, and annualInterestRate. •The <u>accessor</u> method for dateCreated. •A method named getMonthlyInterestRate() that returns the <u>monthlyinterest</u> rate. •A method named getMonthlyInterest() that returns the monthly interest. </pre>

```
    •A method named withdraw that withdraws a specified amount from
    theaccount.
    •A method named deposit that deposits aspecified amount to the account.*/
import java.util.*;

class Account{
    private int id;
    private double balance;
    private double annualInterestRate;
    private String datecreated;
    Scanner sc = new Scanner(System.in);
    Account(){
        id=0;
        balance=500;
        annualInterestRate=7;
        datecreated = "20/10/2003";
    }
    Account(int a,double b){
        id=a;
        balance=b;
    }
    public int getId() {
        return id;
    }

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

    public double getBalance() {
        return balance;
    }

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

    public double getAnnualInterestRate() {
        return annualInterestRate;
    }

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

    public String getDatecreated() {
        return datecreated;
    }

    public void setDatecreated(String datecreated) {
        this.datecreated = datecreated;
    }

    public double getMonthlyInterestRate() {
        return (annualInterestRate/100)/12;
    }
    public double getMonthlyInterest() {
        return balance*getMonthlyInterestRate();
    }
}
```

```

        public void withdraw(double wd){
            balance = balance-wd;
        }
        public void deposit(double de){
            balance = balance+de;
        }
    }

    public class Part2_2 {
        public static void main(String[] args) {

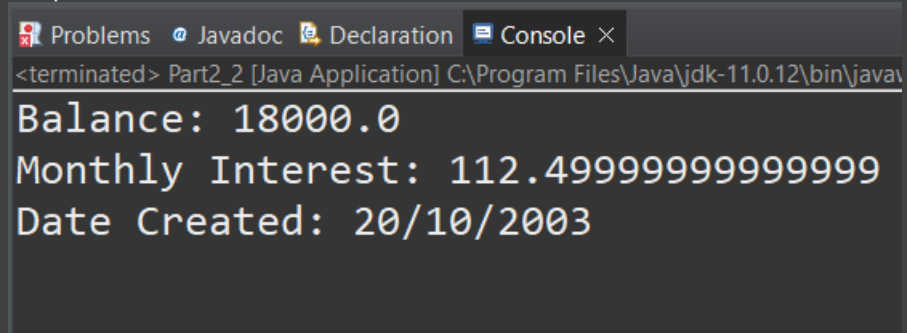
            Account Ac = new Account(114, 20000);
            Ac.setAnnualInterestRate(7.5);
            Ac.withdraw(5000);
            Ac.deposit(3000);
            Ac.setDatecreated("20/10/2003");

            System.out.println("Balance: "+Ac.getBalance());
            System.out.println("Monthly Interest: "+Ac.getMonthlyInterest());
            System.out.println("Date Created: "+Ac.getDatecreated());

        }
    }

```

Output:



```

<terminated> Part2_2 [Java Application] C:\Program Files\Java\jdk-11.0.12\bin\javaw
Balance: 18000.0
Monthly Interest: 112.49999999999999
Date Created: 20/10/2003

```

Question
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

Answer:

```

/*ID: 21CE114
Name: Harsh Rana
Git Repository Link: https://github.com/21ce114/JAVA-Practicals.git
AIM :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 money4.Money Transfer5.
Create Account
6.Deactivate Account
7.Exit */

import java.util.Scanner;

class Account{
    private int id;
    private double balance;
    private double annualInterestRate;
    private String datecreated;
    Account(){
        id=0;
        balance=500;
        annualInterestRate=7;
        datecreated = "20/10/2003";
    }
    Account(int a,double b){
        id=a;
        balance=b;
    }
    public int getId() {
        return id;
    }

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

    public double getBalance() {
        return balance;
    }

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

    public double getAnnualInterestRate() {
        return annualInterestRate;
    }

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

    public String getDatecreated() {
        return datecreated;
    }

    public void setDatecreated(String datecreated) {
        this.datecreated = datecreated;
    }

    public double getMonthlyInterestRate() {
        return (annualInterestRate/100)/12;
    }
    public double getMonthlyInterest() {
        return balance*getMonthlyInterestRate();
    }
    public void withdraw(double wd){
        balance = balance-wd;
    }
    public void deposit(double de){
        balance = balance+de;
    }
}

public class Part2_3 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Account[] acc = new Account[10];

        for(int i=0;i<10;i++) {
            acc[i]=new Account(i+1, 300);
        }
        System.out.println("Enter an ID(From 1-10): ");
        int id = sc.nextInt();

