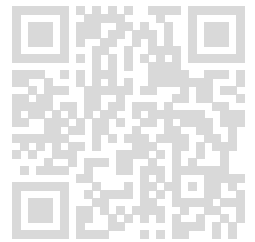


Codekata Report:



Name: Shaurya Singh

Email: singhshaurya851@gmail.com

Specialization: School of Computing Science & Engineering

Completion Year: 2027

Section: Section-52

1. Complex Variable Initialization and Arithmetic Operations

Problem Statement Write a program that takes four integers as input, performs a series of arithmetic operations using these integers, and outputs the results in a specific format.

Description You need to declare four integer variables, perform the following operations, and print the results:

Add the first and the second integer, then multiply the result by the third integer.

Subtract the fourth integer from the second integer, then divide the result by the first integer.

Multiply the first integer by the fourth integer, then add the third integer to the result.

Add all four integers together and divide by two.

Input Format Four integers, each on a new line.

Output Format Four lines of output, each showing the result of the respective operations in the following format: Result of operation 1: <result> Result of operation 2: <result> Result of operation 3: <result> Result of operation 4: <result>

Sample Input: 3421

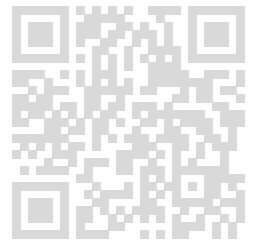
Sample output: Result of operation 1: 14 Result of operation 2: 1 Result of operation 3: 5 Result of operation 4: 5

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA



Source Code:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt();
        int b = sc.nextInt();
        int c = sc.nextInt();
        int d = sc.nextInt();

        //..... YOUR CODE STARTS HERE .....
        int result1 = (a + b) * c;
        int result2 = (b - d) / a;
        int result3 = (a * d) + c;
        int result4 = (a + b + c + d) / 2;
        System.out.println("Result of operation 1: "+result1);
        System.out.println("Result of operation 2: "+result2);
        System.out.println("Result of operation 3: "+result3);
        System.out.println("Result of operation 4: "+result4);
        //..... YOUR CODE ENDS HERE .....
    }
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Result of operation 1: 14
Result of operation 2: 1
Result of operation 3: 5
Result of operation 4: 5

Compilation Status: Passed

Execution Time:

0.107s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Result of operation 1: 150

Result of operation 2: 1

Result of operation 3: 25

Result of operation 4: 18

Compilation Status: Passed

Execution Time:

0.121s

2. OOPs Features & Interface

Problem Statement Implement an interface Employee with methods calculateSalary and getDetails. Create two classes FullTimeEmployee and PartTimeEmployee that implement this interface. Each class should have its own way of calculating salary.

Description Define an interface Employee with methods double calculateSalary() and String getDetails(). Create a class FullTimeEmployee with properties name (String), monthlySalary (double) and implement the interface methods. Create a class PartTimeEmployee with properties name (String), hourlyRate (double), and hoursWorked (int) and implement the interface methods. Ensure each class has a constructor to initialize its properties.

Input Format String representing the employee type (FullTimeEmployee or PartTimeEmployee). If FullTimeEmployee, a string for name and a double for monthlySalary. If PartTimeEmployee, a string for name, a double for hourlyRate, and an integer for hoursWorked. **Output Format** String representing the employee details. Double representing the calculated salary.

Sample Input 1: FullTimeEmployeeJohn3000.0 **Sample Output 1:** Name: John, Salary: 3000.0

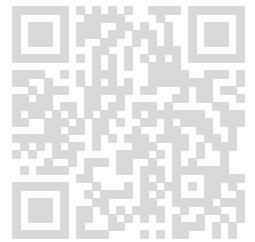
Sample Input 2: PartTimeEmployeeJane15.0120 **Sample Output 2:** Name: Jane, Salary: 1800.0

