

Exercise 1

Hariesh R - 23110344

Aim:

The aim of the program is to develop a Java application that calculates and generates electricity bills for consumers based on their usage and type of connection (domestic or commercial) using specified tariff rates.

Algorithm:

If totalReading < 0:

Return amount

If connectionType is "domestic":

If totalReading > 500:

amount += (totalReading - 500) * 6

totalReading = 500

If totalReading > 200:

amount += (totalReading - 200) * 4

totalReading = 200

If totalReading > 100:

amount += (totalReading - 100) * 2.5

totalReading = 100

If totalReading > 0:

amount += totalReading * 1

Else If connectionType is "commercial":

If totalReading <= 100:

Return totalReading * 2

If totalReading <= 200:

Return (100 * 2) + (totalReading - 100) * 4.5

If totalReading <= 500:

Return $(100 * 2) + (100 * 4.5) + (totalReading - 200) * 6$

Return $(100 * 2) + (100 * 4.5) + (300 * 6) + (totalReading - 500) * 7$

Return amount

Code:

```
import java.util.Scanner;
```

```
class eb {
```

```
    static int consumerNo, prevMonthReading, currentMonthReading;
```

```
    static String consumerName, connectionType;
```

```
    static double calculateTotalAmount(){
```

```
        connectionType = connectionType.trim();
```

```
        int totalReading = currentMonthReading - prevMonthReading;
```

```
        double amount = 0;
```

```
        if(totalReading < 0){
```

```
            return amount;
```

```
        }
```

```
        if(connectionType.equalsIgnoreCase("domestic")){
```

```
            if(totalReading > 500){
```

```
    amount += ((500 - totalReading) * -1) * 6;  
    totalReading = 500;  
}
```

```
if(totalReading > 200){
```

```
    amount += ((200 - totalReading) * -1) * 4;  
    totalReading = 200;  
}
```

```
if(totalReading > 100){
```

```
    amount += ((100 - totalReading) * -1) * 2.5;  
    totalReading = 100;  
}
```

```
if(totalReading > 0)
```

```
    amount += totalReading;  
}
```

```
else if(connectionType.equalsIgnoreCase("commercial")){
```

```
    if(totalReading <= 100)  
        return totalReading * 2;
```

```
    else if(totalReading <= 200)  
        return (100 * 2) + (totalReading - 100) * 4.5;
```

```
    else if(totalReading <= 500)  
        return (100 * 2) + (100 * 4.5) + (totalReading - 200) * 6;
```

```

        else
            return (100 * 2) + (100 * 4.5) + (300 * 6) + (totalReading - 500) * 7;
    }

    return amount;
}

public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter Consumer No: ");
    consumerNo = scanner.nextInt();
    scanner.nextLine();

    System.out.print("Enter Name: ");
    consumerName = scanner.nextLine();

    System.out.print("Enter Previous Month Reading: ");
    prevMonthReading = scanner.nextInt();
    scanner.nextLine();

    System.out.print("Enter Current Month Reading: ");
    currentMonthReading = scanner.nextInt();
    scanner.nextLine();

    System.out.print("Enter Type of EB Connection: ");
    connectionType = scanner.nextLine();
}

```

```
scanner.close();

double result = calculateTotalAmount();

System.out.println("Total Amount To Be Paid: " + result);
}
}
```

Output:

```
PS C:\Visual Studio Code\JAVA> javac eb.java
PS C:\Visual Studio Code\JAVA> java eb
Enter Consumer No: 69
Enter Name: Hariesh
Enter Previous Month Reading: 5
Enter Current Month Reading: 1007
Enter Type of EB Connection: domestic
Total Amount To Be Paid: 4562.0
PS C:\Visual Studio Code\JAVA> java eb
Enter Consumer No: 12
Enter Name: Hariesh
Enter Previous Month Reading: 5
Enter Current Month Reading: 1007
Enter Type of EB Connection: commercial
Total Amount To Be Paid: 5964.0
PS C:\Visual Studio Code\JAVA> java eb
Enter Consumer No: 99
Enter Name: Hariesh
Enter Previous Month Reading: 5
Enter Current Month Reading: 1007
Enter Type of EB Connection: random
Total Amount To Be Paid: 0.0
PS C:\Visual Studio Code\JAVA> █
```

Exercise 2

Hariesh R - 23110344

Aim:

The aim is to develop a Java application that performs conversions for currency (Dollar, Euro, and Yen to INR and vice versa), distance (meters to kilometers, miles to kilometers, and vice versa), and time (hours to minutes, seconds, and vice versa), utilizing packages for better organization and modularity.

Algorithm:

1. Package converter.currency

- a. Start
- b. Define the currency_converter class in Exercise2.converter.currency.
- c. Implement conversion methods:
 - i. dollar_to_inr(value): Return value * 83.73.
 - ii. inr_to_dollar(value): Return value * 0.012.
 - iii. euro_to_inr(value): Return value * 90.86.
 - iv. inr_to_euro(value): Return value * 0.011.
 - v. yen_to_inr(value): Return value * 0.54.
 - vi. inr_to_yen(value): Return value * 1.84.
- d. End

2. Package converter.distance

- a. Start
- b. Define the distance_converter class in Exercise2.converter.distance.
- c. Implement conversion methods:

- i. meter_to_km(value): Return value * 0.001.
 - ii. km_to_meter(value): Return value * 1000.
 - iii. mile_to_km(value): Return value * 1.60934.
 - iv. km_to_mile(value): Return value * 0.621371.
- d. End

3. Package converter.time

- a. Start
- b. Define the time_converter class in Exercise2.converter.time.
- c. Implement conversion methods:
 - i. hour_to_minutes(value): Return value * 60.
 - ii. minute_to_hour(value): Return value / 60.
 - iii. hour_to_second(value): Return value * 3600.
 - iv. second_to_hour(value): Return value / 3600.
 - v. minute_to_second(value): Return value * 60.
 - vi. second_to_minute(value): Return value / 60.
- d. End

4. Package main_program

- a. Start
- b. Import necessary classes.
- c. Create main_program class with main method:
 - i. Initialize a Scanner for user input.
 - ii. Set exit flag to false.
- d. Loop until exit is true:
 - i. Display main menu options.
 - ii. Get user's choice.
- e. Handle user choice:

- f. Case 1: Currency Converter
 - i. Display currency options.
 - ii. Get conversion choice and amount.
 - iii. Perform conversion based on choice.
 - iv. Print converted amount.
- g. Case 2: Distance Converter
 - i. Display distance options.
 - ii. Get conversion choice and amount.
 - iii. Perform conversion based on choice.
 - iv. Print converted distance.
- h. Case 3: Time Converter
 - i. Display time options.
 - ii. Get conversion choice and amount.
 - iii. Perform conversion based on choice.
 - iv. Print converted time.
- i. Case 4: Exit
 - i. Print exit message.
 - ii. Set exit to true and close Scanner.
- j. Default:
 - i. Print "Invalid choice" message.
- k. End

Source Code:

1. Package converter.currency:

```
package Exercise2.converter.currency;

public class currency_converter {

    public static double dollar_to_inr(double value){

        return value * 83.73;
    }
}
```



```

public static double inr_to_dollar(double value){

    return value * 0.012;
}

public static double euro_to_inr(double value){

    return value * 90.86;
}

public static double inr_to_euro(double value){

    return value * 0.011;
}

public static double yen_to_inr(double value){

    return value * 0.54;
}

public static double inr_to_yen(double value){

    return value * 1.84;
}
}

```

2. Package converter.distance

```

package Exercise2.converter.distance;

public class distance_converter {

    public static double meter_to_km(double value){

        return value * 0.001;
    }

    public static double km_to_meter(double value){

        return value * 1000;
    }

    public static double mile_to_km(double value){

        return value * 1.60934;
    }
}

```

```
}

public static double km_to_mile(double value){

    return value * 0.621371;
}
}
```

3. Package converter.time

```
package Exercise2.converter.time;

public class time_converter {

    public static double hour_to_minutes(double value){

        return value * 60;
    }

    public static double minute_to_hour(double value){

        return value / 60;
    }

    public static double hour_to_second(double value){

        return value * 3600;
    }

    public static double second_to_hour(double value){

        return value / 3600;
    }

    public static double minute_to_second(double value){

        return value * 60;
    }

    public static double second_to_minute(double value){

        return value / 60;
    }
}
```

4. Package main_program

```
package Exercise2.main_program;

import Exercise2.converter.currency.currency_converter;
import Exercise2.converter.distance.distance_converter;
import Exercise2.converter.time.time_converter;

import java.util.Scanner;

public class main_program {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        boolean exit = false;

        while (!exit) {

            System.out.println("Select conversion type:");
            System.out.println("1. Currency Converter");
            System.out.println("2. Distance Converter");
            System.out.println("3. Time Converter");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            switch (choice) {

                case 1:
                    System.out.println("Currency Converter:");
                    System.out.println("1. Dollar to INR");
                    System.out.println("2. INR to Dollar");
                    System.out.println("3. Euro to INR");
                    System.out.println("4. INR to Euro");
                    System.out.println("5. Yen to INR");
                    System.out.println("6. INR to Yen");

                    System.out.print("Enter your choice: ");
                    int currencyChoice = scanner.nextInt();

                    System.out.print("Enter amount to convert: ");
                    double amount = scanner.nextDouble();

                    double convertedAmount = 0;
```

```

switch (currencyChoice) {

    case 1:
        convertedAmount = currency_converter.dollar_to_inr(amount);
        break;

    case 2:
        convertedAmount = currency_converter.inr_to_dollar(amount);
        break;

    case 3:
        convertedAmount = currency_converter.euro_to_inr(amount);
        break;

    case 4:
        convertedAmount = currency_converter.inr_to_euro(amount);
        break;

    case 5:
        convertedAmount = currency_converter.yen_to_inr(amount);
        break;

    case 6:
        convertedAmount = currency_converter.inr_to_yen(amount);
        break;
}

```

```

System.out.println("Amount converted: " + convertedAmount);
break;

```

```

case 2:
    System.out.println("Distance Converter:");
    System.out.println("1. Meter to KM");
    System.out.println("2. KM to Meter");
    System.out.println("3. Miles to KM");
    System.out.println("4. KM to Miles");

    System.out.print("Enter your choice: ");
    int distanceChoice = scanner.nextInt();

    System.out.print("Enter amount to convert: ");
    double distance = scanner.nextDouble();
    double convertedDistance = 0;

    switch (distanceChoice) {

```

```

    case 1:
        convertedDistance = distance_converter.meter_to_km(distance);
        break;

    case 2:
        convertedDistance = distance_converter.km_to_meter(distance);
        break;

    case 3:
        convertedDistance = distance_converter.mile_to_km(distance);
        break;

    case 4:
        convertedDistance = distance_converter.km_to_mile(distance);
        break;
}

System.out.println("Distance converted: " + convertedDistance);
break;

```

```

case 3:
    System.out.println("Time Converter:");
    System.out.println("1. Hour to Minute");
    System.out.println("2. Minute to Hour");
    System.out.println("3. Hour to Second");
    System.out.println("4. Second to Hour");
    System.out.println("5. Minute to Second");
    System.out.println("6. Second to Minute");

    System.out.print("Enter your choice: ");
    int timeChoice = scanner.nextInt();

    System.out.print("Enter amount to convert: ");
    double time = scanner.nextDouble();
    double convertedTime = 0;

    switch (timeChoice) {

        case 1:
            convertedTime = time_converter.hour_to_minutes(time);
            break;

        case 2:
            convertedTime = time_converter.minute_to_hour(time);
            break;

```

```

        case 3:
            convertedTime = time_converter.hour_to_second(time);
            break;

        case 4:
            convertedTime = time_converter.second_to_hour(time);
            break;

        case 5:
            convertedTime = time_converter.minute_to_second(time);
            break;

        case 6:
            convertedTime = time_converter.second_to_minute(time);
            break;
    }

    System.out.println("Time converted: " + convertedTime);
    break;

case 4:
    System.out.println("Exited !");
    exit = true;
    scanner.close();
    break;

default:
    System.out.println("Invalid choice");
    break;
    }
    }
    }
}

```

Output:

Select conversion type:

1. Currency Converter
2. Distance Converter
3. Time Converter
4. Exit

Enter your choice: 1

Currency Converter:

1. Dollar to INR
2. INR to Dollar
3. Euro to INR
4. INR to Euro
5. Yen to INR
6. INR to Yen

Enter your choice: 1

Enter amount to convert: 5

Amount converted: 418.650000000000003

Select conversion type:

1. Currency Converter
2. Distance Converter
3. Time Converter
4. Exit

Enter your choice: 1

Currency Converter:

1. Dollar to INR
2. INR to Dollar
3. Euro to INR
4. INR to Euro
5. Yen to INR
6. INR to Yen

Enter your choice: 6

Enter amount to convert: 675

Amount converted: 1242.0

Select conversion type:

1. Currency Converter
2. Distance Converter
3. Time Converter
4. Exit

Enter your choice: 2

Distance Converter:

1. Meter to KM
2. KM to Meter
3. Miles to KM
4. KM to Miles

Enter your choice: 2

Enter amount to convert: 8

Distance converted: 8000.0

Select conversion type:

1. Currency Converter
2. Distance Converter
3. Time Converter
4. Exit

Enter your choice: 2

Distance Converter:

1. Meter to KM
2. KM to Meter
3. Miles to KM
4. KM to Miles

Enter your choice: 4

Enter amount to convert: 23

Distance converted: 14.291533

Select conversion type:

1. Currency Converter
2. Distance Converter
3. Time Converter
4. Exit

Enter your choice: 3

Time Converter:

1. Hour to Minute
2. Minute to Hour
3. Hour to Second
4. Second to Hour
5. Minute to Second
6. Second to Minute

Enter your choice: 3

Enter amount to convert: 5

Time converted: 18000.0

```
Select conversion type:
1. Currency Converter
2. Distance Converter
3. Time Converter
4. Exit
Enter your choice: 3
Time Converter:
1. Hour to Minute
2. Minute to Hour
3. Hour to Second
4. Second to Hour
5. Minute to Second
6. Second to Minute
Enter your choice: 6
Enter amount to convert: 500
Time converted: 8.333333333333334
Select conversion type:
1. Currency Converter
2. Distance Converter
3. Time Converter
4. Exit
Enter your choice: 4
Exited !|

Process finished with exit code 0
```

Result:

The Java application performs currency, distance, and time conversions. It converts between Dollar, Euro, Yen, and INR; meters, kilometers, and miles; and hours, minutes, and seconds. Users select the conversion type and input values via a command-line menu, and the program provides the converted results, all organized into separate packages for modularity.

