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:

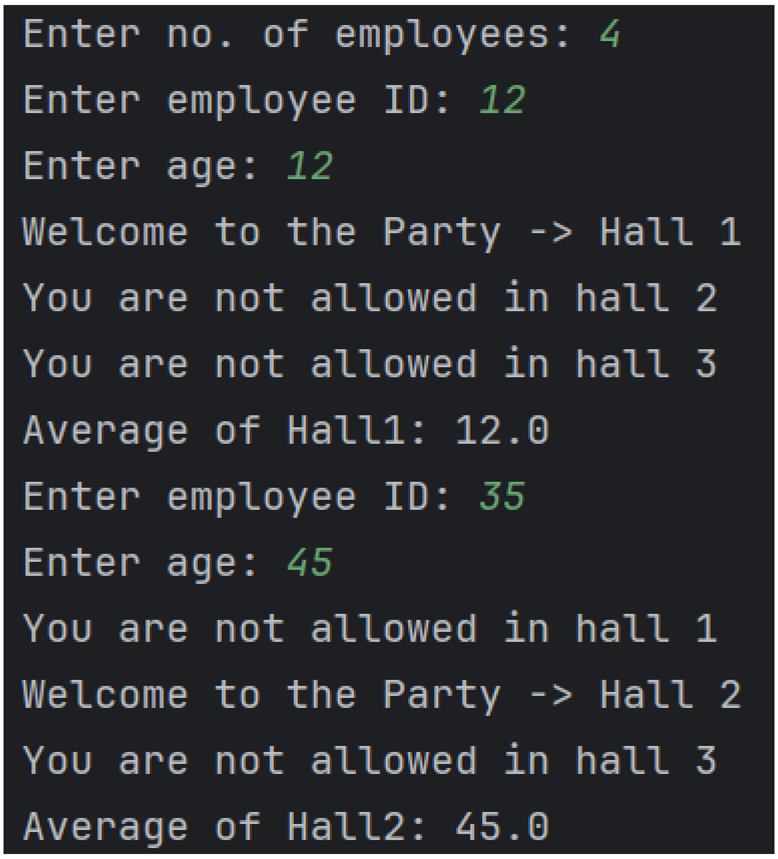
```
package Exercise8.Q1;

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:

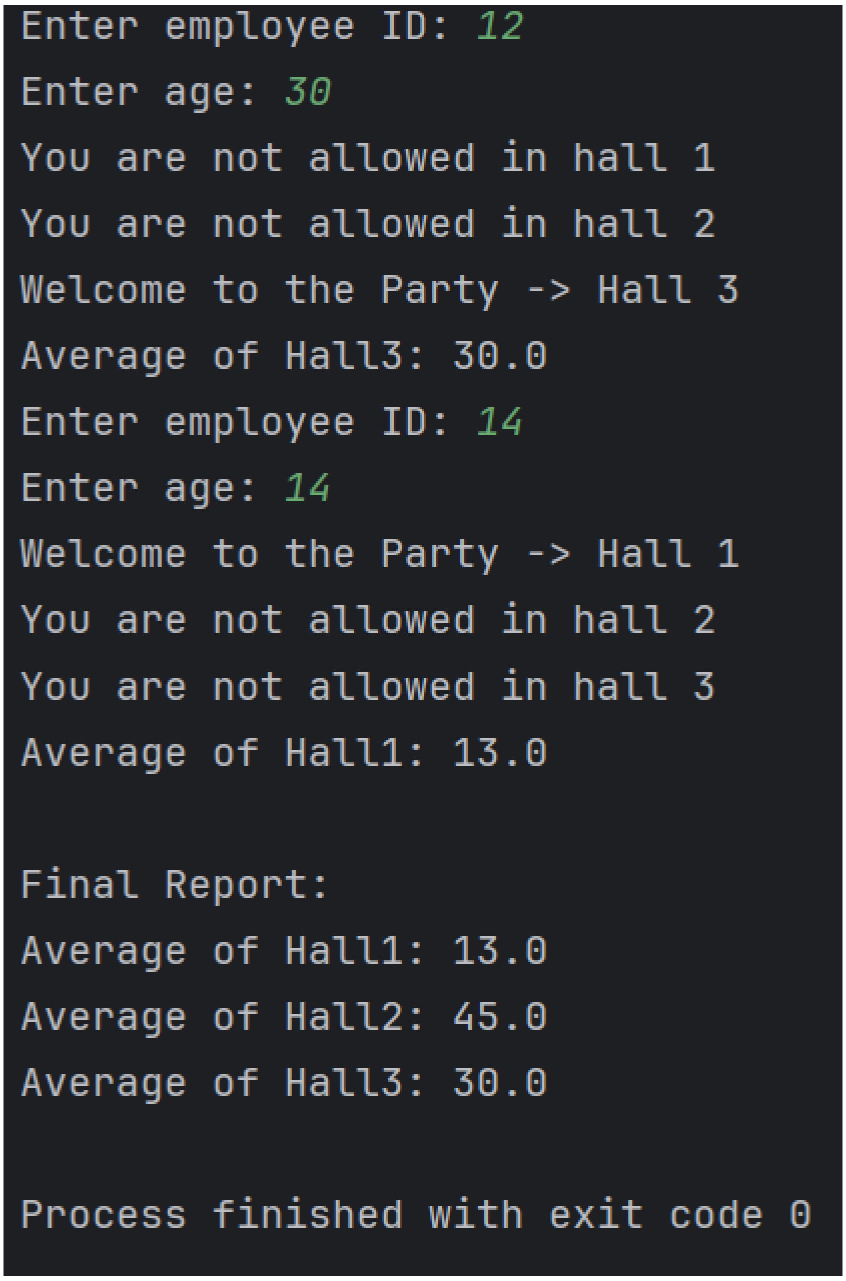
```
package Exercise8.Q2;