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA



Source Code:

```
import java.util.Scanner;

interface Employee {
    double calculateSalary();
    String getDetails();
}

class FullTimeEmployee implements Employee {
    //..... YOUR CODE STARTS HERE .....
    private String name;
    private double monthlySalary;

    public FullTimeEmployee(String name, double monthlySalary) {
        this.name = name;
        this.monthlySalary = monthlySalary;
    }

    @Override
    public double calculateSalary()
    {
        return monthlySalary;
    }

    @Override
    public String getDetails() {
        return "Name: " + name + ", Salary: " + calculateSalary();
    }
    //..... YOUR CODE ENDS HERE .....
}

class PartTimeEmployee implements Employee {
    //..... YOUR CODE STARTS HERE .....
    private String name;
    private double hourlyRate;
    private int hoursWorked;

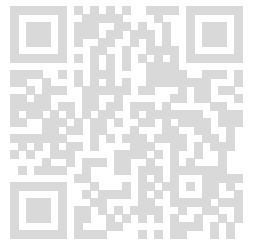
    public PartTimeEmployee(String name, double hourlyRate, int hoursWorked) {

        this.name = name;
        this.hourlyRate = hourlyRate;
        this.hoursWorked = hoursWorked;
    }
    @Override

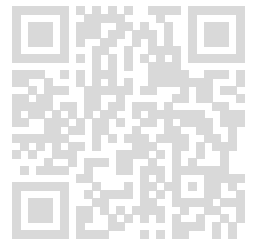
    public double calculateSalary()

    {
        return hourlyRate * hoursWorked;
    }

    @Override
```



```
public String getDetails()
{
return "Name: " + name + ", Salary: " + calculateSalary();
}
//..... YOUR CODE ENDS HERE .....
}
```



```
public class Main {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
String employeeType = scanner.next();
Employee employee;

if (employeeType.equals("FullTimeEmployee")) {
String name = scanner.next();
double monthlySalary = scanner.nextDouble();
employee = new FullTimeEmployee(name, monthlySalary);
} else if (employeeType.equals("PartTimeEmployee")) {
String name = scanner.next();
double hourlyRate = scanner.nextDouble();
int hoursWorked = scanner.nextInt();
employee = new PartTimeEmployee(name, hourlyRate, hoursWorked);
} else {
System.out.println("Invalid employee type");
return;
}

System.out.println(employee.getDetails());
}
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Name: John, Salary: 3000.0

Compilation Status: Passed

Execution Time:

0.137s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

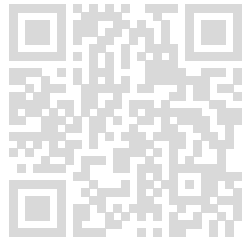
Output:

Name: Jane, Salary: 1800.0

Compilation Status: Passed

Execution Time:

0.135s



3. Bitwise Operations with Floating Point Variables

Problem Statement Write a program that takes two floating-point numbers as input, performs bitwise operations by converting them to integers, and outputs the results in a specific format.

Description You need to declare two floating-point variables, convert them to integers, perform the following bitwise operations, and print the results:

Perform bitwise AND on the two integers.

Perform bitwise OR on the two integers.

Perform bitwise XOR on the two integers.

Perform bitwise NOT on the first integer and AND it with the second integer.

Input Format Two floating-point numbers, each on a new line.

Output Format Four lines of output, each showing the result of the respective operations in the following format: Bitwise AND result: <result> Bitwise OR result: <result> Bitwise XOR result: <result> Bitwise NOT and AND result: <result>

Sample Input: 5.53.3

Sample Output: Bitwise AND result: 1 Bitwise OR result: 7 Bitwise XOR result: 6 Bitwise NOT and AND result: 2

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        float f1 = sc.nextFloat();
        float f2 = sc.nextFloat();

        int n1 = (int) f1;
        int n2 = (int) f2;

        //..... YOUR CODE STARTS HERE .....
        int andResult = n1 & n2;
        int orResult = n1 | n2;
        int xorResult = n1 ^ n2;
        int notAndResult = (~n1)& n2;

        System.out.println("Bitwise AND result: "+ andResult);
        System.out.println("Bitwise OR result: "+ orResult);
        System.out.println("Bitwise XOR result: "+ xorResult);
        System.out.println("Bitwise NOT and AND result: " + notAndResult);
        //..... YOUR CODE ENDS HERE .....
    }
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

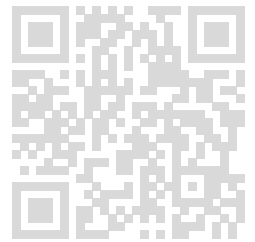
Bitwise AND result: 1
Bitwise OR result: 7
Bitwise XOR result: 6
Bitwise NOT and AND result: 2

Compilation Status: Passed

Execution Time:

0.12s

TestCase2:



Shaurya Singh (singhshaurya851@gmail.com)

Input:

< hidden >

Expected Output:

< hidden >

Output:

Bitwise AND result: 8

Bitwise OR result: 10

Bitwise XOR result: 2

Bitwise NOT and AND result: 0

Compilation Status: Passed

Execution Time:

0.116s

4. Variable Transformation Challenge

Problem Statement: In a parallel universe, the concept of variable transformation is quite different. You need to write a Java program that performs a series of operations on variables based on specific conditions. Given three integers a, b, and c, your task is to transform these variables according to the following rules and output the final values.

Description:

If a is even, add b to a.

If b is odd, multiply c by 2.

If c is a multiple of 3, add a to c.

If the sum of a, b, and c is greater than 100, subtract 100 from each of a, b, and c.

Your program should then print the transformed values of a, b, and c in the format "a: [value], b: [value], c: [value]".

Input Format: Three integers a, b, and c are given as input from the user, each on a new line.

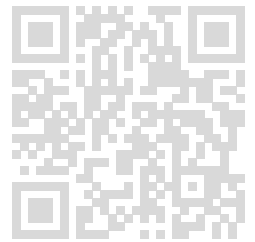
Output Format: Print the transformed values of a, b, and c in the specified format.

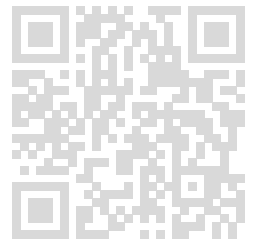
Sample input: 10 15 9

Sample output: a: 25, b: 15, c: 43

Completion Status: Completed

Concepts Included:





Language Used: JAVA

Source Code:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input
        int a = scanner.nextInt();
        int b = scanner.nextInt();
        int c = scanner.nextInt();
        // Transformations
        //..... YOUR CODE STARTS HERE .....

        if (a % 2 == 0) {
            a += b;
        }

        if (b % 2 != 0) {
            c *= 2;
        }

        if (c % 3 == 0) {
            c += a;
        }

        if(a + b + c > 100) {
            a -= 100;
            b -= 100;
            c -= 100;
        }
        //..... YOUR CODE ENDS HERE .....
        // Output
        System.out.println("a: " + a + ", b: " + b + ", c: " + c);
    }
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

a: 25, b: 15, c: 43

Compilation Status: Passed

Execution Time:

0.113s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

a: 0, b: -50, c: -50

Compilation Status: Passed

Execution Time:

0.113s

5. Odd-Even Sum Checker

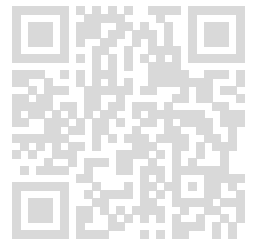
Problem Statement: Write a program that reads a list of integers from the user and checks if the sum of odd numbers is greater than the sum of even numbers in the list. If the sum of odd numbers is greater, print "Odd Sum Greater". If the sum of even numbers is greater, print "Even Sum Greater". If both sum's are equal, print "Equal". Print "Equal" if both sum's are equal.

Description: The program should continuously read integers until the user inputs a specific stop value. You need to handle edge cases where there are no odd or even numbers.

Input Format: The first line contains an integer n (number of elements). The second line contains n integers separated by spaces. The input stops when the user enters -1. **Output Format:** Print "Odd Sum Greater" if the sum of odd numbers is greater. Print "Even Sum Greater" if the sum of even numbers is greater. Print "Equal" if both sum's are equal.

Sample input: 4 2 4 6 8 -1 **Sample Output:** Even Sum Greater

Completion Status: Completed



Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        //..... YOUR CODE STARTS HERE .....
        int n = scanner.nextInt();
        int[] numbers = new int[n];

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

            numbers[i] = scanner.nextInt();
        }

        int oddSum = 0;
        int evenSum = 0;

        for (int number : numbers) {

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

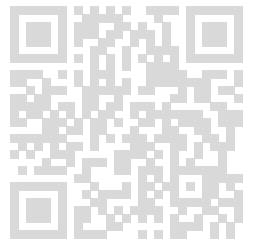
        if(oddSum > evenSum) {
            System.out.println("Odd Sum Greater");
        }

        else if (evenSum > oddSum) {
            System.out.println("Even Sum Greater");
        }

        else

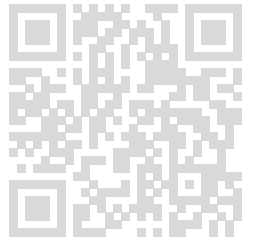
        {

            System.out.println("Equal");
        }
        //..... YOUR CODE ENDS HERE .....
    }
}
```