Exercise 3

Hariesh R - 23110344

Aim:

The aim of this Java application is to create an employee management system for an academic institution. It models employee details and salary calculations, including various allowances and deductions, and generates pay slips for different types of employees like Programmers, Assistant Professors, Associate Professors, and Professors.

Algorithm:

1. Employee.java

- a. Define Employee Class:
 - i. Declare EmployeeName, EmployeeID, EmployeeAddress, EmployeeMailId, and EmployeeMobileNo as member variables.
- b. Constructor Initialization:
 - i. Create a constructor that takes EmployeeID, EmployeeName, EmployeeAddress, EmployeeMailId, and EmployeeMobileNo as parameters.
 - ii. Initialize the member variables with the constructor parameters.

2. Emp_children.java

- a. Define Inherited Classes:
 - i. Create Programmer, Assistant_Professor, Associate_Professor, and Professor classes extending Employee.
- b. Declare Salary Components:
 - i. Declare basicPay, DA, HRA, PF, staffClubFund, grossSalary, and netSalary as member variables in each class.
- c. Constructor Initialization:

- i. Create a constructor for each class that takes employee details and basicPay as parameters.
 - ii. Use super to call the Employee constructor.
 - iii. Initialize basicPay and calculate:
 - 1. DA as 97% of basicPay.
 - 2. HRA as 10% of basicPay.
 - 3. PF as 12% of basicPay.
 - 4. staffClubFund as 0.1% of basicPay.
 - 5. grossSalary as the sum of basicPay, DA, and HRA.
 - 6. netSalary as basicPay minus PF and staffClubFund.
- d. Generate Pay Slip:
 - i. Define generatePaySlip method to print employee details and salary components.
- e. Repeat for Each Class:
 - i. Repeat steps 1-4 for Programmer, Assistant_Professor, Associate_Professor, and Professor classes.

3. Main_program.java

- i. Initialize Scanner:
 - 1. Create a Scanner object to read input.
- ii. Prompt User for Employee Details:
 - 1. Read and store EmployeeName, EmployeeID, EmployeeAddress, EmployeeMailId, EmployeeMobileNo, and basicPay from user input.
- iii. Select Employee Type:
 - 1. Display options for employee types: Programmer, Assistant Professor, Associate Professor, Professor.
 - 2. Read the selected employee type from user input.
- iv. Create and Generate Pay Slip:
 - 1. Use a switch statement to create an object of the selected employee type.
 - 2. Call generatePaySlip method for the created object to print the pay slip.

- v. Close Scanner:
 - 1. Close the Scanner object.

Code:

1. Employee.java

```
package Exercise3;

public class Employee {

    String EmployeeName;
    int EmployeeID;
    String EmployeeAddress;
    String EmployeeMailId;
    long EmployeeMobileNo;

    public Employee(int EmployeeID, String EmployeeName, String EmployeeAddress,
String EmployeeMailId, long EmployeeMobileNo) {
        this.EmployeeID = EmployeeID;
        this.EmployeeName = EmployeeName;
        this.EmployeeAddress = EmployeeAddress;
        this.EmployeeMailId = EmployeeMailId;
        this.EmployeeMobileNo = EmployeeMobileNo;
    }
}
```

2. Emp_children.java

```
package Exercise3;

class Programmer extends Employee{
```

```

        double basicPay, DA, HRA, PF, staffClubFund, grossSalary, netSalary;

        Programmer(int EmployeeID, String EmployeeName, String EmployeeAddress,
String EmployeeMailId, long EmployeeMobileNo, double basicPay) {
            super(EmployeeID, EmployeeName, EmployeeAddress, EmployeeMailId,
EmployeeMobileNo);
            this.basicPay = basicPay;

            this.DA = 0.97 * basicPay;
            this.HRA = 0.10 * basicPay;
            this.PF = 0.12 * basicPay;
            this.staffClubFund = 0.01 * basicPay;
            this.grossSalary = basicPay + this.DA + this.HRA;
            this.netSalary = basicPay - this.PF - this.staffClubFund;
        }

        public void generatePaySlip(){

            System.out.println("Pay Slip for: " + EmployeeName);
            System.out.println("Employee ID: " + EmployeeID);
            System.out.println("Address: " + EmployeeAddress);
            System.out.println("Email: " + EmployeeMailId);
            System.out.println("Mobile No: " + EmployeeMobileNo);
            System.out.println("Basic Pay: $" + basicPay);
            System.out.println("DA (97% of BP): $" + DA);
            System.out.println("HRA (10% of BP): $" + HRA);
            System.out.println("PF (12% of BP): $" + PF);
            System.out.println("Staff Club Fund (1% of BP): $" + staffClubFund);
            System.out.println("Gross Salary: $" + grossSalary);
            System.out.println("Net Salary: $" + netSalary);
            System.out.println("-----");
        }
    }

    class Assistant_Professor extends Employee{

        double basicPay, DA, HRA, PF, staffClubFund, grossSalary, netSalary;

        Assistant_Professor(int EmployeeID, String EmployeeName, String
EmployeeAddress, String EmployeeMailId, long EmployeeMobileNo, double
basicPay) {
            super(EmployeeID, EmployeeName, EmployeeAddress, EmployeeMailId,
EmployeeMobileNo);
            this.basicPay = basicPay;

```

```

        this.DA = 0.97 * basicPay;
        this.HRA = 0.10 * basicPay;
        this.PF = 0.12 * basicPay;
        this.staffClubFund = 0.01 * basicPay;
        this.grossSalary = basicPay + this.DA + this.HRA;
        this.netSalary = basicPay - this.PF - this.staffClubFund;
    }

    public void generatePaySlip(){

        System.out.println("Pay Slip for: " + EmployeeName);
        System.out.println("Employee ID: " + EmployeeID);
        System.out.println("Address: " + EmployeeAddress);
        System.out.println("Email: " + EmployeeMailId);
        System.out.println("Mobile No: " + EmployeeMobileNo);
        System.out.println("Basic Pay: $" + basicPay);
        System.out.println("DA (97% of BP): $" + DA);
        System.out.println("HRA (10% of BP): $" + HRA);
        System.out.println("PF (12% of BP): $" + PF);
        System.out.println("Staff Club Fund (1% of BP): $" + staffClubFund);
        System.out.println("Gross Salary: $" + grossSalary);
        System.out.println("Net Salary: $" + netSalary);
        System.out.println("-----");
    }
}

class Associate_Professor extends Employee{

    double basicPay, DA, HRA, PF, staffClubFund, grossSalary, netSalary;

    Associate_Professor(int EmployeeID, String EmployeeName, String
EmployeeAddress, String EmployeeMailId, long EmployeeMobileNo, double
basicPay) {
        super(EmployeeID, EmployeeName, EmployeeAddress, EmployeeMailId,
EmployeeMobileNo);
        this.basicPay = basicPay;

        this.DA = 0.97 * basicPay;
        this.HRA = 0.10 * basicPay;
        this.PF = 0.12 * basicPay;
        this.staffClubFund = 0.01 * basicPay;
        this.grossSalary = basicPay + this.DA + this.HRA;
        this.netSalary = basicPay - this.PF - this.staffClubFund;
    }

    public void generatePaySlip(){

```



```

        System.out.println("Pay Slip for: " + EmployeeName);
        System.out.println("Employee ID: " + EmployeeID);
        System.out.println("Address: " + EmployeeAddress);
        System.out.println("Email: " + EmployeeMailId);
        System.out.println("Mobile No: " + EmployeeMobileNo);
        System.out.println("Basic Pay: $" + basicPay);
        System.out.println("DA (97% of BP): $" + DA);
        System.out.println("HRA (10% of BP): $" + HRA);
        System.out.println("PF (12% of BP): $" + PF);
        System.out.println("Staff Club Fund (1% of BP): $" + staffClubFund);
        System.out.println("Gross Salary: $" + grossSalary);
        System.out.println("Net Salary: $" + netSalary);
        System.out.println("-----");
    }
}

```

```

class Professor extends Employee{

```

```

    double basicPay, DA, HRA, PF, staffClubFund, grossSalary, netSalary;

```

```

    Professor(int EmployeeID, String EmployeeName, String EmployeeAddress,
String EmployeeMailId, long EmployeeMobileNo, double basicPay) {
        super(EmployeeID, EmployeeName, EmployeeAddress, EmployeeMailId,
EmployeeMobileNo);
        this.basicPay = basicPay;

        this.DA = 0.97 * basicPay;
        this.HRA = 0.10 * basicPay;
        this.PF = 0.12 * basicPay;
        this.staffClubFund = 0.01 * basicPay;
        this.grossSalary = basicPay + this.DA + this.HRA;
        this.netSalary = basicPay - this.PF - this.staffClubFund;
    }

```

```

    public void generatePaySlip(){

```

```

        System.out.println("Pay Slip for: " + EmployeeName);
        System.out.println("Employee ID: " + EmployeeID);
        System.out.println("Address: " + EmployeeAddress);
        System.out.println("Email: " + EmployeeMailId);
        System.out.println("Mobile No: " + EmployeeMobileNo);
        System.out.println("Basic Pay: $" + basicPay);
        System.out.println("DA (97% of BP): $" + DA);
        System.out.println("HRA (10% of BP): $" + HRA);
        System.out.println("PF (12% of BP): $" + PF);
    }
}

```

```

        System.out.println("Staff Club Fund (1% of BP): $" + staffClubFund);
        System.out.println("Gross Salary: $" + grossSalary);
        System.out.println("Net Salary: $" + netSalary);
        System.out.println("-----");
    }
}

```

3. Main_program.java

```

package Exercise3;
import java.util.Scanner;

public class main_program {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the Employee Name: ");
        String empName = scanner.nextLine();

        System.out.print("Enter the Employee ID: ");
        int empId = scanner.nextInt();
        scanner.nextLine();

        System.out.print("Enter the Address: ");
        String address = scanner.nextLine();

        System.out.print("Enter the Mail ID: ");
        String mailId = scanner.nextLine();

        System.out.print("Enter the Mobile No: ");
        long mobileNo = scanner.nextLong();
        scanner.nextLine();

        System.out.print("Enter the Basic Pay: ");
        double basicPay = scanner.nextDouble();

        System.out.println("Enter the type of Employee: ");
        System.out.println("1. Programmer");
        System.out.println("2. Assistant Professor");
        System.out.println("3. Associate Professor");
        System.out.println("4. Professor");
        System.out.print("Enter: ");
        int type = scanner.nextInt();
    }
}

```

```

scanner.nextLine();

switch (type) {

    case 1:
        Programmer programmer = new Programmer(empId,empName, address,
mailId, mobileNo, basicPay);
        programmer.generatePaySlip();
        break;

    case 2:
        Assistant_Professor assistant_Professor = new
Assistant_Professor(empId,empName, address, mailId, mobileNo, basicPay);
        assistant_Professor.generatePaySlip();
        break;

    case 3:
        Associate_Professor associate_Professor = new Associate_Professor(empId,
empName, address, mailId, mobileNo, basicPay);
        associate_Professor.generatePaySlip();
        break;

    case 4:
        Professor professor = new Professor(empId, empName, address, mailId,
mobileNo, basicPay);
        professor.generatePaySlip();
        break;

    default:
        System.out.println("Invalid type of Employee");
        break;
}

scanner.close();
}
}

```

Output:

Enter the Employee Name: *Hariesh*
Enter the Employee ID: *12345*
Enter the Address: *something*
Enter the Mail ID: *smth@gmail.com*
Enter the Mobile No: *1234567890*
Enter the Basic Pay: *7000*
Enter the type of Employee:

1. Programmer
2. Assistant Professor
3. Associate Professor
4. Professor

Enter: *1*

Pay Slip for: Hariesh

Employee ID: 12345

Address: something

Email: smth@gmail.com

Mobile No: 1234567890

Basic Pay: \$7000.0

DA (97% of BP): \$6790.0

HRA (10% of BP): \$700.0

PF (12% of BP): \$840.0

Staff Club Fund (1% of BP): \$70.0

Gross Salary: \$14490.0

Net Salary: \$6090.0

Process finished with exit code 0

```
Enter the Employee Name: User
Enter the Employee ID: 3442
Enter the Address: adasd
Enter the Mail ID: asdsdasa
Enter the Mobile No: 123233424
Enter the Basic Pay: 12000
Enter the type of Employee:
1. Programmer
2. Assistant Professor
3. Associate Professor
4. Professor
Enter: 3
Pay Slip for: User
Employee ID: 3442
Address: adasd
Email: asdsdasa
Mobile No: 123233424
Basic Pay: $12000.0
DA (97% of BP): $11640.0
HRA (10% of BP): $1200.0
PF (12% of BP): $1440.0
Staff Club Fund (1% of BP): $120.0
Gross Salary: $24840.0
Net Salary: $10440.0
-----

Process finished with exit code 0
```

Result:

To develop a Java application, start with an Employee class containing Emp_name, Emp_id, Address, Mail_id, and Mobile_no. Inherit this class into Programmer, AssistantProfessor, AssociateProfessor, and Professor classes, each adding a BasicPay attribute. Calculate salaries where DA is 97% of Basic Pay, HRA is 10%, PF is 12%, and staff club fund is 0.1%. For each employee, compute gross salary (Basic Pay + DA + HRA) and net salary (Gross Salary - PF - Staff Club Fund). Generate payslips displaying the gross and net salary for employees of each class.