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:

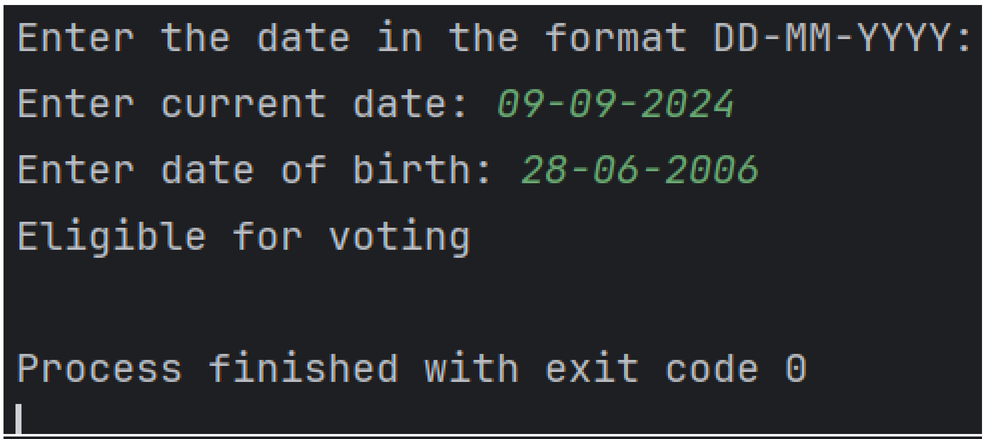


```
Enter no. of employees: 4
Enter employee ID: 12
Enter age: 12
Welcome to the Party -> Hall 1
You are not allowed in hall 2
You are not allowed in hall 3
Average of Hall1: 12.0
Enter employee ID: 35
Enter age: 45
You are not allowed in hall 1
Welcome to the Party -> Hall 2
You are not allowed in hall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

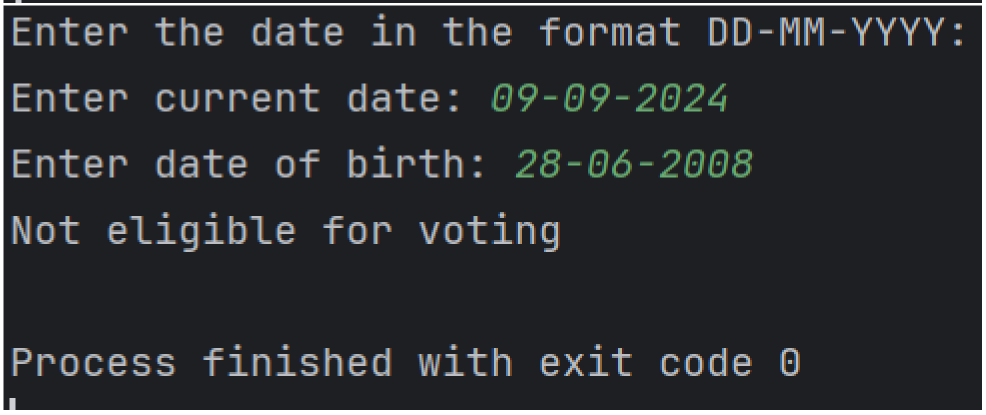


```
Enter employee ID: 12
Enter age: 30
You are not allowed in hall 1
You are not allowed in hall 2
Welcome to the Party -> Hall 3
Average of Hall3: 30.0
Enter employee ID: 14
Enter age: 14
Welcome to the Party -> Hall 1
You are not allowed in hall 2
You are not allowed in hall 3
Average of Hall1: 13.0

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.