Shaurya Singh (singhshaurya851@gmail.com)

}



Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Even Sum Greater

Compilation Status: Passed

Execution Time:

0.09s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Odd Sum Greater

Compilation Status: Passed

Execution Time:

0.091s

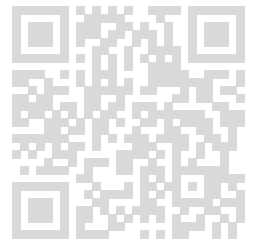
6. Month Days Finder

Problem Statement: Write a program that takes a month (as a number) and a year as input and prints the number of days in that month, considering leap years.

Description: The program should read the month and year continuously until the user inputs a stop value. You need to handle leap years and invalid inputs.

Input Format: The first line contains an integer n (number of entries). The next n lines contain two integers each: the month and the year. The input stops when the user enters -1 -1. **Output Format:** Print the number of days in the given month for each input.

Sample Input:2 2020 -1 -1Sample Output:29



Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        //..... YOUR CODE STARTS HERE .....

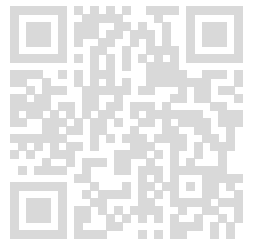
        while (true) {
            int month = scanner.nextInt();
            int year = scanner.nextInt();

            if (month == -1 && year == -1) {
                break;
            }

            if (month < 1 || month > 12) {
                System.out.println("Invalid month");
                continue;
            }

            int daysInMonth;
            switch (month) {
                case 1: case 3: case 5: case 7: case 8: case 10: case 12:
                    daysInMonth = 31;
                    break;
                case 4: case 6: case 9: case 11:
                    daysInMonth = 30;
                    break;
                case 2:
                    if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {
                        daysInMonth = 29;
                    } else {
                        daysInMonth = 28;
                    }
                    break;
                default:
                    daysInMonth = 0;
            }
        }
    }
}
```

```
System.out.println(daysInMonth);
}  
  
//..... YOUR CODE ENDS HERE .....  
  
scanner.close();  
}  
}
```



Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

29

Compilation Status: Passed

Execution Time:

0.089s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

28

Compilation Status: Passed

Execution Time:

0.092s

7. Grade Classification

Problem Statement: Write a program that classifies grades based on the score input

by the user. The program should classify the score into "Excellent", "Good", "Average", "Pass", or "Fail" using if-else statements.

Description: The program should read the scores continuously until the user inputs a stop value. The classification is based on the following criteria:

90-100: Excellent 75-89: Good 50-74: Average 35-49: Pass 0-34: Fail

Input Format: The first line contains an integer n (number of scores). The second line contains n integers separated by spaces. The input stops when the user enters -1. Output Format: For each score, print its classification.

Sample Input: 5 95 85 65 45 30 -1 Sample Output: Excellent Good Average Pass Fail

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

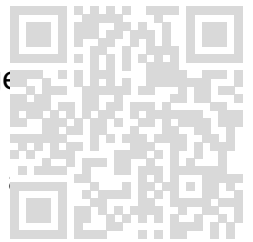
Language Used: JAVA

Source Code:

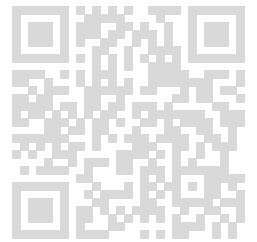
```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        //..... YOUR CODE STARTS HERE .....
        //System.out.println("Enter the Number of scores:");
        int n = scanner.nextInt();
        // System.out.println("Enter Scores:");
        for(int i = 0; i < n ; i++)
        {
            int score = scanner.nextInt();

            if (score == -1)
            {
                break;
            }
            if (score >= 90 && score <= 100) {
                System.out.println("Excellent");
            } else if (score >= 75 && score <= 89) {
                System.out.println("Good");
            } else if (score >= 50 && score <= 74) {
                System.out.println("Average");
            } else if (score >= 35 && score <= 49) {
                System.out.println("Pass");
            } else if (score >= 0 && score <= 34) {
                System.out.println("Fail");
            } else {
```



```
System.out.println("Invalid score");
}
}
//..... YOUR CODE ENDS HERE .....
scanner.close();
}
}
```



Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Odd Sum Greater

Compilation Status: Failed

Execution Time:

0.087s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Even Sum Greater

Compilation Status: Failed

Execution Time:

0.088s

8. The Voting System

Problem Statement: You are designing a voting system where voters can cast their votes for various candidates. Each voter can vote only once, and the system needs to

validate each vote and ensure the following rules are adhered to:

A vote must be cast for a valid candidate number.

The total number of votes should not exceed the number of registered voters.

A vote for a candidate should only be allowed if the candidate's ID is within the valid range.

Description: You need to implement a Java program that:

Takes the number of registered voters as input.

Takes the number of candidates as input.

Takes each vote (candidate ID) as input.

Outputs whether each vote is valid or invalid based on the rules above.

Input Format: First line: Integer N (Number of registered voters) Second line: Integer C (Number of candidates) Third line: Integer V (Number of votes) Next V lines: Each line contains an integer ID (candidate ID) Output Format: For each vote, print "VALID" if the vote is valid, otherwise print "INVALID".

Sample Input: 100541632 Sample Output: VALIDINVALIDVALIDVALID

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.Scanner;

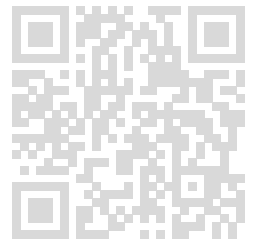
public class Main {
    public static void main(String[] args) {
        //..... YOUR CODE STARTS HERE .....
        Scanner scanner = new Scanner(System.in);

        int numberOfVoters = scanner.nextInt();

        int numberOfCandidates = scanner.nextInt();

        int numberOfVotes = scanner.nextInt();

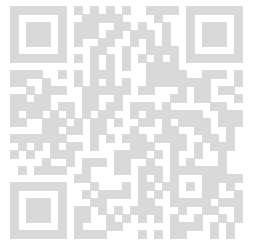
        for(int i = 0; i < numberOfVotes; i++) {
            int candidateID = scanner.nextInt();
```



```
if (candidateID >= 1 && candidateID <= numberOfCandidates)
{
    System.out.println("VALID");

}
else{
    System.out.println("INVALID");

}
}
scanner.close();//scaaner closed
//..... YOUR CODE ENDS HERE .....
}
```



Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

VALID
INVALID
VALID
VALID

Compilation Status: Passed

Execution Time:

0.089s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

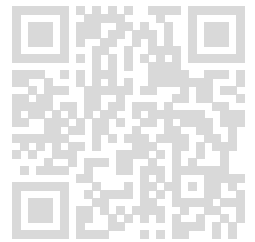
VALID
VALID
VALID
INVALID

Shaurya Singh (singhshaurya851@gmail.com)

Compilation Status: Passed

Execution Time:

0.092s



9. Nested Loop Patterns

Problem Statement: Write a program that generates a pattern based on user input using nested loops. The pattern to be generated is a pyramid where each level contains a specific number of symbols. Each level's pattern should follow a unique rule:

The number of symbols on each level is equal to the level number.

The symbol for each level is based on the level number where odd levels use * and even levels use #.

Description: You need to implement a Java program that:

Takes an integer

N (number of levels) as input.

Generates and prints the pyramid pattern according to the rules described.

Input Format: Single line: Integer N (Number of levels) **Output Format:** Pyramid pattern with N levels, where odd levels use * and even levels use #.

Sample Input: 3 **Sample Output:** *

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        //..... YOUR CODE STARTS HERE .....
        Scanner scanner = new Scanner(System.in);
```

```
int levels = scanner.nextInt();

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

char symbol = (i % 2 == 0) ? '#':'*';

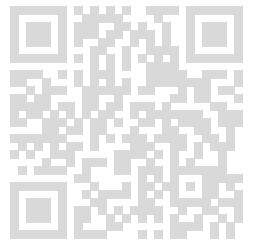
for(int j = 0; j < i; j++) {

System.out.print(symbol);
}

System.out.println();

scanner.close();

//..... YOUR CODE ENDS HERE .....
}
}
```



Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
*
##
***
```

Compilation Status: Passed

Execution Time:

0.09s

TestCase2:

Input:

< hidden >

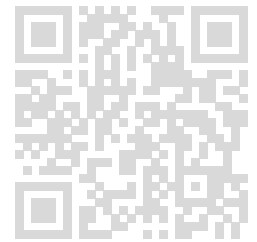
Expected Output:

< hidden >

Output:

*

####



Compilation Status: Passed

Execution Time:

0.087s

10. Classes and Objects

Problem Statement Create a class Car that represents a car with a unique twist. The Car class should include properties such as make, model, and year. Additionally, it should have a method getCarAge that calculates the car's age based on the current year input by the user. However, you need to ensure that the car's year of manufacture is validated to be not in the future.

Description Define a class Car with private properties: make (String), model (String), and year (int). Include a constructor that initializes these properties. Create a method getCarAge that calculates the age of the car based on the current year provided by the user. Ensure the year property is validated to be less than or equal to the current year.

Input Format String representing the make of the car. String representing the model of the car. Integer representing the year of the car. Integer representing the current year. **Output Format** Integer representing the age of the car. A message indicating if the year is invalid.

Sample Input 1: Toyota Corolla 2015 2024 Sample Output 1: 9

Sample Input 2: Tesla Model S 2025 2024 Sample Output 2: Invalid year

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String make = scanner.next();
        String model = scanner.next();
        int year = scanner.nextInt();
```

```
int currentYear = scanner.nextInt();

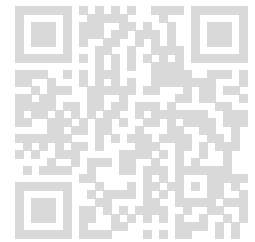
Car car = new Car(make, model, year);
int age = car.getCarAge(currentYear);

if (age != -1) {
    System.out.println(age);
}
}
}

class Car {
//..... YOUR CODE STARTS HERE .....
private String make;
private String model;
private int year;

public Car (String make,String model,int year){
    this.make = make;
    this.model = model;
    this.year = year;
}