Exercise 4

Hariesh R - 23110344

Aim:

The program illustrates key Java concepts: defining an abstract class Shape with a method area() and implementing it in Rectangle, Triangle, and Circle to print their areas. It also covers method overriding by providing specific implementations in subclasses and method overloading by using the same method name with different parameters.

Algorithm:

1. i) Shape.java

- Define an abstract class Shape.
- Declare two protected integers integer1 and integer2.
- Initialize these integers via a constructor.
- Declare an abstract method area() for calculating the shape's area.

ii) Rectangle.java

- Define class Rectangle extending Shape.
- Implement constructor to accept width and height, and pass them to the superclass constructor.
- Override area() method to calculate and print the rectangle's area using width * height.

iii) Triangle.java

- Define class Triangle extending Shape.
- Implement constructor to accept base and height, and pass them to the superclass constructor.
- Override area() method to calculate and print the triangle's area using $0.5 * \text{base} * \text{height}$.

iv) Circle.java

- Define class Circle extending Shape.
- Declare constant PI for π .
- Implement constructor to accept radius and pass it to the superclass constructor.
- Override area() method to calculate and print the circle's area using $\pi * \text{radius}^2$.

v) Main.java

- Create instances of Rectangle, Circle, and Triangle using Shape references.
- Call the area() method on each shape instance to print the area of each shape.

2. Main.java

- Define class Animal with method makeSound() printing "Animal is making sound".
- Define class Dog extending Animal, override makeSound() to print "Dog is making sound".
- Define class Cat extending Animal, override makeSound() to print "Cat is making sound".
- In the Main class:
 - Create instances of Animal, Dog, and Cat.
 - Call makeSound() on each instance to demonstrate method overriding.

3. Main.java

- Define class Calculator with multiple add() methods:
 - add(int a, int b): Adds two integers.
 - add(int a, int b, int c): Adds three integers.
 - add(double a, double b): Adds two doubles.
 - add(int[] numbers): Adds elements of an integer array.
- In the Main class:
 - Create an instance of Calculator.
 - Demonstrate method overloading by calling each version of add() with different parameters and print the results.

Code:

1. i) Shape.java

```
package Exercise4.Q1;

public abstract class Shape {

    protected int integer1;
    protected int integer2;

    public Shape(int integer1, int integer2){
        this.integer1 = integer1;
        this.integer2 = integer2;
    }

    abstract void area();
}
```

ii) Rectangle.java

```
package Exercise4.Q1;

public class Rectangle extends Shape{
    public Rectangle(int width, int height){
        super(width, height);
    }
    @Override
    public void area(){

        int area = integer1 * integer2;
        System.out.println("Area of Rectangle: " + area);
    }
}
```

iii) Triangle.java

```
package Exercise4.Q1;
```

```

public class Triangle extends Shape{
    public Triangle(int base, int height){
        super(base, height);
    }

    @Override
    public void area(){

        double area = integer1 * integer2 * 0.5;
        System.out.println("Area of Triangle is: " + area);
    }
}

```

iv) Circle.java

```

package Exercise4.Q1;

```

```

public class Circle extends Shape{

    public static final double PI = 3.14159;

    public Circle(int radius){

        super(radius, 0);
    }

    @Override
    public void area(){

        double area = integer1 * integer1 * PI;
        System.out.println("Area of Circle: " + area);
    }
}

```

v) Main.java

```

package Exercise4.Q1;

```

```

public class Main {

```

```

        public static void main(String[] args) {
            Shape rectangle = new Rectangle(5, 10);
            Shape circle = new Circle(5);
            Shape triangle = new Triangle(5, 15);
            rectangle.area();
            circle.area();
            triangle.area();
        }
    }
}

```

2. Main.java

```

package Exercise4.Q2;

class Animal{

    void makeSound(){

        System.out.println("Animals is making sound");
    }
}

class Dog extends Animal{

    @Override
    void makeSound(){

        System.out.println("Dog is making sound");
    }
}

class Cat extends Animal{

    @Override
    void makeSound(){
        System.out.println("Cat is making sound");
    }
}

```

```

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Animal dog = new Dog();
        Animal cat = new Cat();

        animal.makeSound();
        dog.makeSound();
        cat.makeSound();
    }
}

```

3. Main.java

```

package Exercise4.Q3;

class Calculator{

    int add(int a, int b){
        return a + b;
    }

    int add(int a, int b, int c){
        return a + b + c;
    }

    double add(double a, double b){

        return a + b;
    }
    int add(int[] numbers){

        int sum = 0;

        for(int i:numbers){

            sum += i;
        }
    }
}

```

```

        return sum;
    }
}

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

        Calculator calc = new Calculator();

        System.out.println("Sum of 5 and 10: " + calc.add(5, 10));
        System.out.println("Sum of 5, 10, and 15: " + calc.add(5, 10, 15));
        System.out.println("Sum of 5.5 and 10.5: " + calc.add(5.5, 10.5));
        System.out.println("Sum of array elements: " + calc.add(new int[]{1, 2, 3,
4, 5}));
    }
}

```

Output:

1.

```

snu-cse@snu-cse-HP-ProDesk-400-G7-Microtower-PC:~/harizzesh$ java Exercise4.Q1.Main
Area of Rectangle: 50
Area of Circle: 78.53975
Area of Triangle is: 37.5
snu-cse@snu-cse-HP-ProDesk-400-G7-Microtower-PC:~/harizzesh$

```

2.

```

snu-cse@snu-cse-HP-ProDesk-400-G7-Microtower-PC:~/harizzesh$ java Exercise4.Q2.Main
Animals is making sound
Dog is making sound
Cat is making sound
snu-cse@snu-cse-HP-ProDesk-400-G7-Microtower-PC:~/harizzesh$

```

3.

```

snu-cse@snu-cse-HP-ProDesk-400-G7-Microtower-PC:~/harizzesh$ java Exercise4.Q3.Main
Sum of 5 and 10: 15
Sum of 5, 10, and 15: 30
Sum of 5.5 and 10.5: 16.0
Sum of array elements: 15
snu-cse@snu-cse-HP-ProDesk-400-G7-Microtower-PC:~/harizzesh$

```

Result:

The program will demonstrate area calculations for different shapes and method behaviors. For the Shape abstract class and its subclasses Rectangle, Triangle, and Circle, the output will show the respective areas of these shapes. The method overriding example will print specific sounds for Animal, Dog, and Cat. The method overloading example will show how different add() methods handle integers, doubles, and integer arrays, displaying sums accordingly.

Exercise 5

Hariesh R - 23110344

Aim:

To demonstrate string manipulation using an ArrayList, including appending, inserting, searching, filtering by starting letter, and sorting strings in ascending and descending order.

Algorithm:

1. Initialize:
 - a. Create an ArrayList<String> to store strings.
 - b. Use a Scanner object to handle user input.
2. Display Menu:
 - a. Continuously display a menu with options to perform various string operations until the user chooses to exit.
3. Handle User Choice:
 - a. Append a string:
 - i. Prompt the user to enter a string and add it to the end of the ArrayList.
 - b. Insert a string at a specific index:
 - i. Prompt the user for an index and a string, then insert the string at the given index in the ArrayList.
 - c. Search for a string:
 - i. Prompt the user for a string and search the ArrayList for it, returning the index if found, or -1 if not.
 - d. List strings starting with a given letter:
 - i. Prompt the user for a letter and print all strings in the ArrayList that start with that letter.
 - e. Sort strings:
 - i. Prompt the user to choose between ascending or descending order, sort the ArrayList accordingly, and print the sorted list.

- f. Print the list:
 - i. Print all strings currently in the ArrayList.
 - g. Exit:
 - i. Exit the program.
- 4. Loop until Exit:
 - a. Continue to display the menu and handle user input until the user chooses the "Exit" option.
- 5. Terminate Program:
 - a. Close the Scanner and end the program when the user selects "Exit."

Source Code:

```
package Exercise5;

import java.util.ArrayList;
import java.util.Scanner;
import java.util.Collections;

class methods{

    public static void printMenu(){

        System.out.println("\n1. Append");
        System.out.println("2. Insert");
        System.out.println("3. Search");
        System.out.println("4. List all string starts with given letter");
        System.out.println("5. Sort the strings in ascending and descending order");
        System.out.println("6. Print the list");
        System.out.println("7. Exit");
```



```
        System.out.print("Enter: ");  
    }
```

```
public static void addElement(ArrayList<String> stringArrayList, String dataString){  
  
    stringArrayList.add(dataString);  
}
```

```
public static void insertElement(ArrayList<String> stringArrayList, String dataString, int  
index){  
  
    stringArrayList.add(index, dataString);  
}
```

```
public static int search(ArrayList<String> stringArrayList, String dataString){  
  
    int index = 0;  
  
    for (String stringElement : stringArrayList) {  
  
        if(stringElement.equals(dataString)) return index;  
        index++;  
    }  
  
    return -1;  
}
```

```
public static void startsWithLetter(ArrayList<String> stringArrayList, String dataString){  
  
    for (String stringElement : stringArrayList) {
```

```
        if(stringElement.startsWith(dataString)){

            System.out.println(stringElement);

        }

    }

}
```

```
public static ArrayList<String> sortStrings(ArrayList<String> stringArrayList, boolean ascending){
```

```
    ArrayList<String> newArrayList = new ArrayList<String>();
    //    ArrayList<String> newArrayList = new ArrayList<String>(stringArrayList);
```

```
    for (String element: stringArrayList) newArrayList.add(element);
```

```
    if(ascending)
        Collections.sort(newArrayList);
```

```
    else
        Collections.sort(newArrayList, Collections.reverseOrder());
```

```
    return newArrayList;
}
```

```
public static void printList(ArrayList<String> stringArrayList){
```

```
    for (String iString : stringArrayList) {
        System.out.print(iString + " ");
    }
}
```

```
}
```

```
public class main_program {
```

```
    static int choice;
```

```
    static String data;
```

```
    static int index;
```

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

```
        Scanner scanner = new Scanner(System.in);
```

```
        ArrayList<String> list = new ArrayList<String>();
```

```
        while (true) {
```

```
            methods.printMenu();
```

```
            choice = scanner.nextInt();
```

```
            scanner.nextLine();
```

```
            switch (choice) {
```

```
                case 1:
```

```
                    System.out.print("Enter the value: ");
```

```
                    data = scanner.nextLine();
```

```
                    methods.addElement(list, data);
```

```
                    break;
```

```
                case 2:
```

```
System.out.print("Enter the index: ");  
index = scanner.nextInt();  
scanner.nextLine();  
System.out.print("Enter the value: ");  
data = scanner.nextLine();  
methods.insertElement(list, data, index);  
  
break;
```

case 3:

```
System.out.print("Enter the value: ");  
data = scanner.nextLine();  
int result = methods.search(list, data);  
System.out.print("Found at: " + result);  
break;
```

case 4:

```
System.out.print("Enter the value: ");  
data = scanner.nextLine();  
methods.startsWithLetter(list, data);  
break;
```

case 5:

```
System.out.println("1. Ascending");  
System.out.println("2. Descending");  
System.out.print("Enter: ");  
int choice2 = scanner.nextInt();  
  
if (choice2 == 1){
```

```
        ArrayList<String> newArray = new ArrayList<>();
        newArray = methods.sortStrings(list, true);
        System.out.print("Sorted Array: ");
        methods.printList(newArray);
    }
```

```
    else if(choice2 == 2){
```

```
        ArrayList<String> newArray = new ArrayList<>();
        newArray = methods.sortStrings(list, false);
        System.out.print("Sorted Array: ");
        methods.printList(newArray);
    }
```

```
    else
```

```
        System.out.println("Invalid Choice");
```

```
    break;
```

```
case 6:
```

```
    for (String iString : list) {
        System.out.print(iString + " ");
    }
    break;
```

```
case 7:
```

```
    System.out.println("Exited !");
    scanner.close();
    return;
```

```
        default:
            System.out.println("Invalid Choice");
            break;
    }
}
}
```

Output:

```
1. Append
2. Insert
3. Search
4. List all string starts with given letter
5. Sort the strings in ascending and descending order
6. Print the list
7. Exit
Enter: 1
Enter the value: hariesh
```

```
1. Append
2. Insert
3. Search
4. List all string starts with given letter
5. Sort the strings in ascending and descending order
6. Print the list
7. Exit
Enter: 1
Enter the value: apple
```

1. Append
2. Insert
3. Search
4. List all string starts with given letter
5. Sort the strings in ascending and descending order
6. Print the list
7. Exit

Enter: *1*

Enter the value: *zebra*

1. Append
2. Insert
3. Search
4. List all string starts with given letter
5. Sort the strings in ascending and descending order
6. Print the list
7. Exit

Enter: *1*

Enter the value: *banana*

1. Append
2. Insert
3. Search
4. List all string starts with given letter
5. Sort the strings in ascending and descending order
6. Print the list
7. Exit

Enter: *6*

hariesh apple zebra banana

1. Append
2. Insert
3. Search
4. List all string starts with given letter
5. Sort the strings in ascending and descending order
6. Print the list
7. Exit

Enter: 2

Enter the index: 2

Enter the value: *hello*

1. Append
2. Insert
3. Search
4. List all string starts with given letter
5. Sort the strings in ascending and descending order
6. Print the list
7. Exit

Enter: 6

hariesh apple hello zebra banana

hariesh apple hello zebra banana

1. Append
2. Insert
3. Search
4. List all string starts with given letter
5. Sort the strings in ascending and descending order
6. Print the list
7. Exit

Enter: 3

Enter the value: *apple*

Found at: 1

Found at: 1

1. Append
2. Insert
3. Search
4. List all string starts with given letter
5. Sort the strings in ascending and descending order
6. Print the list
7. Exit

Enter: 3

Enter the value: *hey*

Found at: -1

1. Append
2. Insert
3. Search
4. List all string starts with given letter
5. Sort the strings in ascending and descending order
6. Print the list
7. Exit

Enter: 4

Enter the value: *a*

apple

1. Append
2. Insert
3. Search
4. List all string starts with given letter
5. Sort the strings in ascending and descending order
6. Print the list
7. Exit

Enter: 5

1. Ascending
2. Descending

Enter: 1

Sorted Array: apple banana hariesh hello zebra

1. Append
2. Insert
3. Search
4. List all string starts with given letter
5. Sort the strings in ascending and descending order
6. Print the list
7. Exit

Enter: 5

1. Ascending
2. Descending

Enter: 2

Sorted Array: zebra hello hariesh banana apple

```
1. Append
2. Insert
3. Search
4. List all string starts with given letter
5. Sort the strings in ascending and descending order
6. Print the list
7. Exit
Enter: 6
harish apple hello zebra banana

1. Append
2. Insert
3. Search
4. List all string starts with given letter
5. Sort the strings in ascending and descending order
6. Print the list
7. Exit
Enter: 7
Exited !

Process finished with exit code 0
```

Result:

This program lets users manipulate an ArrayList of strings through a menu, allowing operations like appending, inserting, searching, filtering by starting letter, sorting, and printing. For example, users can create and modify a list like ["apple", "banana", "cherry"] and view the updated list after each operation before exiting the program.