```

```

        if(id<1 || id>10) {
            id= incorrect(id);
        }

        menuDisplay();
        System.out.println("Enter your Choice:");
        int choice = sc.nextInt();

        switch (choice){
            case 1:

                System.out.println("The Balance is: "+acc[id-1].getBalance());
                break;

            case 2:

                System.out.println("Enter Amount to Withdraw: ");
                acc[id-1].withdraw(sc.nextDouble());
                break;

            case 3:

                System.out.println("Enter Amount to Deposit: ");
                acc[id-1].withdraw(sc.nextDouble());
                break;

            case 4:

                System.out.println("Enter account number to transfer money:");
                int id1 = sc.nextInt();
                System.out.println("Enter amount to transfer:");
                double amu= sc.nextDouble();
                acc[id-1].withdraw(amu);
                acc[id1-1].deposit(amu);
                System.out.println("Balance in your account after money
transfer:"+acc[id-1].getBalance());
                System.out.println("Balance in the account you transfered money:
"+acc[id1-1].getBalance());
                break;

            case 5:
                System.out.println("Enter account ID and Balance:");
                int id3= sc.nextInt();
                double bal= sc.nextDouble();
                Account newacc = new Account(id3,bal);
                break;

            case 6:
                System.out.println("To deactivate your account visit nearest bank
branch:");
                break;

            case 7:
                System.out.println("Thank You for using the System.");
                break;

            default:
                break;
        }
    }

    public static int incorrect (int id) {
        Scanner sc = new Scanner(System.in);
        while (id<1 || id>10) {
            System.out.println("Enter valid ID: ");
            id = sc.nextInt();
            System.out.println();
        }
        return id;
    }

    public static void menuDisplay() {
        System.out.printf("%nMain menu%n");
        System.out.println("1: Balance inquiry");
        System.out.println("2: Withdraw money [Maintain minimum balance 300₹]");
        System.out.println("3: Deposit money");
        System.out.println("4: Money Transfer");
        System.out.println("5: Create Account");
        System.out.println("6: Deactivate Account");
        System.out.println("7: exit");
    }
}

```

Output:

```

Problems Javadoc Declaration Console ×
<terminated> Part2_3 [Java Application] C:\Program Files\Java\jdk-11.0.12\bin\javaw.exe (06-Aug-2022, 10:18:08 PM - 10:
Enter an ID(From 1-10):
5

Main menu
1: Balance inquiry
2: Withdraw money [Maintain minimum balance 300₹]
3: Deposit money
4: Money Transfer
5: Create Account
6: Deactivate Account
7: exit
Enter your Choice:
4
Enter account number to transfer money:
6
Enter amount to transfer:
200
Balance in your account after money transfer:100.0
Balance in the account you transfered money: 500.0

```

Question
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.

Answer:

```

/*ID: 21CE114
Name: Harsh Rana
Git Repository Link: https://github.com/21ce114/JAVA-Practicals.git
AIM : (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. */

class Account{
    private int id;