public int getCarAge(int currentYear) {
    if(year > currentYear) {
        return -1;
    } else {
        return currentYear - year;
    }
}
//..... YOUR CODE ENDS HERE .....
}
```



Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

9

Compilation Status: Passed

Execution Time:

0.101s

Shaurya Singh (singhshaurya851@gmail.com)

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

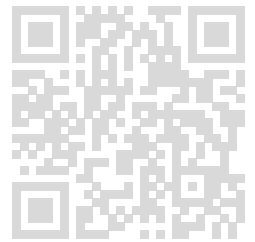
Output:

6

Compilation Status: Passed

Execution Time:

0.096s



11. OOPs Features & Interface

Problem Statement:Create an interface Shape with methods area and perimeter. Implement this interface in two classes: Rectangle and Circle. Each class should have appropriate constructors and methods to calculate the area and perimeter.

Description:Define an interface Shape with methods double area() and double perimeter().

Create a class Rectangle that implements Shape with properties length and width.

Create a class Circle that implements Shape with property radius.

Ensure the Rectangle and Circle classes have constructors to initialize their properties.

Implement the methods area and perimeter in both classes.

Input Format:String representing the shape type (Rectangle or Circle).If Rectangle, two doubles representing length and width.If Circle, one double representing radius.**Output Format:**Double representing the area of the shape.Double representing the perimeter of the shape.

Sample Input:Rectangle510**Sample Output:**Area: 50.0Perimeter: 30.0

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.Scanner;

interface Shape {
    double area();
    double perimeter();
}

class Rectangle implements Shape {
    //..... YOUR CODE STARTS HERE .....
    private double length;
    private double width;

    public Rectangle(double length , double width) {
        this.length = length;
        this.width = width;
    }

    public double area() {
        return length * width;
    }

    @Override

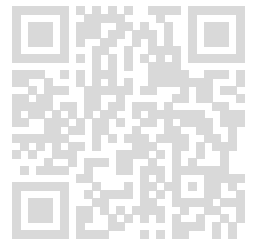
    public double perimeter() {
        return 2 * (length + width);
    }
    //..... YOUR CODE ENDS HERE .....
}

class Circle implements Shape {
    //..... YOUR CODE STARTS HERE .....
    public double radius;
    public Circle(double radius) {
        this.radius = radius;
    }

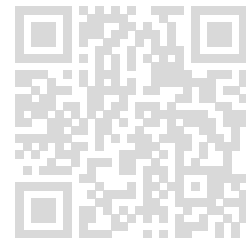
    @Override

    public double area() {
        return Math.PI * radius * radius;
    }
    public double perimeter() {
        return 2 * Math.PI * radius ;
    }
    //..... YOUR CODE ENDS HERE .....
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String shapeType = scanner.next();
        Shape shape;
```



Shaurya Singh (singhshaurya851@gmail.com)



```
if (shapeType.equals("Rectangle")) {  
    double length = scanner.nextDouble();  
    double width = scanner.nextDouble();  
    shape = new Rectangle(length, width);  
} else if (shapeType.equals("Circle")) {  
    double radius = scanner.nextDouble();  
    shape = new Circle(radius);  
} else {  
    System.out.println("Invalid shape type");  
    return;  
}  
  
System.out.println("Area: " + shape.area());  
System.out.println("Perimeter: " + shape.perimeter());  
}  
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Area: 50.0
Perimeter: 30.0

Compilation Status: Passed

Execution Time:

0.123s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

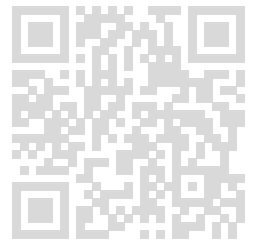
Output:

Area: 12.0
Perimeter: 14.0

Compilation Status: Passed

Execution Time:

0.126s



12. Access Modifiers

Problem Statement: Create a class Person with private properties and public methods for accessing these properties. Ensure that the age of the person is always a positive value.

Description: Define a class Person with private properties: name (String) and age (int). Include public methods setName and setAge for setting the properties. Include public methods getName and getAge for accessing the properties. Ensure that the setAge method only allows positive values.

Input Format: String representing the name of the person. Integer representing the age of the person. **Output Format:** String representing the name of the person. Integer representing the age of the person. A message "Invalid age" indicating if the age is invalid.

Sample Input 1: Alice25 Sample Output 1: Name: Alice Age: 25

Sample Input 2: Diana0 Sample Output 2: Invalid age

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

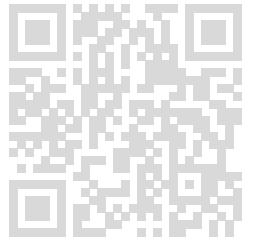
Source Code:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String name = scanner.next();
        int age = scanner.nextInt();

        Person person = new Person();
        person.setName(name);
        person.setAge(age);

        if (age > 0) {
            System.out.println("Name: " + person.getName());
            System.out.println("Age: " + person.getAge());
        }
    }
}
```



```
}  
}  
}  
  
class Person {  
//..... YOUR CODE STARTS HERE .....  
  
private String name;  
  
private int age;  
  
public void setName(String name) {  
this.name = name;  
}  
  
public void setAge(int age) {  
if(age > 0) {  
this.age = age;  
} else {  
  
System.out.println("Invalid age");  
}  
}  
public String getName() {  
return this.name ;  
}  
// age declaration...  
public int getAge() {  
return this.age;  
}  
//..... YOUR CODE ENDS HERE .....  
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Name: Alice
Age: 25

Compilation Status: Passed

Execution Time:

0.113s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Invalid age

Compilation Status: Passed

Execution Time:

0.09s

13. Polymorphic Zoo

Problem Statement:Create a program that models a zoo with animals demonstrating polymorphism. Each animal should have a speak method, but the sound they make depends on the type of animal. Implement a system that allows adding different animals to the zoo and prints their sounds.

Description:Your task is to design a Java program that uses polymorphism to model a zoo. You need to create an abstract class Animal with a method speak(). Then, create at least three subclasses (Lion, Elephant, Monkey) that override the speak() method to return their respective sounds. The program should allow the user to add animals to the zoo and then print out the sounds of all animals in the zoo.

Input Format:The first line of input contains an integer N, the number of animals. The next N lines each contain a string representing the type of animal (Lion, Elephant, Monkey). **Output Format:**The output should be the sounds of all animals in the zoo, each on a new line, in the order they were added.

Sample Input:3LionElephantMonkey**Sample Output:**RoarTrumpetOoh Ooh Aah Aah

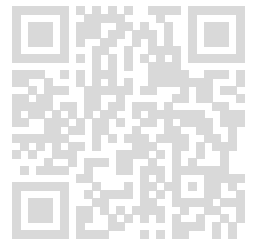
Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:



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

```
abstract class Animal {
    abstract String speak();
}
```

```
class Lion extends Animal {
    //..... YOUR CODE STARTS HERE .....
    @Override
    String speak() {
        return "Roar";
    }
    //..... YOUR CODE ENDS HERE .....