Exercise 6

Hariesh R - 23110344

Aim:

1. The program retrieves and displays the character at a given index in a string. It prompts the user for the string and index, checks if the index is valid, and then prints the character at that position. If the index is invalid, it notifies the user.
2. The program is to determine if two given strings are *k-anagrams* of each other. Two strings are considered k-anagrams if they have the same length and can be transformed into anagrams by changing at most k characters in one of the strings. The program should evaluate these conditions and return whether the strings meet the criteria for being k-anagrams.
3. The program is to demonstrate the use of the LinkedList class from the Collections Framework by performing insert, delete, and display operations. The program should showcase how to add elements to the linked list, remove elements from it, and print the current state of the list.

Algorithm:

- 1) Q1:
 - a) Create a Scanner for input.
 - b) Read a string from the user.
 - c) Read an index from the user.
 - d) Check if the index is valid for the string:
 - i) Print "Invalid Index" if out of bounds.
 - ii) Print the character at the index if valid.
 - e) Close the Scanner.
- 2) Q2:
 - a) Define invalid_char.
 - b) Check Lengths: Return false if word1 and word2 differ in length.
 - c) Match and Mark: Replace matching characters with invalid_char in both words.
 - d) Count Unmatched: Count characters in word1 that are not invalid_char.
 - e) Return Result: Return true if unmatched count $\leq k$, otherwise false.

- f) Read Inputs: Get word1, word2, and k from the user.
- g) Print Result: Output the result of kAnagrams().
- h) Close Scanner.

3) Q3:

- a) Initialize: Create a LinkedList and a Scanner.
- b) Menu Loop: Continuously display the menu and read user choices.
- c) Handle Choices:
- d) Choice 1: Add user input to the list.
- e) Choice 2: Remove and print the first item from the list.
- f) Choice 3: Print all elements in the list.
- g) Choice 4: Print the list in its current state.
- h) Choice 5: Exit the program and close the scanner.
- i) Print Menu: Define and display the menu options.
- j) Exit: End the loop and close resources when the user selects exit.

Source Code:

Q1)

```
package Exercise6;

import java.util.Scanner;

public class Q1 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the string: ");

        String word = scanner.nextLine();

        System.out.print("Enter the index: ");

        int index = scanner.nextInt();

        scanner.nextLine();

        if(index >= word.length() || index < 0)

            System.out.println("Invalid Index");
```

```

        else

            System.out.println("Character at index " + index + " is " +
word.charAt(index));

        scanner.close();

    }

}

```

Q2)

```

package Exercise6;

import java.util.Scanner;

public class Q2 {

    private static final char invalid_char = '*';

    public static boolean kAnagrams(String word1, String word2, int k){

        if(word1.length() != word2.length()) return false;

        for(int i = 0; i < word1.length(); i++){

            for(int j = 0; j < word2.length(); j++){

                if(word1.charAt(i) == word2.charAt(j)){

                    word1 = word1.substring(0, i) + invalid_char +
word1.substring(i+1);

                    word2 = word2.substring(0, j) + invalid_char +
word2.substring(j+1);

                }

            }

        }

    }

}

```

```

        }
    }

    int count = 0;

    for(int i = 0; i < word1.length(); i++){

        if(word1.charAt(i) != invalid_char) count++;

    }

    return count <= k;
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the word 1:" );
    String word1 = scanner.nextLine();
    System.out.print("Enter the word 2: ");
    String word2 = scanner.nextLine();

    System.out.print("Enter the value of k: ");
    int k = scanner.nextInt();
    scanner.nextLine();

    System.out.println(kAnagrams(word1, word2, k));

    scanner.close();
}

```

```
}
```

Q3)

```
package Exercise6;

import java.util.LinkedList;
import java.util.Scanner;

public class Q3 {
    public static void main(String[] args) {
        LinkedList<String> list = new LinkedList<>();
        Scanner scanner = new Scanner(System.in);

        while(true){

            printMenu();
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch(choice){

                case 1:
                    System.out.print("Enter the value: ");
                    String data = scanner.nextLine();

                    list.add(data);
                    break;
```


case 2:

System.out.println("Removed: " + list.remove());

break;

case 3:

for(String element: list) System.out.print(element + " ");

System.out.println();

break;

case 4:

System.out.println(list);

break;

case 5:

System.out.println("Exited !!");

scanner.close();

return;

default:

System.out.println("Invalid Choice");

break;

}

}

}

public static void printMenu(){

```

        System.out.println("1. Insert");
        System.out.println("2. Remove");
        System.out.println("3. Display");
        System.out.println("4. Print");
        System.out.println("5. Exit");
        System.out.print("Enter: ");

    }
}

```

Output:

Q1)

```

ai_ds-a2@snuce-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzesh$ javac Exercise6/Q1.java
ai_ds-a2@snuce-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzesh$ java Exercise6.Q1
Enter the string: Hariesh
Enter the index: 4
Character at index 4 is e
ai_ds-a2@snuce-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzesh$ █

```

Q2)

```

ai_ds-a2@snuce-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzesh$ javac Exercise6/Q2.java
ai_ds-a2@snuce-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzesh$ java Exercise6.Q2
Enter the word 1:anagram
Enter the word 2: grammer
Enter the value of k: 3
true
ai_ds-a2@snuce-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzesh$ javac Exercise6/Q2.java
ai_ds-a2@snuce-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzesh$ java Exercise6.Q2
Enter the word 1:anagram
Enter the word 2: grammer
Enter the value of k: 2
false
ai_ds-a2@snuce-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzesh$ █

```

Q3)

```
ai_ds-a2@snuce-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzesh$ javac Exercise6/Q3.java
ai_ds-a2@snuce-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzesh$ java Exercise6.Q3
```

```
1. Insert
2. Remove
3. Display
4. Print
5. Exit
Enter: 1
Enter the value: Hariesh
1. Insert
2. Remove
3. Display
4. Print
5. Exit
Enter: 1
Enter the value: Hello
```

```
1. Insert
2. Remove
3. Display
4. Print
5. Exit
Enter: 4
[Hariesh, Hello]
1. Insert
2. Remove
3. Display
4. Print
5. Exit
Enter: 2
Removed: Hariesh
```

```
1. Insert
2. Remove
3. Display
4. Print
5. Exit
Enter: 4
[Hello]
1. Insert
2. Remove
3. Display
4. Print
5. Exit
Enter: 5
Exited !!
```

```
ai_ds-a2@snuce-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzesh$ █
```

Result:

- 1) The Java program retrieves and displays the character at a specified index within a string. It prompts the user for a string and an index, then checks if the index is valid. If valid, it prints the character at that index; if not, it displays an error message.
- 2) The Java program determines if two strings are k-anagrams by checking if they have the same length and can be made anagrams by changing at most k characters. It takes two strings and an integer k as input, then calculates the number of changes needed and prints true if the changes are within the allowed limit, otherwise false.
- 3) The Java program demonstrates operations on a LinkedList by inserting elements, deleting elements, and displaying the list. It shows the list's state after each operation to confirm the changes.

Exercise 7

Hariesh R - 23110344

Aim:

1. The aim of this Java program is to check whether the provided array size is negative and handle it using try and catch blocks. If a negative size is entered, a `NegativeArraySizeException` is thrown, which is caught by the catch block, displaying an appropriate error message. This prevents the program from crashing due to invalid input, ensuring graceful error handling when a negative array size is provided.
2. The aim of this Java program is to create a method that checks if the input age is valid for voting, where the minimum required age is 18. If the input is less than 18, the method throws an exception using the `throws` keyword. The exception is then handled appropriately, ensuring that invalid ages (less than 18) are caught and processed without causing the program to crash. This approach enforces proper age validation using exception handling mechanisms in Java.
3. The aim of this Java program is to demonstrate the use of multiple catch statements to handle different types of exceptions that may occur within a try block. The program handles various exceptions, such as `ArithmeticException` (e.g., division by zero) and `ArrayIndexOutOfBoundsException` (e.g., accessing an invalid array index). Each specific exception is caught by its corresponding catch block, ensuring that the program can manage multiple types of errors gracefully without crashing.
4. The aim of this Java program is to demonstrate exception handling using the `throw` keyword for explicitly throwing exceptions, and the `finally` block to ensure code execution regardless of exceptions. This ensures proper error handling and guarantees that necessary cleanup or final actions are always executed.

Algorithm:

- 1) Q1:
 - i) Start the program.

- ii) Initialize scanner:
 - (a) Create a Scanner object to read user input.
- iii) Prompt for array size:
 - (a) Display a message "Enter size:" to prompt the user for input.
 - (b) Read the integer input for the array size.
- iv) Try block:
 - (a) In the try block, attempt to create an array of the specified size.
 - (b) If the array is successfully created, print a success message along with the array size.
- v) Catch block (exception handling):
 - (a) If a `NegativeArraySizeException` occurs (i.e., if the input size is negative), catch the exception.
 - (b) Print an error message containing the exception details.
- vi) Close scanner:
 - (a) Close the Scanner object to prevent resource leaks.
- vii) End the program.

2) Q2:

- i) Start the program.
- ii) Define `CustomError` class:
 - (a) Create a custom exception class `CustomError` that extends `Exception` and takes a message as input.
- iii) Initialize scanner:
 - (a) Create a Scanner object to read user input.
- iv) Prompt for age:
 - (a) Display a message "Enter the age:" to prompt the user for input.
 - (b) Read the integer input for the age.
- v) Try block:
 - (a) In the try block, call the `checkVotingAge` method, passing the user's input (age) as a parameter.
- vi) Catch block (`CustomError` handling):
 - (a) If a `CustomError` is thrown (age is less than 18), catch the exception and print the error message.
- vii) Method `checkVotingAge`:
 - (a) If the input age is less than 18, throw a `CustomError` with the message "Not Eligible To Vote."
 - (b) Otherwise, print "Eligible To Vote."
- viii) Close scanner:
 - (a) Close the Scanner object to prevent resource leaks.
- ix) End the program.

3) Q3:

- i) Start the program.

- ii) Initialize an array of strings:
 - (a) Create an array inputs containing string values: "100", "abc", "12.34", and null.
- iii) Iterate through the array:
 - (a) For each input string in the array, proceed with the following steps.
- iv) Try block:
 - (a) Attempt to convert the input string to an integer using Integer.parseInt(input).
 - (b) Attempt to divide the converted integer by 0.
 - (c) If no exception occurs, print the result of the division.
- v) Catch block (multiple exceptions):
 - (a) NumberFormatException: If the input is not a valid integer (e.g., "abc", "12.34"), catch the exception and print an error message about invalid number format.
 - (b) ArithmeticException: If a division by zero occurs, catch the exception and print an error message about the arithmetic exception.
 - (c) NullPointerException: If the input is null, catch the exception and print an error message about null pointer access.
 - (d) Exception: Catch any other unexpected exceptions and print a generic error message.
- vi) End the loop and the program.

4) Q4:

- i) Start the program.
- ii) Initialize scanner:
 - (a) Create a Scanner object to read user input.
- iii) Prompt for numerator:
 - (a) Display a message "Enter numerator:" and read the integer input for the numerator.
- iv) Prompt for divisor:
 - (a) Display a message "Enter divisor:" and read the integer input for the divisor.
- v) Close scanner:
 - (a) Close the Scanner object to prevent resource leaks.
- vi) Try block:
 - (a) Call the performDivision method, passing the numerator and divisor as parameters.
 - (b) Catch block (ArithmeticException handling):
 - (c) If an ArithmeticException occurs (e.g., division by zero), catch the exception and print an error message.
- vii) Finally block:
 - (a) Execute the finally block, printing a message indicating it has been executed, regardless of whether an exception was caught.

- viii) Method performDivision:
 - (a) If the divisor is 0, throw an ArithmeticException with the message "Cannot divide by zero."
 - (b) Otherwise, perform the division of the numerator by the divisor and print the result.
- ix) End the program.