```

```
private double balance;
private double annualInterestRate;
private String datecreated;
Account(){
    id=0;
    balance=500;
    annualInterestRate=7;
    datecreated = "20/10/2003";
}
Account(int a,double b){
    id=a;
    balance=b;
}
public int getId() {
    return id;
}

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

public double getBalance() {
    return balance;
}

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

public double getAnnualInterestRate() {
    return annualInterestRate;
}

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

public String getDatecreated() {
    return datecreated;
}

public void setDatecreated(String datecreated) {
    this.datecreated = datecreated;
}

public double getMonthlyInterestRate() {
    return (annualInterestRate/100)/12;
}
public double getMonthlyInterest() {
    return balance*getMonthlyInterestRate();
}
public void withdraw(double wd){
    balance = balance-wd;
}
public void deposit(double de){
    balance = balance+de;
}
public String toString() {
    return "Account ID:"+getId()+" Account balance"+ getBalance();
}
}

class CheckingAccount extends Account {
    private double overdraftLimit;
```



```
public CheckingAccount() {
    super();
    overdraftLimit = -50;
}

public CheckingAccount(int id, double balance, double overdraftLimit) {
    super(id, balance);
    this.overdraftLimit = overdraftLimit;
}

public void setOverdraftLimit(double overdraftLimit) {
    this.overdraftLimit = overdraftLimit;
}

public double getOverdraftLimit() {
    return overdraftLimit;
}

public void withdraw(double amount) {
    if (getBalance() - amount > overdraftLimit) {
        setBalance(getBalance() - amount);
    }
    else
        System.out.println("Amount exceeds overdraft limit!!");
}

public String toString() {
    return "Checking Account ID: "+getId()+" Checking Account Balance: "+getBalance()+"\nOverdraft limit: " +String.format("%.2f", overdraftLimit);
}

class SavingsAccount extends Account {

    public SavingsAccount() {
        super();
    }

    public SavingsAccount(int id, double balance) {
        super(id, balance);
    }

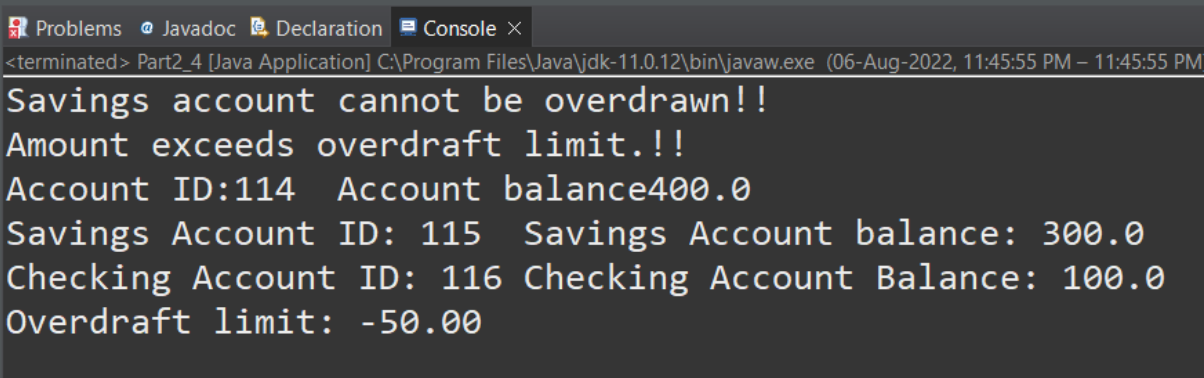
    public void withdraw(double amount) {
        if (amount < getBalance()) {
            setBalance(getBalance() - amount);
        }
        else
            System.out.println("Savings account cannot be overdrawn!!");
    }

    public String toString() {
        return "Savings Account ID: "+getId()+" Savings Account balance: "+getBalance();
    }
}

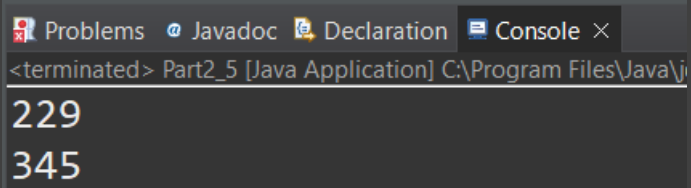
public class Part2_4 {

    public static void main(String[] args) {

        Account account = new Account(114, 500);
        SavingsAccount savings = new SavingsAccount(115, 300);
        CheckingAccount checking = new CheckingAccount(116, 100, -50);
    }
}
```

	<pre> account.withdraw(100); savings.withdraw(500); checking.withdraw(160); System.out.println(account); System.out.println(savings); System.out.println(checking); } } </pre> <p>Output:</p> 
Question 5:	Develop a Program that illustrate method overloading concept.
Answer:	<pre> /*ID: 21CE114 Name: Harsh Rana Git Repository Link: https://github.com/21ce114/JAVA-Practicals.git AIM : Develop a Program that illustrate method overloading concept. */ class Adder{ static int add(int a,int b){ return a+b; } static int add(inta,int b,int c){ return a+b+c; } } class Part2_5{ public static void main(String[] args){ System.out.println(Adder.add(114,115)); System.out.println(Adder.add(114,115,116)); } } </pre>

Output:



The screenshot shows an IDE's console window with tabs for Problems, Javadoc, Declaration, and Console. The console output displays the text "<terminated> Part2_5 [Java Application] C:\Program Files\Java\j" followed by the numbers "229" and "345" on separate lines.

```
<terminated> Part2_5 [Java Application] C:\Program Files\Java\j  
229  
345
```