}
```

```
class Elephant extends Animal {
    //..... YOUR CODE STARTS HERE .....
    @Override
    //Override method..
    String speak() {
        return "Trumpet";
    }
    //..... YOUR CODE ENDS HERE .....
}
```

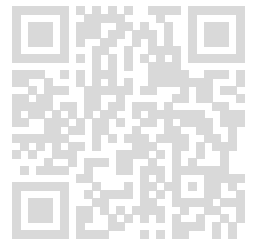
```
class Monkey extends Animal {
    //..... YOUR CODE STARTS HERE .....
    @Override
    // override method..
    String speak() {
        return "Ooh Ooh Aah Aah";
    }
    //..... YOUR CODE ENDS HERE .....
}
```

```
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();
        sc.nextLine(); // Consume the newline character
        ArrayList<Animal> zoo = new ArrayList<>();
        //..... YOUR CODE STARTS HERE .....
        for (int i = 0; i < N; i++)
        {
            String animalType = sc.nextLine();

            switch (animalType) {

                case "Lion":
                    zoo.add(new Lion());

                    break;
```



Shaurya Singh (singhshaurya851@gmail.com)

```
case "Elephant":
zoo.add(new Elephant());

break;

case "Monkey":
zoo.add(new Monkey());

break;
default:
System.out.println("Unknown animaltype");

break;
//Exits from the loop...
}
}
//..... YOUR CODE ENDS HERE .....
for (Animal animal : zoo) {
System.out.println(animal.speak());
}
}
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Roar
Trumpet
Ooh Ooh Aah Aah

Compilation Status: Passed

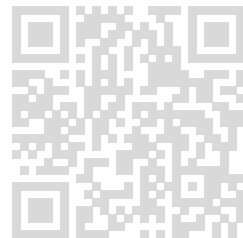
Execution Time:

0.093s

TestCase2:

Input:

< hidden >



Shaurya Singh (singhshaurya851@gmail.com)

Expected Output:

< hidden >

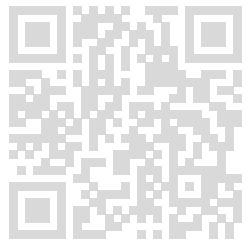
Output:

Roar

Compilation Status: Passed

Execution Time:

0.091s



14. Library Management System

Problem Statement: Design a library management system using OOP principles. The system should allow adding books, borrowing books, and returning books. Implement classes for Book and Library.

Description: Your task is to create a Java program that models a library management system. You need to create a class Book with attributes for the book title, author, and a boolean to indicate if it is borrowed. Create another class Library that contains a list of books and methods to add a book, borrow a book by title, and return a book by title.

Input Format: The first line contains an integer N, the number of operations. The next N lines contain operations in the format add <title> <author>, borrow <title>, or return <title>. **Output Format:** Print the result of each operation. For borrow, print "Book borrowed" if successful or "Book not available" if not. For return, print "Book returned". For add, print "Book added".

Sample Input: 3 add HarryPotter J.K.Rowling borrow HarryPotter return HarryPotter
Sample Output: Book added Book borrowed Book returned

Completion Status: Completed

Concepts Included:

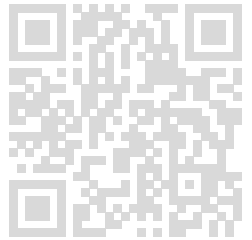
gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

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

class Book {
//..... YOUR CODE STARTS HERE .....
private String title;
private String author;
```



```
private boolean isBorrowed;

public Book(String title, String author) {
    this.title = title;
    this.author = author;
    this.isBorrowed = false;
}

public String getTitle() {
    return title;
}

public boolean isBorrowed() {

    return isBorrowed;

}

public void borrowBook() {

    this.isBorrowed = true;
}

public void returnBook() {
    this.isBorrowed = false;

}

@Override // Override method from parent class..
public String toString()
{
    return title + " by " + author;
}
//..... YOUR CODE ENDS HERE .....
}

class Library {
    //..... YOUR CODE STARTS HERE .....
    private ArrayList<Book> books;

    public Library() {
        books = new ArrayList<>();
    }

    public void addBook(String title, String author) {
        books.add(new Book(title,author));
        System.out.println("Book added");
    }

    public void borrowBook(String title) {

        for (Book book : books) {
            if (book.getTitle().equals(title) &&! book.isBorrowed()) {
                book.borrowBook();
            }
        }
    }
}
```



```
System.out.println("Book borrowed");

return;
}

}
System.out.println("Book not available");
}

public void returnBook(String title) {
for (Book book : books) {
if (book.getTitle().equals(title) && book.isBorrowed()) {

book.returnBook();

System.out.println("Book returned");

return;
}

}

System.out.println("Book not found");
}
//..... YOUR CODE ENDS HERE .....
}
```

```
public class Main {
public static void main(String[] args) {
//..... YOUR CODE STARTS HERE .....
Scanner scanner = new Scanner(System.in);
Library library = new Library();
int n = scanner.nextInt();
scanner.nextLine(); // Newline..
for (int i = 0; i < n; i++) {
String operation = scanner.nextLine();
```

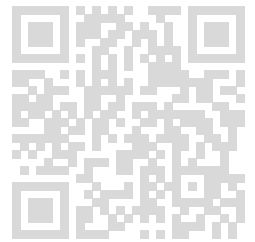
```
String[] parts = operation.split(" ", 3); // similar data type should be stored here..
```

```
switch (parts[0]) {
// multiple condition using switch case..
case "add" :
library.addBook(parts[1],parts[2]);
```

```
break; // Exits from the switch case...
```

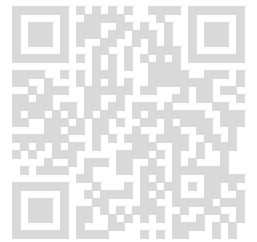
```
case "borrow":
library.borrowBook(parts[1]);
```

```
break;
// Exits from the entire loop..
case "return":
library.returnBook(parts[1]);
```