Source Code:

1) Q1:

```
package Exercise7;

import java.util.Scanner;

public class Q1 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter size: ");
        int size = scanner.nextInt();
        scanner.nextLine();

        try {

            int[] Array = new int[size];
            System.out.println("Successfully Created !\nSize: " + Array.length);
        }

        catch(NegativeArraySizeException exception){

            System.out.println("Error !\nReceived: " + exception.getMessage());
        }

        scanner.close();
    }
}
```

2) Q2:

```
package Exercise7;
import java.util.Scanner;

class CustomError extends Exception{
```

```

    public CustomError(String message){

        super(message);
    }
}

public class Q2 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the age: ");
        int age = scanner.nextInt();
        scanner.nextLine();

        try{

            checkVotingAge(age);
        }

        catch(CustomError error){

            System.out.println(error.getMessage());
        }

        scanner.close();
    }

    public static void checkVotingAge(int age) throws CustomError{

        if (age < 18){

            throw new CustomError("Not Eligible To Vote");
        }

        else
            System.out.println("Eligible To Vote");
    }
}

```

3) Q3:
package Exercise7;

```

public class Q3 {

    public static void main(String[] args) {

        String[] inputs = {"100", "abc", "12.34", null};

        for (String input : inputs) {

            try {

                int number = Integer.parseInt(input);

                int result = number / 0;

                System.out.println("Result of division: " + result);

            } catch (NumberFormatException e) {

                System.out.println("Error: Invalid number format. " + e.getMessage());

            } catch (ArithmeticException e) {

                System.out.println("Error: Arithmetic exception. " + e.getMessage());

            } catch (NullPointerException e) {

                System.out.println("Error: Null pointer exception. " + e.getMessage());

            } catch (Exception e) {

                System.out.println("Unexpected error: " + e.getMessage());

            }

        }

    }

}

```

4) Q4:

```

package Exercise7;

import java.util.Scanner;

public class Q4{

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
    }

}

```

```

System.out.print("Enter numerator: ");
int numerator = scanner.nextInt();
scanner.nextLine();

System.out.print("Enter divisor: ");
int divisor = scanner.nextInt();
scanner.nextLine();

scanner.close();

try {

    performDivision(numerator, divisor);
} catch (ArithmeticException e) {

    System.out.println("Caught an exception: " + e.getMessage());
} finally {

    System.out.println("Finally block executed.");
}
}

public static void performDivision(int numerator, int divisor) throws
ArithmeticException{

    if (divisor == 0) {

        throw new ArithmeticException("Cannot divide by zero");
    }

    int result = numerator / divisor;
    System.out.println("Result of division: " + result);
}
}

```

Output:

1) Q1:

```
Enter size: 10
Successfully Created !
Size: 10

Process finished with exit code 0
```

```
Enter size: -5
Error !
Received: -5

Process finished with exit code 0
```

2) Q2:

```
Enter the age: 10
Not Eligible To Vote

Process finished with exit code 0
```

```
Enter the age: 19
Eligible To Vote

Process finished with exit code 0
```

3) Q3:

```
Error: Arithmetic exception. / by zero
Error: Invalid number format. For input string: "abc"
Error: Invalid number format. For input string: "12.34"
Error: Invalid number format. Cannot parse null string

Process finished with exit code 0
|
```

4) Q4:

```
Enter numerator: 10
Enter divisor: 0
Caught an exception: Cannot divide by zero
Finally block executed.

Process finished with exit code 0
```

```
Enter numerator: 10
Enter divisor: 5
Result of division: 2
Finally block executed.

Process finished with exit code 0
|
```

Result:

1. The result of the Java program that checks whether the given array size is negative involves handling an exception if a negative size is entered. If the user inputs a non-negative size, the program successfully creates the array and displays a success message with the array size. However, if the user provides a negative size, the program catches the `NegativeArraySizeException` and

displays an error message indicating that the array size cannot be negative. This ensures that the program handles invalid input gracefully without crashing.

2. The result of the Java program that checks voting eligibility involves using a method to determine if the input age is valid for voting. If the age is less than 18, the method throws a CustomError exception, which is then caught and handled in the catch block of the main program. If the age is 18 or older, the program prints a message indicating eligibility to vote. The use of the throws keyword ensures that the CustomError exception is properly handled, while providing clear feedback on age validation.
3. The result of the Java program demonstrating multiple catch statements is that it handles various exceptions that may arise from a single try block. For each input processed, the program attempts to parse it as an integer and perform a division operation. If the input causes a NumberFormatException, ArithmeticException, NullPointerException, or any other unexpected exception, the corresponding catch block handles it by printing a specific error message. This ensures that different types of errors are managed appropriately, providing clear feedback for each type of exception encountered.
4. The result of the Java program demonstrating exception handling with the throw keyword and finally block involves explicitly throwing an exception when a certain condition is met and ensuring that a finally block always executes. In the program, an exception (e.g., IllegalArgumentException) is thrown if a specific condition (e.g., invalid input) occurs. Regardless of whether an exception is thrown or not, the finally block runs to perform cleanup or final actions, such as closing resources or printing a message. This approach guarantees that essential code is executed even if an error occurs.

Exercise 8

Hariesh R - 23110344

Aim:

1. The program allocates employees to different party halls based on their ID and age, following COVID protocols. User-defined exceptions prevent employees from entering incorrect halls, and the program also calculates the average age of employees in each hall while handling potential exceptions.
2. The program calculates a user's age from their date of birth and checks if they are eligible to vote. If the age is less than 18, a user-defined exception is thrown stating the person cannot vote; otherwise, it confirms eligibility.

Algorithm:

1. Q1)
 - i) Initialize variables:
 - (a) Create an array halls to store up to 100 Hall objects and a halls_index to track entries.
 - ii) Input employee details:
 - (a) Prompt for the number of employees and gather each employee's ID and age.
 - iii) Assign employees to halls:
 - (a) Try to assign each employee to Hall1, Hall2, or Hall3 based on their ID and age.
 - (b) If valid, store the hall object in halls; otherwise, catch and print the CustomError exception.
 - iv) Compute and print average age:
 - (a) After each employee, call CalculateAverageAge(halls, false) to print the average age for the last assigned hall.
 - v) Final report:
 - (a) After all employees are processed, call CalculateAverageAge(halls, true) to print the average age for all halls.
 - vi) Close input:
 - (a) Close the Scanner to end user input.

2. Q2)

- i) Input current date and date of birth:
 - (a) Prompt the user to enter the current date and their date of birth in DD-MM-YYYY format.
- ii) Calculate age:
 - (a) Split the input strings to extract day, month, and year for both the current date and the date of birth.
 - (b) Calculate the age by subtracting birth year from the current year.
 - (c) Adjust the age if the birth month and day are later than the current month and day.
- iii) Check voting eligibility:
 - (a) If the age is 18 or older, print "Eligible for voting."
 - (b) Otherwise, throw a `InvalidAgeForVoting` exception with the message "Not eligible for voting."
- iv) Handle exceptions:
 - (a) Catch and display the `InvalidAgeForVoting` exception if the user is underage.
- v) End the program:
 - (a) Close the Scanner after the process is complete.

Source Code:

1) Q1:

```
package Exercise8.Q1;

public interface Hall {

    int getAge();
    int getEmployeeID();

    void setAge(int age);
    void setEmployeeID(int employeeID);
}

package Exercise8.Q1;

public class Hall1 implements Hall {

    private int employeeID;
    private int age;

    Hall1(int employeeID, int age) throws CustomError {

        if (employeeID % 2 == 0 && age < 30) {
```

```

        this.employeeID = employeeID;
        this.age = age;

        System.out.println("Welcome to the Party -> Hall 1");
    }

    else throw new CustomError("You are not allowed in hall 1");

}

@Override
public int getAge() {
    return age;
}

@Override
public int getEmployeeID() {
    return employeeID;
}

@Override
public void setAge(int age) {
    this.age = age;
}

@Override
public void setEmployeeID(int employeeID) {
    this.employeeID = employeeID;
}
}

package Exercise8.Q1;

public class Hall2 implements Hall {

    private int employeeID;
    private int age;

    Hall2(int employeeID, int age) throws CustomError {

        if (employeeID % 2 == 1 && age > 30) {

            this.employeeID = employeeID;
            this.age = age;

```

```

        System.out.println("Welcome to the Party -> Hall 2");
    }

    else throw new CustomError("You are not allowed in hall 2");

}

@Override
public int getAge() {
    return age;
}

@Override
public int getEmployeeID() {
    return employeeID;
}

@Override
public void setAge(int age) {
    this.age = age;
}

@Override
public void setEmployeeID(int employeeID) {
    this.employeeID = employeeID;
}
}

package Exercise8.Q1;

public class Hall3 implements Hall{

    int employeeID;
    int age;

    Hall3(int employeeID, int age) throws CustomError {

        if((employeeID % 2 == 0 && age < 30) || (employeeID % 2 == 1 && age > 30))
            System.out.println("You are not allowed in hall 3");

        else {

            this.employeeID = employeeID;
            this.age = age;

            System.out.println("Welcome to the Party -> Hall 3");

```

```

    }
}

@Override
public int getAge() {
    return age;
}

@Override
public int getEmployeeID() {
    return employeeID;
}

@Override
public void setAge(int age) {
    this.age = age;
}

@Override
public void setEmployeeID(int employeeID) {
    this.employeeID = employeeID;
}
}

package Exercise8.Q1;

public class CustomError extends Exception {

    CustomError(String message) {
        super(message);
    }
}

package Exercise8.Q1;

public class CalculateAverage {

    public static void CalculateAverageAge(Hall[] halls, boolean all) {

        double[] average = {0, 0, 0};
        int[] count = {0, 0, 0};
        int[] sum = {0, 0, 0};

        Hall lastHall = null;

        for (Hall hall : halls) {

```

```

    if(hall != null) {

        if (hall instanceof Hall1){

            sum[0] += hall.getAge();
            count[0]++;
        }

        if (hall instanceof Hall2){

            sum[1] += hall.getAge();
            count[1]++;
        }

        if (hall instanceof Hall3){

            sum[2] += hall.getAge();
            count[2]++;
        }

        lastHall = hall;
    }
}

for (int i = 0; i < 3; i++) {

    if (count[i] != 0) average[i] = (double) sum[i] / count[i];
}

if (all){

    for (int i = 0; i < 3; i++) System.out.println("Average of Hall" + (i+1) + ": " +
average[i]);
}

else{

    if (lastHall != null) {

        if (lastHall instanceof Hall1) System.out.println("Average of Hall1: " +
average[0]);
        else if (lastHall instanceof Hall2) System.out.println("Average of Hall2: " +
average[1]);
        else System.out.println("Average of Hall3: " + average[2]);
    }
}

```

```

        else System.out.println("Unable to find any Hall");
    }
}
}

```

```
package Exercise8.Q1;
```

```
import java.util.Scanner;
```

```
public class Q1 {
```

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

```
        Scanner scanner = new Scanner(System.in);
```

```
        Hall[] halls = new Hall[100];
```

```
        int halls_index = 0;
```

```
        int noOfEmployees;
```

```
        int employeeID;
```

```
        int age;
```

```
        System.out.print("Enter no. of employees: ");
```

```
        noOfEmployees = scanner.nextInt();
```

```
        scanner.nextLine();
```

```
        for (int i = 0; i < noOfEmployees; i++) {
```

```
            System.out.print("Enter employee ID: ");
```

```
            employeeID = scanner.nextInt();
```

```
            scanner.nextLine();
```

```
            System.out.print("Enter age: ");
```

```
            age = scanner.nextInt();
```

```
            scanner.nextLine();
```

```
            Hall hall1 = null;
```

```
            Hall hall2 = null;
```

```
            Hall hall3 = null;
```

```
            try {
```

```
                hall1 = new Hall1(employeeID, age);
```

```
            } catch (CustomError e){
```

```

        System.out.println(e.getMessage());
    }

    try {

        hall2 = new Hall2(employeeID, age);

    } catch (CustomError e){
        System.out.println(e.getMessage());
    }

    try {

        hall3 = new Hall3(employeeID, age);
    } catch (CustomError e){
        System.out.println(e.getMessage());
    }

    if (hall1 != null) halls[halls_index++] = hall1;
    else if (hall2 != null) halls[halls_index++] = hall2;
    else if (hall3 != null) halls[halls_index++] = hall3;

    try {

        CalculateAverage.CalculateAverageAge(halls, false);
    } catch (Exception e){
        System.out.println(e.getMessage());
    }
}

System.out.println("\nFinal Report: ");
CalculateAverage.CalculateAverageAge(halls, true);
scanner.close();
}
}

```

2) Q2:

```

package Exercise8.Q2;

public class InvalidAgeForVoting extends Exception{

    InvalidAgeForVoting(String msg) {
        super(msg);
    }
}

```

```

    }
}

package Exercise8.Q2;

public class Voting {

    public static void validAgeForVoting(String currentData, String dateOfBirth)
throws InvalidAgeForVoting{

        String[] dobParts = dateOfBirth.split("-");
        String[] dateParts = currentData.split("-");

        int birthDay = Integer.parseInt(dobParts[0]);
        int birthMonth = Integer.parseInt(dobParts[1]);
        int birthYear = Integer.parseInt(dobParts[2]);

        int currentDay = Integer.parseInt(dateParts[0]);
        int currentMonth = Integer.parseInt(dateParts[1]);
        int currentYear = Integer.parseInt(dateParts[2]);

        int age = currentYear - birthYear;

        if (birthMonth > currentMonth || (birthMonth == currentMonth && birthDay >
currentDay)) age--;

        if (age >= 18) System.out.println("Eligible for voting");
        else throw new InvalidAgeForVoting("Not eligible for voting");
    }
}

```

```

package Exercise8.Q2;

import java.util.Scanner;

public class Q2 {

    public static void main(String[] args) throws InvalidAgeForVoting {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the date in the format DD-MM-YYYY: ");

        System.out.print("Enter current date: ");
        String curDate = scanner.nextLine();
    }
}

```



```
System.out.print("Enter date of birth: ");
String birthDate = scanner.nextLine();

try {

    Voting.validAgeForVoting(curDate, birthDate);
} catch (InvalidAgeForVoting e) {

    System.out.println(e.getMessage());
}

scanner.close();
}
}
```

Output:

1) Q1:

```
Enter no. of employees: 4
Enter employee ID: 12
Enter age: 12
Welcome to the Party -> Hall 1
You are not allowed in hall 2
You are not allowed in hall 3
Average of Hall1: 12.0
Enter employee ID: 35
Enter age: 45
You are not allowed in hall 1
Welcome to the Party -> Hall 2
You are not allowed in hall 3
Average of Hall2: 45.0
```

```
Enter employee ID: 12
Enter age: 30
You are not allowed in hall 1
You are not allowed in hall 2
Welcome to the Party -> Hall 3
Average of Hall3: 30.0
Enter employee ID: 14
Enter age: 14
Welcome to the Party -> Hall 1
You are not allowed in hall 2
You are not allowed in hall 3
Average of Hall1: 13.0

Final Report:
Average of Hall1: 13.0
Average of Hall2: 45.0
Average of Hall3: 30.0

Process finished with exit code 0
```

2) Q2

```
Enter the date in the format DD-MM-YYYY:
Enter current date: 09-09-2024
Enter date of birth: 28-06-2006
Eligible for voting

Process finished with exit code 0
|
Enter the date in the format DD-MM-YYYY:
Enter current date: 09-09-2024
Enter date of birth: 28-06-2008
Not eligible for voting

Process finished with exit code 0
|
```

Result:

1. The program assigns employees to different halls based on their ID and age, following COVID protocols. User-defined exceptions prevent entry into incorrect halls, and after each assignment, the average age for each hall is calculated and printed, with proper exception handling.
2. The program calculates a user's age from their date of birth and checks if they are eligible to vote. If the age is less than 18, a user-defined exception is thrown indicating they cannot vote; otherwise, it confirms they are eligible.

Exercise 9

hariesh - 23110344

Aim:

1. The Python script validates user credentials from credentials.txt. If valid, it writes command-line argument content to a specified file.
2. The script reads a file with 10 students and their marks, randomly assigned. It creates two files: best_performers for students with marks above 90, and low_performers for those with marks below 40. Exception handling ensures robust file operations.

Algorithm:

1. Q1:

Initialize: Set up the file (validation.txt) and input scanner.

Check Args: Ensure at least three arguments are provided (username, password, and content).

Read Args: Extract username, password, and content from arguments.

Validate: Check credentials against validation.txt. Print "Access Denied" if not found.

Write to File: If credentials are valid, write the content to secret.txt.

Handle exceptions as needed.

Close Resources: Close the scanner and file writers.

2. Q2:

Generate Marks: Create marks.txt with 10 student names and random marks (1-100).

Write Marks: Save the names and marks to marks.txt.

Classify Performances:

Open marks.txt and read the data.

Create best_performance.txt for students with marks > 90.

Create low_performance.txt for students with marks < 40.

Write Results: Write names to the appropriate files based on their marks.

Handle Exceptions: Manage file I/O errors and print appropriate messages.

Source Code:

1. Q1:

```
package Exercise9.Q1;

import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.FileWriter;

public class main_program {
    public static void main(String[] args) {
        File validationFile = new File("validation.txt");
        Scanner scanner = new Scanner(System.in);
        String userName, password, data, userNameCurrent, passwordCurrent;
        boolean found = false;

        String content = "";

        if (args.length < 3){

            System.out.println("Insufficient number of args");
            scanner.close();
            return;
        }

        userName = args[0];
        password = args[1];
```

```

for (int i = 2; i < args.length; i++) content += args[i] + " ";

try {
    Scanner readValidationFile = new Scanner(validationFile);

    while (readValidationFile.hasNextLine()){
        data = readValidationFile.nextLine();

        userNameCurrent = data.split(" ")[0];
        passwordCurrent = data.split(" ")[1];
        if (userNameCurrent.equals(userName) &&
            passwordCurrent.equals(password)){
            System.out.println("Found: " + userNameCurrent + passwordCurrent);
            found = true;
            break;
        }
    }

    readValidationFile.close();
} catch (FileNotFoundException e){
    System.out.println("Error: " + e.getMessage());
}

if(!found) System.out.println("Access Denied");

if(found){

    try {

        File secretFile = new File("secret.txt");
        FileWriter secretFileWriter = new FileWriter("secret.txt");

        secretFile.createNewFile();
        secretFileWriter.write(content);
        System.out.println("Written Successfully");
    }
}

```

```
secretFileWriter.close();

} catch (IOException e){

System.out.println("Error: " + e.getMessage());
}
}

scanner.close();
}
}
```

2. Q2:

```
package Exercise9.Q2;

import java.util.Random;
import java.util.Scanner;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.FileNotFoundException;

public class main_program {

    public static void main(String[] args) {

        Random random = new Random();
        String[] names = {"Liam", "Olivia", "Noah", "Emma", "Oliver", "Charlotte",
            "James", "Amelia", "Elijah", "Sophia"};
        String data, studentName;
        int marks;
        try {

            File marksFile = new File("marks.txt");
```



```

FileWriter marksFileWriter = new FileWriter(marksFile);

marksFile.createNewFile();

for (String name : names) {
    marks = random.nextInt(100) + 1;
    data = name + " " + marks + "\n";

    marksFileWriter.write(data);
}

marksFileWriter.close();

} catch (IOException e){

    System.out.println("Error: " + e.getMessage());
}

try {

    File marksFile = new File("marks.txt");
    Scanner readMarks = new Scanner(marksFile);

    File bestPerformance = new File("best performace.txt");
    bestPerformance.createNewFile();
    FileWriter bestPerformanceWriter = new FileWriter(bestPerformance);

    File lowPerformance = new File("low performace.txt");
    lowPerformance.createNewFile();
    FileWriter lowPerformanceWriter = new FileWriter(lowPerformance);

    while (readMarks.hasNextLine()){

        data = readMarks.nextLine();
        marks = Integer.parseInt(data.split(" ")[1]);
        studentName = data.split(" ")[0];
    }
}

```

```
if (marks > 90) bestPerformanceWriter.write(studentName + "\n");
if (marks < 40) lowPerformanceWriter.write(studentName + "\n");
}

readMarks.close();
bestPerformanceWriter.close();
lowPerformanceWriter.close();

} catch (FileNotFoundException e){


System.out.println("Error: " + e.getMessage());
} catch (IOException e){


System.out.println("Error: " + e.getMessage());
}
}
}
```

Output:


1. Q1:


```
snucse@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzzesh/Exercise9/Q1$ javac main_program.java
snucse@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzzesh/Exercise9/Q1$ java main_program.java hariesh 123 hello this is a sample text
Found: hariesh123
Written Successfully
snucse@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzzesh/Exercise9/Q1$
```

 validation.txt ✕

Exercise9 > Q1 >  validation.txt

```
1 hariesh 123
2 arun 321
3 userhello 345
4 userhi 987
```

 secret.txt ✕

Exercise9 > Q1 >  secret.txt

```
1 hello this is a sample text
```

2. Q2:

```
● snucse@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzzesh/Exercise9/Q2$ javac main_program.java
● snucse@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzzesh/Exercise9/Q2$ java main_program.java
○ snucse@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/harizzzesh/Exercise9/Q2$
```

Exercise9 > Q2 > marks.txt

```
1  Liam 12
2  Olivia 78
3  Noah 92
4  Emma 97
5  Oliver 27
6  Charlotte 87
7  James 57
8  Amelia 7
9  Elijah 18
10 Sophia 98
11
```

Exercise9 > Q2 > best performace.txt

```
1  Noah
2  Emma
3  Sophia
4
```

Exercise9 > Q2 > low performace.txt

```
1  Liam
2  Oliver
3  Amelia
4  Elijah
5
```

Result:

1. The program validates a username and password against credentials in a file. If the validation succeeds, it writes command-line arguments to a new file.
2. The program creates a file `marks.txt` with 10 students and random marks between 1 and 100. It then generates two additional files: `best_performers`, listing students with marks above 90, and `low_performers`, listing students with marks below 40. It handles file reading and writing errors with appropriate messages.

q1)

Aim: Write a java program using built-in exception to check if the file is found at a particular location

Algorithm:

- **Input:** Prompt the user to enter a file path.
- **Create File Object:** Instantiate a `File` object using the provided file path.
- **Check Existence:** Use the `exists()` method of the `File` object to check if the file exists.
- **Output Result:** Print the appropriate message based on the file's existence.
- **Handle Exceptions:** If a `SecurityException` occurs, inform the user about access denial.
- **Close Scanner:** Finally, close the scanner resource.

Code:

```
import java.util.*;
public class Filecheck
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the path of the file:");
        String filepath = scan.nextLine();
        File file = new File(filepath);

        try
        {
            if(file.exists())
            {
                System.out.println("File is found at folder "+file.getAbsolutePath());
            }
            else
            {
                System.out.println("File is not found at folder "+file.getAbsolutePath());
            }
        }
        catch(SecurityException e)
        {
            System.out.println("Security error occurred "+e.getLocalizedMessage());
        }
        finally
        {
            scan.close();
        }
    }
}
```

}

OUTPUT:

```
ai_ds-b2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$ javac Filecheck.java
ai_ds-b2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$ java Filecheck
Enter the path of the file:
/home/ai_ds-b2
File is found at folder /home/ai_ds-b2
ai_ds-b2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$
```

q2)

Aim: To Write a java program to get the last modification date and time of a file

Algorithm:

- Prompt the user to enter a file path.
- Create a File object using the provided file path.
- Check if the file exists.
- If it exists, get the last modified time in milliseconds and convert it to a Date object.
- Format the date to a readable string and display it.
- If the file does not exist, inform the user that the file was not found.

CODE:

```
import java.util.*;
import java.io.*;
import java.text.SimpleDateFormat;
public class dateset
{
    public static void main(String[] args) {
        try
        {
            Scanner scan = new Scanner(System.in);
            System.out.println("Enter the file's path :");
            String filepath = scan.nextLine();
```

```

File file = new File(filepath);
if(file.exists())
{
    long lastsaved = file.lastModified();
    Date date = new Date(lastsaved);

    SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    String Dateformat = format.format(date);

    System.out.println("The file is last modified at "+Dateformat);
}
else
{
    System.out.println("The given file doesn't exist");
}

}
catch(SecurityException e)
{
    System.out.println("Security exception occurred:"+ e.getLocalizedMessage());
}
}
}

```

OUTPUT:

```

ai_ds-b2@s nucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$ javac dateset.java
ai_ds-b2@s nucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$ java dateset
Enter the file's path :
/home/ai_ds-b2
The file is last modified at 2024-10-04 12:39:11
ai_ds-b2@s nucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$ 

```

q3)

Aim: To Write a java program to rename an existing file.

Algorithm:

- Prompt the user to enter the file's path and read it.
- Create a `File` object for the specified path.
- Prompt the user to enter the new file name and read it.
- Create a new `File` object using the original file's parent directory and the new name.

- Check if the original file exists; if it does, attempt to rename it.
- Print a success message if renamed, or an error message if renaming fails or if the file was not found.
-

CODE:

```
import java.util.*;
import java.io.*;

public class filerename
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in) ;
        System.out.println("Enter the file's path :");
        String filepath = scan.nextLine();
        File file = new File(filepath);
        System.out.println("Enter the file's new name :");
        String newfilename = scan.nextLine();
        File newnamedfile= new File(file.getParent(),newfilename);

        if(file.exists())
        {
            if(file.renameTo(newnamedfile))
            {
                System.out.println("File Renamed...");
            }
            else{
                System.out.println("error occured while renaming the file");
            }
        }

        else {
            System.out.println("File not found at: " + file.getAbsolutePath());
        }

        scan.close();
    }
}
```

OUTPUT:

```
ai_ds-b2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$ javac filerename.java
ai_ds-b2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$ java filerename
Enter the file's path :
/home/ai_ds-b2/oopslab10
Enter the file's new name :
rename.txt
File Renamed...
ai_ds-b2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$
```


q4)

Aim: To Write a java program to create directory or folder in particular drive

Algorithm:

- Prompt the user to enter the directory path where the new folder should be created.
- Create a File object using the provided directory path.
- Attempt to create the directory using the mkdir() method.
- If the directory is created successfully, print a success message with the directory's absolute path.
- If directory creation fails, print an error message indicating the failure reason.
- Close the scanner to release resources.

CODE:

```
import java.util.*;
import java.io.*;

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

        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the directory path");
        String dir = scan.nextLine();
        File directory = new File(dir);

        if(directory.mkdir())
        {
            System.out.println("Directory created...");
        }
        else
        {
            System.out.println("Directory cannot be created");
        }

        scan.close();

    }
}
```

OUTPUT:

```

File renamed...
ai_ds-b2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$ javac directory.java
ai_ds-b2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$ java directory
Enter the directory path
/home/ai_ds-b2/testfolder
Directory created...
ai_ds-b2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$ javac directory.java
ai_ds-b2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$ java directory
Enter the directory path
/home/ai_ds-b2/testfolder
Directory cannot be created
ai_ds-b2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$ 

```

q5)

Aim: Write a java program to check whether a file can be read or not

Algorithm:

Prompt the user to enter the file path to check its readability.

- Create a `File` object using the provided file path.
- Check if the file exists.
- If the file exists, use the `canRead()` method to determine if it can be read.
- Print a message indicating whether the file can be read or not.
- If the file does not exist, inform the user that the file was not found.