```
break;

default:
System.out.println("Invalid operation");
}
}
scanner.close();
//..... YOUR CODE ENDS HERE .....
}
}
```



Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Book added
Book borrowed
Book returned

Compilation Status: Passed

Execution Time:

0.1s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

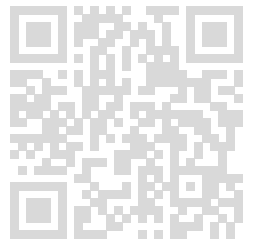
Output:

Book added
Book borrowed
Book not available
Book returned

Compilation Status: Passed

Execution Time:

Shaurya Singh (singhshaurya851@gmail.com)



15. Smart Appliance Management System Problem

Statement: Design a smart appliance management system that allows users to manage a collection of appliances in a smart home. Each appliance has a unique ID, a name, a power rating, and a status (on/off). You need to implement a class called `Appliance` to represent an appliance, and a class called `SmartHome` to manage the collection of appliances. The `SmartHome` class should provide methods to add an appliance, remove an appliance by ID, toggle the status of an appliance by ID, and calculate the total power consumption of all appliances that are currently on.

Description: Create an `Appliance` class with the following attributes: `int applianceID`, `String name`, `double powerRating`, and `boolean status`.

Implement a parameterized constructor in the `Appliance` class to initialize all attributes.

Implement methods to toggle the status of the appliance and retrieve the power rating.

The `SmartHome` class should maintain a list of `Appliance` objects.

Implement the following methods in the `SmartHome` class:

`void addAppliance(Appliance appliance):` Adds a new appliance to the system.

`boolean removeAppliance(int applianceID):` Removes an appliance from the system by its ID. Returns true if the appliance was successfully removed, false otherwise.

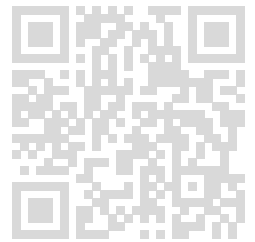
`boolean toggleApplianceStatus(int applianceID):` Toggles the status of an appliance by its ID. Returns true if the status was successfully toggled, false otherwise.

`double calculateTotalPower():` Calculates the total power consumption of all appliances that are currently on.

Input Format: The first line contains an integer `n`, the number of appliances to be added to the system. The next `n` lines each contain the details of an appliance in the following order: `applianceID`, `name`, `powerRating`, and `status` (either "on" or "off"). The next line contains an integer `m`, the number of operations to be performed. The next `m` lines each contain a string representing an operation: either "add", "remove", "toggle", or "calculate", followed by the relevant parameters (appliance details for "add", `applianceID` for "remove" and "toggle").

Output Format: For each "remove" operation, output "Appliance removed successfully" if the appliance was removed, otherwise output "Appliance not found". For each "toggle" operation, output "Appliance status toggled" if the status was toggled, otherwise output "Appliance not found". For the "calculate" operation, output the total power consumption of all appliances that are currently on.

Sample Input:2501 Air_Conditioner 1.5 on502 Refrigerator 0.8 off3toggle
502calculateremove 501Sample Output:Appliance status toggledTotal Power
Consumption: 2.3 kWAppliance removed successfully



Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

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

class Appliance {
//..... YOUR CODE STARTS HERE .....
private int applianceID;
private String name;
private double powerRating;
private boolean status;

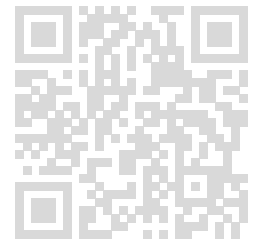
public Appliance(int applianceID, String name, double powerRating, boolean status) {

this.applianceID = applianceID;
this.name = name;
this.powerRating = powerRating;
this.status = status;
}

public void toggleStatus() {
this.status = !this.status;
}

public double getPowerRating()
{
return this.powerRating;
}

// getter for status
public boolean getStatus() {
return this.status;
}
public int getApplianceID() {
return this.applianceID;
}
//..... YOUR CODE ENDS HERE .....
}
```



```
class SmartHome {
//..... YOUR CODE STARTS HERE .....
private ArrayList<Appliance> appliances;

public SmartHome() {
this.appliances = new ArrayList<>();
}

public void addAppliance(Appliance appliances) {
this.appliances.add(appliances);
}

public boolean removeAppliance(int applianceID) {
return this.appliances.removeIf(appliances -> appliances.getApplianceID() ==
applianceID);
}

public boolean toggleApplianceStatus(int applianceID) {
for (Appliance appliance : appliances)
{
if(appliance.getApplianceID() == applianceID) {
appliance.toggleStatus();
return true;
}
}
return false;
}

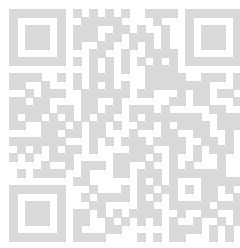
public double calculateTotalPower() {
double totalPower = 0;
for (Appliance appliance : appliances) {
if (appliance.getStatus()) {
totalPower += appliance.getPowerRating();
}
}

return totalPower;
}
//..... YOUR CODE ENDS HERE .....
}

public class Main {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
SmartHome smartHome = new SmartHome();

// Input the number of appliances
int n = scanner.nextInt();
scanner.nextLine(); // Consume newline

// Add appliances to the system
for (int i = 0; i < n; i++) {
int applianceID = scanner.nextInt();
String name = scanner.next();
double powerRating = scanner.nextDouble();
}
```



```

String status = scanner.next();
scanner.nextLine(); // Consume newline

boolean isOn = status.equalsIgnoreCase("on");
Appliance appliance = new Appliance(applianceID, name, powerRating, isOn);
smartHome.addAppliance(appliance);
}

// Input the number of operations
int m = scanner.nextInt();
scanner.nextLine(); // Consume newline

// Perform operations
for (int i = 0; i < m; i++) {
String operation = scanner.next();
switch (operation.toLowerCase()) {
case "remove":
int applianceID = scanner.nextInt();
if (smartHome.removeAppliance(applianceID)) {
System.out.println("Appliance removed successfully");
} else {
System.out.println("Appliance not found");
}
break;

case "toggle":
applianceID = scanner.nextInt();
if (smartHome.toggleApplianceStatus(applianceID)) {
System.out.println("Appliance status toggled");
} else {
System.out.println("Appliance not found");
}
break;

case "calculate":
double totalPower = smartHome.calculateTotalPower();
System.out.println("Total Power Consumption: " + totalPower + " kW");
break;
}
}
scanner.close();
}

```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Appliance status toggled
Total Power Consumption: 2.3 kW
Appliance removed successfully

Compilation Status: Passed

Execution Time:

0.132s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Appliance status toggled
Total Power Consumption: 3.0 kW
Appliance status toggled
Total Power Consumption: 4.3 kW

Compilation Status: Passed

Execution Time:

0.129s

16. Student Grading System

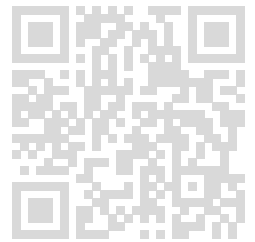
Problem Statement: Implement a student grading system using OOP principles. The system should handle multiple students and calculate their grades based on different criteria.