CODE:

```

import java.util.*;
import java.io.*;

public class fileread
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("enter the file path: ");
        String path = scan.nextLine();
        File file = new File(path);
        if(file.exists())
        {
            if(file.canRead())
            {
                System.out.println("File is read succesfully!!!");
            }
            else
            {
                System.out.println("File cannot be read...");
            }
        }
    }
}

```

```
    else
    {
        System.out.println("File not existing at "+ file.getAbsolutePath());
    }
    scan.close();
}
}
```

OUTPUT:

```
ai_ds-b2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$ javac fileread.java
ai_ds-b2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$ java fileread
enter the file path:
/home/ai_ds-b2/filetest.txt
File is read succesfully!!!
ai_ds-b2@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/oopslab10$
```

Exercise 11

Hariesh R - 23110344

Aim:

1. The aim of this program is to create a multithreaded application by subclassing the Thread class. Two thread objects are initialized and started, which execute concurrently. Each thread displays the message "Today is hot, humid, and sunny."
2. The program creates two threads using the Runnable interface. One thread prints "Hi" and the other prints "AI," each with a time interval of 300 milliseconds. These threads run concurrently, continuously printing their respective messages with the specified delay.
3. This Java program simulates a bank account where multiple threads concurrently perform deposits and withdrawals. Inter-thread communication methods like wait() and notify() are used to synchronize the operations, ensuring that withdrawals only occur when there is enough balance, and deposits notify waiting threads when funds become available.
4. This Java program defines a generic method that takes a list of numbers and calculates the sum of all the even and odd numbers separately. The method iterates through the list, checks whether each number is even or odd, and then returns the sums of both types of numbers.

Algorithm:

1. Q1)
 1. Define the Thread Class:
 2. Create a class myThread extending Thread.
 3. Override the run() method to print:
 4. "Today is hot"
 5. "Today is humid"
 6. "Today is sunny"
 7. Create the Main Class:

8. Define the Main class with the main method.
9. Instantiate and Start Threads:
 10. Create two myThread instances (thread1 and thread2).
 11. Call start() on both threads to begin execution.
 12. Wait for Threads to Complete:
 13. Use a try-catch block to call join() on both threads to ensure they finish before proceeding.
 14. Print any exception message if an error occurs.

2. Q2)

1. Define the Runnable Class:
 2. Create a class myThread that implements the Runnable interface.
 3. Declare a private final String message to hold the message.
 4. Define a constructor to initialize the message.
 5. Override the run() method to continuously print the message in a loop.
 6. Add Sleep and Interrupt Handling:
 7. Inside the loop, use Thread.sleep(300) to pause for 300 milliseconds after printing the message.
 8. Catch InterruptedException, interrupt the current thread, and break the loop if the thread is interrupted.
 9. Create the Main Class:
 10. Define the Main class with the main method.
 11. Instantiate and Start Threads:
 12. Create two Thread objects (thread1 and thread2), passing myThread instances with "HI" and "AI" as messages.
 13. Call start() on both threads to begin execution.
 14. Wait for 30 Seconds:
 15. Use a try-catch block to call Thread.sleep(30000) in the main thread to allow the child threads to run for 30 seconds.
 16. Interrupt Threads:
 17. After 30 seconds, call interrupt() on both threads to stop their execution.

3. Q3)

1. Initialize Components:
2. Create instances for Bank, RandomGenerator, List<Thread>, and Scanner.
3. Get User Input:
4. Prompt the user for the initial balance, number of transactions, and waiting time (in seconds).
5. Create Bank Instance:
6. Initialize a Bank object with the user-defined initial balance and wait time.
7. Generate Transactions:
8. Loop for the specified number of transactions:
9. Use RandomGenerator to determine the transaction type (deposit or withdraw) and amount.
10. Print the transaction details.
11. Call createAndStartThread() to create and start a new thread for each transaction.
12. Wait for Threads to Complete:
13. After starting all threads, loop through the threads list and call join() on each to wait for their completion.
14. Check for Failed Transactions:
15. After all threads finish, check if there are any failed transactions in the bank account.
16. Print "No Failed Transactions" if the list is empty.
17. Otherwise, print the list of failed transactions.
18. Display Final Balance:
19. Print the final account balance after all transactions are processed.
20. Define Thread Classes:
21. DepositTask: Inherits from Thread, calls the deposit() method on the Bank object and simulates a delay.
22. WithdrawTask: Inherits from Thread, calls the withdraw() method on the Bank object, handles exceptions, and simulates a delay.
23. Define Bank Class:
24. Implement methods for depositing, withdrawing, checking balance, and handling failed transactions using synchronized blocks to ensure thread safety.
25. Define RandomGenerator Class:

26. Create methods for generating random transaction types and amounts.

27. Define WithdrawalTimeoutException Class:

28. Create a custom exception class for handling withdrawal timeouts.

4. Q4)

1. Initialize Main Class:

2. Define the Main class with the main method.

3. Create a List:

4. Initialize an ArrayList<Integer> and populate it with integers from 0 to 20.

5. Call Generic Method:

6. Invoke myClass.myMethod() and pass the list of numbers as an argument.

7. Define the Generic Method:

8. Create a static method myMethod in myClass that accepts a List<T> where T extends Number.

9. Initialize Sum Variables:

10. Declare and initialize oddSum and evenSum to 0.

11. Iterate Through the List:

12. Loop through each number in the list:

13. Check if the number is even or odd using the modulus operator (%).

14. Add the number to evenSum if it's even, or to oddSum if it's odd.

15. Print the Results:

16. After processing all numbers, print the sums of odd and even numbers.

Source Code:

1. Q1)

```
package Exercise11.Q1;
```

```
class myThread extends Thread {
```

```

@Override
public void run (){

    System.out.println("Today is hot");
    System.out.println("Today is humid");
    System.out.println("Today is sunny");
}
}

public class Main {

    public static void main(String[] args) {

        myThread thread1 = new myThread();
        myThread thread2 = new myThread();

        thread1.start();
        thread2.start();

        // Wait for the threads to complete
        try {

            thread1.join();
            thread2.join();
        } catch (Exception exception){

            System.out.println(exception.getMessage());
        }
    }
}

```

2. Q2)

```

package Exercise11.Q2;

class myThread implements Runnable {

    private final String message;

    public myThread(String message){

        this.message = message;
    }
}

```



```

@Override
public void run(){

    while (true){

        System.out.println(message);

        try {

            Thread.sleep(300);
        } catch (InterruptedException exception){

            Thread.currentThread().interrupt();
            break;
        }
    }
}

public class Main {

    public static void main(String[] args) {

        Thread thread1 = new Thread(new myThread("HI"));
        Thread thread2 = new Thread(new myThread("AI"));

        thread1.start();
        thread2.start();

        //    Wait Main Thread For 3 seconds
        try {

            Thread.sleep(3000);
        } catch (InterruptedException exception){

            System.out.println(exception.getMessage());
        }

        //    Kill the threads
        thread1.interrupt();
        thread2.interrupt();
    }
}

```

```
}
```

3. Q3)

```
package Exercise11.Q3;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Bank account;
        RandomGenerator rand;
        List<Thread> threads;
        Scanner scanner;

        String transactionType;
        double amount;
        int transactionCount, initialBalance;
        long waitTime;

        {
            scanner = new Scanner(System.in);
            rand = new RandomGenerator();
            threads = new ArrayList<>();
        }

        System.out.print("Enter Initial Balance: ");
        initialBalance = scanner.nextInt();

        System.out.print("Enter No Of Transactions: ");
        transactionCount = scanner.nextInt();

        System.out.print("Enter Waiting Time In Seconds: ");
        waitTime = scanner.nextLong() * 1000;

        {
            account = new Bank(initialBalance, waitTime);
```

```

    }

    for (int i = 1; i < transactionCount + 1; i++){

        transactionType = rand.deposit_withdraw();
        amount = rand.getAmount();

        System.out.println("Starting Transaction " + i + " " + transactionType + " " +
            "Amount: " + amount);

        createAndStartThread(account, amount, transactionType + " id:" + i, threads,
            transactionType);
    }

    // Wait for all the threads to complete
    for (Thread thread : threads){

        try {

            thread.join();
        } catch (InterruptedException e) {
            System.out.println(e.getMessage());
            Thread.currentThread().interrupt();
        }
    }

    if (account.getFailedTransactions().isEmpty()) {

        System.out.println("No Failed Transactions.");
    }
    else {

        System.out.println("Failed Transactions:");
        account.getFailedTransactions().forEach(Main::log);
    }

    System.out.println("\nFinal Account Balance: " + account.getAmount());
}

private static void createAndStartThread (Bank bank, double amount, String name,
    List<Thread> threads, String type){

    Thread transactionThread;

```

```

        if (type.equals("deposit")){
            transactionThread = new DepositTask(bank, amount, name);

            // If current balance is low, prioritize deposits to avoid failed withdrawals
            if (bank.getAmount() < 1000)
                transactionThread.setPriority(Thread.MAX_PRIORITY);

            else
                transactionThread.setPriority(Thread.NORM_PRIORITY);
        }

        else {

            transactionThread = new WithdrawTask(bank, amount, name);

            // If balance is sufficient, prioritize withdrawals to avoid long wait times
            if (bank.getAmount() >= amount)
                transactionThread.setPriority(Thread.MAX_PRIORITY);

            else
                transactionThread.setPriority(Thread.NORM_PRIORITY);
        }

        transactionThread.start();
        threads.add(transactionThread);
    }

    private static void log (String message){

        System.out.println("[ " + java.time.LocalDateTime.now() + " ] " + message);
    }
}

class Bank {

    private double amount;
    private final long waitTime;
    private final List<String> failedTransactions;

    {
        failedTransactions = new ArrayList<>();
    }
}

```

```

public Bank(){

    this.amount = 0.0;
    waitTime = 5000;
}

public Bank(double amount, long waitTime){

    this.amount = amount;
    this.waitTime = waitTime;
}

public Bank(int amount, long waitTime){

    this((double) amount, waitTime);
}

public synchronized void deposit (double amount){

    this.amount += amount;

    System.out.println("Deposited: " + amount);
    this.printBalance();

    notifyAll();
}

public synchronized void withdraw (double amount) throws InterruptedException,
WithdrawalTimeoutException{

    long startTime = System.currentTimeMillis();

    while (amount > this.amount){

        long elapsedTime = System.currentTimeMillis() - startTime;
        long remainingTime = waitTime - elapsedTime;

        if (remainingTime <= 0){

            String errorMessage = "Timeout waiting for deposits for withdrawal of
amount: " + amount;

```

```

        failedTransactions.add(Thread.currentThread().getName());

        throw new WithdrawalTimeoutException(errorMessage);
    }

    System.out.println("Insufficient Funds! Waiting For Deposits...");
    wait(remainingTime);
}

this.amount -= amount;
System.out.println("Withdrawn: " + amount);
this.printBalance();
}

public synchronized double getAmount(){

    return amount;
}

private void printBalance(){

    System.out.println("Balance: " + Math.round(this.amount * 100.0) / 100.0);
}

public List<String> getFailedTransactions(){
    return failedTransactions;
}
}

class DepositTask extends Thread {

    private final Bank account;
    private final double amount;

    public DepositTask(Bank account, double amount, String name){

        super(name);
        this.account = account;
        this.amount = amount;
    }

    @Override
    public void run (){

```

```

        account.deposit(amount);

        try {

            // Depositing Delay
            Thread.sleep(1000);

        } catch (InterruptedException exception){

            Thread.currentThread().interrupt();
        }
    }
}

```

```

class WithdrawTask extends Thread {

    private final Bank account;
    private final double amount;

    public WithdrawTask(Bank account, double amount, String name){

        super(name);
        this.account = account;
        this.amount = amount;
    }

    @Override
    public void run (){

        try {

            account.withdraw(amount);

            // Withdrawing Delay
            Thread.sleep(2000);

        } catch (InterruptedException | WithdrawalTimeoutException exception){

            System.out.println("Withdrawal failed: " + exception.getMessage());
            Thread.currentThread().interrupt();
        }
    }
}

```

```
    }  
}
```

```
class RandomGenerator {
```

```
    private int min_amount;  
    private int max_amount;  
    private final Random random;
```

```
    {  
        min_amount = 200;  
        max_amount = 500;  
        random = new Random();  
    }
```

```
    public RandomGenerator(){  
  
    }
```

```
    public RandomGenerator(int min_amount, int max_amount){  
        this.min_amount = min_amount;  
        this.max_amount = max_amount;  
    }
```

```
    public String deposit_withdraw(){  
  
        return (random.nextBoolean()) ? "deposit" : "withdraw";  
    }
```

```
    public double getAmount(){  
  
        double amount = min_amount + random.nextDouble() * (max_amount -  
min_amount);  
  
        return Math.round(amount * 100.0) / 100.0;  
    }
```

```
    public int getMin_amount() {  
        return min_amount;  
    }
```

```
    public void setMin_amount(int min_amount) {  
        this.min_amount = min_amount;
```



```

    }

    public int getMax_amount() {
        return max_amount;
    }

    public void setMax_amount(int max_amount) {
        this.max_amount = max_amount;
    }
}

class WithdrawalTimeoutException extends Exception {
    public WithdrawalTimeoutException(String message) {
        super(message);
    }
}

```

4. Q4)

```

package Exercise11.Q4;

import java.util.ArrayList;
import java.util.List;

public class Main {

    public static void main(String[] args) {

        ArrayList<Integer> list = new ArrayList<>();

        for (int i = 0; i < 21; i++) list.add(i);

        myClass.myMethod(list);
    }
}

class myClass {

    public static <T extends Number> void myMethod(List<T> numbers) {

        int oddSum, evenSum;
    }
}

```

```
{
    oddSum = 0;
    evenSum = 0;
}

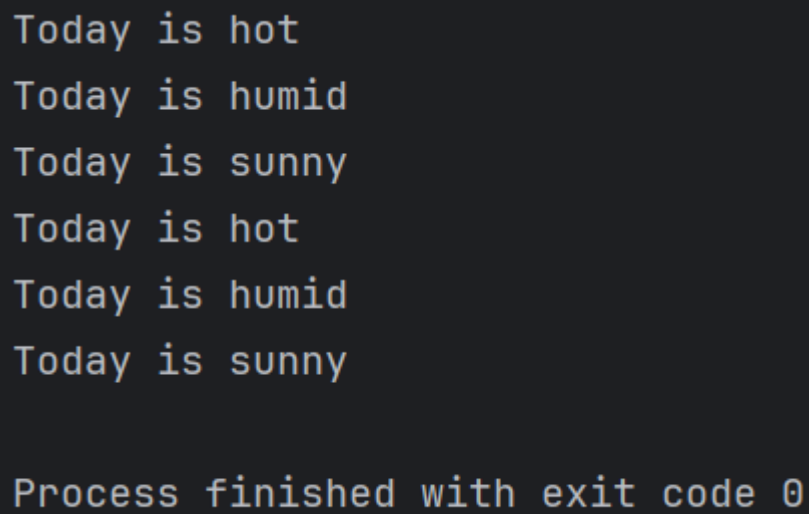
for (T number: numbers){

    if ((int) number % 2 == 0) evenSum += (int) number;
    else oddSum += (int) number;
}

System.out.println("Odd Number Sum: " + oddSum);
System.out.println("Even Number Sum: " + evenSum);
}
}
```

Output:

1.

A terminal window with a dark background and light-colored text. It displays six lines of weather-related strings: "Today is hot", "Today is humid", "Today is sunny", "Today is hot", "Today is humid", and "Today is sunny". At the bottom, it shows "Process finished with exit code 0".

```
Today is hot
Today is humid
Today is sunny
Today is hot
Today is humid
Today is sunny

Process finished with exit code 0
```

2.

```
HI  
AI  
AI  
HI  
AI  
HI  
HI  
AI  
HI  
AI  
AI  
HI
```

```
Process finished with exit code 0
```

3.

```
Enter Initial Balance: 1000
Enter No Of Transactions: 10
Enter Waiting Time In Seconds: 3
Starting Transaction 1 withdraw Amount: 493.4
Starting Transaction 2 withdraw Amount: 326.7
Starting Transaction 3 deposit Amount: 273.31
Starting Transaction 4 deposit Amount: 497.53
Starting Transaction 5 deposit Amount: 447.56
Starting Transaction 6 withdraw Amount: 240.81
Starting Transaction 7 deposit Amount: 309.66
```

```
Withdrawn: 326.7
Balance: 673.3
Deposited: 447.56
Balance: 1120.86
Withdrawn: 240.81
Balance: 880.05
Deposited: 497.53
Balance: 1377.58
Deposited: 273.31
Balance: 1650.89
Withdrawn: 493.4
Balance: 1157.49
```

```
Starting Transaction 8 withdraw Amount: 361.0
Starting Transaction 9 withdraw Amount: 474.77
Starting Transaction 10 deposit Amount: 203.45
Deposited: 309.66
Balance: 1467.15
Withdrawn: 361.0
Balance: 1106.15
Deposited: 203.45
Balance: 1309.6
Withdrawn: 474.77
Balance: 834.83
No Failed Transactions.
```

```
Final Account Balance: 834.8299999999999
Process finished with exit code 0
```

4.

```
Odd Number Sum: 100
Even Number Sum: 110
Process finished with exit code 0
```

Result:

1. The result of the multithreaded Java program will display the messages "Today is hot," "Today is humid," and "Today is sunny" from two concurrently executing threads. The output may vary in order due to thread scheduling, but it will consist

of a total of six printed statements—three from each thread—resulting in the appearance of these phrases twice, potentially interleaved in different runs.

2. The result of the Java program that creates two threads using the Runnable interface will display the messages "Hi" and "AI" alternately, with each thread printing its message every 300 milliseconds. The output will consist of these two phrases appearing in quick succession, potentially interleaved, depending on the thread scheduling. The program will continue printing "Hi" and "AI" until the main thread completes or the program is terminated.
3. The result of the Java program that simulates a bank account with concurrent deposits and withdrawals will display the results of various transactions processed by multiple threads. It will handle deposits and withdrawals while ensuring thread safety through inter-thread communication methods. The output will include messages indicating successful deposits or withdrawals, along with any failed transactions due to insufficient funds. Additionally, the program will print the final account balance after all transactions are completed, and if there are any failed transactions, those will also be displayed. The exact output may vary based on the timing of thread execution.
4. The result of the Java program that defines a generic method for calculating the sums of even and odd numbers from a list will display the total sums for each category. After processing the provided list of integers, the output will include two lines: one showing the sum of odd numbers and the other showing the sum of even numbers. For example, if the input list contains integers from 0 to 20, the output will specify the calculated sums, with the exact values depending on the numbers in the list.

LAB12

19-10-2024

Q1)

Aim:

Write a program that will display check boxes and option buttons they are numbered from 1 to 3. Use a textbox to display the number those corresponding boxes or button checked.

Algorithm:

Initialize the Frame and Components:

- Create a JFrame and set up the layout to FlowLayout.
- Initialize three checkboxes (CheckBox 1, CheckBox 2, CheckBox 3).
- Initialize three radio buttons (Option 1, Option 2, Option 3).
- Group the radio buttons so only one can be selected at a time.
- Initialize a non-editable text field to display the selected options.

Add Components to Frame:

- Add the checkboxes, radio buttons, and the text field to the JFrame.

ActionListener Setup:

- Create an ActionListener that reacts whenever a checkbox or radio button is selected or deselected.
- Inside the ActionListener, call the updateSelectedItems() method to update the text field.

UpdateSelectedItems Method:

- Create a method updateSelectedItems() to:
 - Clear the text field.
 - Check which checkboxes and radio buttons are selected.

- Append the corresponding numbers (1, 2, 3) for the selected checkboxes and radio buttons to a string.
- Update the text field to display the selected numbers.

Display the Frame:

- Set the frame visibility to true to display the GUI.

Main Method:

- Use `SwingUtilities.invokeLater()` to start the GUI in a thread-safe manner and instantiate the Q1 class.

Code:

```
/*
```

Write a program that will display check boxes and option buttons they are numbered from 1 to 3. Use a textbox to display the number those corresponding boxes or button checked.

```
*/
```

```
package LAB12;
```

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
public class Q1 {
```

```
    private JFrame frame;
```

```
    private JCheckBox checkBox1, checkBox2, checkBox3;
```

```
    private JRadioButton radioButton1, radioButton2, radioButton3;
```

```
    private JTextField textField;
```



```
private ButtonGroup radioGroup;
```

```
public Q1() {
```

```
    frame = new JFrame("Check Boxes and Option Buttons");
```

```
    frame.setLayout(new FlowLayout());
```

```
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
    frame.setSize(300, 200);
```

```
    checkBox1 = new JCheckBox("CheckBox 1");
```

```
    checkBox2 = new JCheckBox("CheckBox 2");
```

```
    checkBox3 = new JCheckBox("CheckBox 3");
```

```
    radioButton1 = new JRadioButton("Option 1");
```

```
    radioButton2 = new JRadioButton("Option 2");
```

```
    radioButton3 = new JRadioButton("Option 3");
```

```
    radioGroup = new ButtonGroup();
```

```
    radioGroup.add(radioButton1);
```

```
    radioGroup.add(radioButton2);
```

```
    radioGroup.add(radioButton3);
```

```
    textField = new JTextField(15);
```

```
    textField.setEditable(false);
```

```
    frame.add(checkBox1);
```

```
    frame.add(checkBox2);
```

```
frame.add(checkBox3);  
frame.add(radioButton1);  
frame.add(radioButton2);  
frame.add(radioButton3);  
frame.add(textField);
```

```
ActionListener updateTextField = new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        updateSelectedItems();  
    }  
};
```

```
checkBox1.addActionListener(updateTextField);  
checkBox2.addActionListener(updateTextField);  
checkBox3.addActionListener(updateTextField);  
radioButton1.addActionListener(updateTextField);  
radioButton2.addActionListener(updateTextField);  
radioButton3.addActionListener(updateTextField);
```

```
frame.setVisible(true);  
}
```

```
private void updateSelectedItems() {  
    StringBuilder selectedItems = new StringBuilder();
```

```

        if (checkBox1.isSelected()) {
            selectedItems.append("1 ");
        }
        if (checkBox2.isSelected()) {
            selectedItems.append("2 ");
        }
        if (checkBox3.isSelected()) {
            selectedItems.append("3 ");
        }

        if (radioButton1.isSelected()) {
            selectedItems.append("1 ");
        }
        if (radioButton2.isSelected()) {
            selectedItems.append("2 ");
        }
        if (radioButton3.isSelected()) {
            selectedItems.append("3 ");
        }

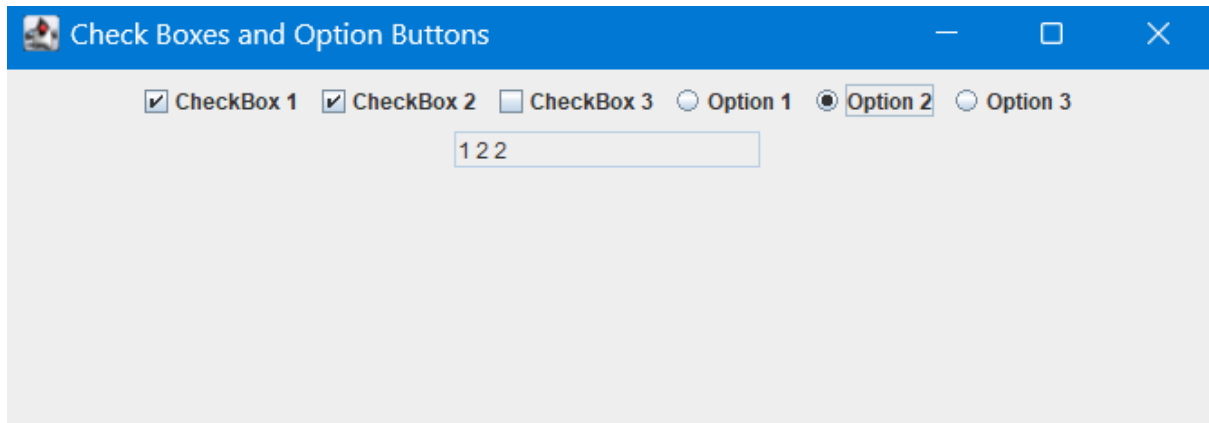
        textField.setText(selectedItems.toString());
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new Q1());
    }

```

```
}
```

Output:



Q2)

Aim:

Write a program to create a Applet life cycle.

Algorithm:

Applet Initialization (init()):

- When the applet is loaded for the first time, the init() method is called.
- Print "Applet is initialized" to the console.

Applet Start (start()):

- After initialization, or when the applet comes into view, the start() method is called.
- Print "Applet is started" to the console.

Applet Stop (stop()):

- When the applet goes out of view or is no longer active, the stop() method is called.
- Print "Applet is stopped" to the console.

Applet Destruction (destroy()):

- Just before the applet is unloaded or destroyed, the destroy() method is called.
- Print "Applet is destroyed" to the console.

Painting the Applet (paint()):

- The paint(Graphics g) method is responsible for rendering any graphical output.
- Draw the string "Welcome to Applet Life Cycle Demo" at coordinates (20, 50) in the applet window.

Code:

Q2COMMENTS.java

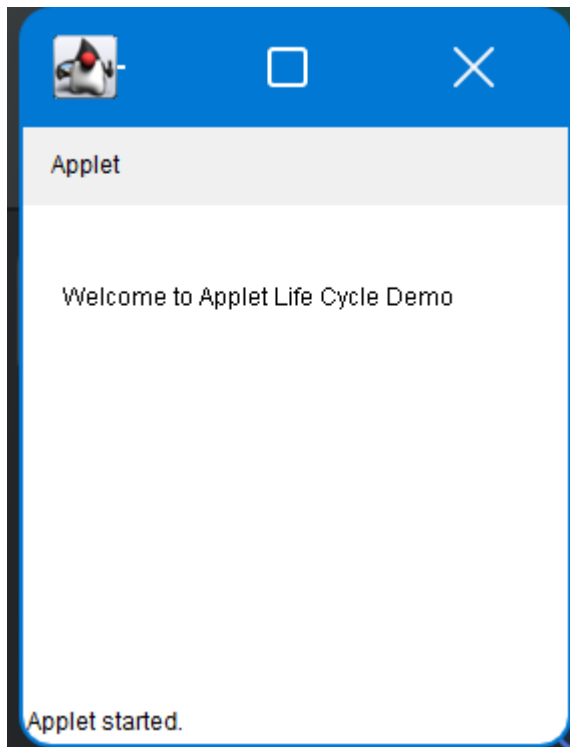
```
/*  
Write a program to create a Applet life cycle.  
*/  
package LAB12;  
  
import java.applet.Applet;  
import java.awt.Graphics;  
  
public class Q2COMMENTS extends Applet {  
    public void init() {  
        System.out.println("Applet is initialized.");  
    }  
  
    public void start() {  
        System.out.println("Applet is started.");  
    }  
  
    public void stop() {  
        System.out.println("Applet is stopped.");  
    }  
}
```

```
public void destroy() {  
    System.out.println("Applet is destroyed.");  
}  
  
public void paint(Graphics g) {  
    g.drawString("Welcome to Applet Life Cycle Demo", 20, 50);  
}  
}
```

applet.html

```
<html>  
  
<body>  
  
    <applet code="LAB12.Q2COMMENTS" width="300" height="200">  
  
    </applet>  
  
</body>  
  
</html>
```

Output:



```
Applet is initialized.  
Applet is started.  
Applet is stopped.  
Applet is destroyed.
```