Description: Your task is to create a Java program that models a student grading system. You need to create a class Student with attributes like name, marks, and grade. Create methods to add marks, calculate grades, and display student details. The grade should be calculated based on the average of the marks.

Input Format: The first line contains an integer N, the number of operations. The next N lines contain operations in the format add <name> <marks>, calculate <name>, or display <name>. **Output Format:** Print the result of each operation. For add, print "Marks added". For calculate, print "Grade calculated". For display, print the student's details.

Sample Input: 4 add John 85 add John 90 calculate John display John

Sample Output: Marks added Marks added Grade calculated John: 87.5 - B



Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;

class Student {
//..... YOUR CODE STARTS HERE .....
private String name;
private double totalMarks;
private int marksCount;
private char grade;

public Student(String name) {
this.name = name;
this.totalMarks = 0;
this.marksCount = 0;
}
public void addMarks(double mark) {
totalMarks += mark;
marksCount++;
System.out.println("Marks added");
}
public void calculatedGrade() {
if(marksCount > 0) {

double average = totalMarks / marksCount;

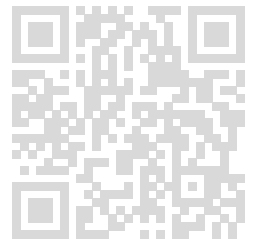
if (average >= 90) {
grade = 'A';

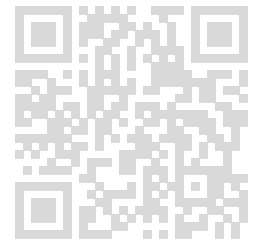
} else if (average >= 80) {
grade = 'B';

} else if (average >= 70) {
grade = 'C';

} else if (average >= 60) {
grade = 'D';

} else {
grade = 'F';
}
System.out.println("Grade calculated");
```





```
} else {
System.out.println("No Marks available to calculate grade");
}
}

public void displayDetails() {

double average = marksCount > 0 ? totalMarks / marksCount : 0;

System.out.printf("%s: %.1f - %c%n", name, average, grade); // Changed to %.1f
}
//..... YOUR CODE ENDS HERE .....
}

public class Main {
public static void main(String[] args) {
//..... YOUR CODE STARTS HERE .....
Scanner scanner = new Scanner(System.in);

HashMap <String, Student> studentMap = new HashMap<>();

int N = scanner.nextInt();

scanner.nextLine(); // new line..

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

String [] input = scanner.nextLine().split(" ");

String command = input[0];

String name = input[1];

Student student = studentMap.computeIfAbsent(name, Student::new);

if(command.equals("add")) {
double mark = Double.parseDouble(input[2]);

student.addMarks(mark);

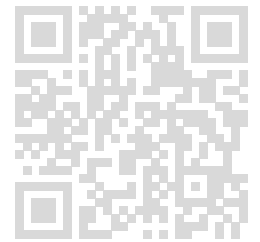
} else if (command.equals ("calculate")) {
student.calculatedGrade();

} else if (command.equals("display")) {
student.displayDetails();
}

}

scanner.close();
//..... YOUR CODE ENDS HERE .....
}
}
```

Compilation Details:



TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Marks added
Marks added
Grade calculated
John: 87.5 - B

Compilation Status: Passed

Execution Time:

0.096s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Marks added
Grade calculated
Alice: 70.0 - C

Compilation Status: Passed

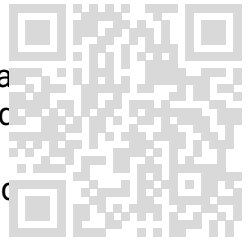
Execution Time:

0.095s

17. Unique User Activity Tracker

Problem Statement: You are tasked with developing a system to track unique user activities in a social media application. Each user can perform various activities, and you need to ensure that each activity is only logged once per user. Your task is to create a class that maintains user activity logs, ensuring that each activity for a user is unique.

Description: Implement a class UserActivityTracker with methods to add activities and display unique activities. Use a Map<String, Set<String>> where the key is the userId and the value is a Set of activities performed by that user. Implement the following methods: addActivity(String userId, String activity): Adds an activity for the specified user. If the activity already exists for the user, it should not be added again. displayActivities(String userId): Displays all unique activities performed by the specified user in a sorted order.



Input Format: The first line contains an integer n, the number of operations. Each of the next n lines contains a command followed by the necessary details: ADD_ACTIVITY userId activity DISPLAY_ACTIVITIES userId Output Format: For each DISPLAY_ACTIVITIES command, output the list of unique activities for the specified user, each on a new line in sorted order. If the user has no activities, print "No activities found."

Sample Input: 6 ADD_ACTIVITY user1 login ADD_ACTIVITY user1 post ADD_ACTIVITY user1 login ADD_ACTIVITY user2 login DISPLAY_ACTIVITIES user1 DISPLAY_ACTIVITIES user2 Sample Output: login post login

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.*;

class UserActivityTracker {
//..... YOUR CODE STARTS HERE .....

private Map<String, Set<String>> userActivities;

public UserActivityTracker() {
userActivities = new HashMap<>();
}

public void addActivity(String userId, String activity) {
userActivities.putIfAbsent(userId, new HashSet<>());
userActivities.get(userId).add(activity);
}

public void displayActivities(String userId) {
if(!userActivities.containsKey(userId)|| userActivities.get(userId).isEmpty()) {
System.out.println("No Activities Found.");
} else {
List<String>activities = new ArrayList<>(userActivities.get(userId));
Collections.sort(activities);
}
```

```
for(String activity : activities) {  
    System.out.println(activity);  
}  
}  
}  
//..... YOUR CODE ENDS HERE .....
```

```
public class Main {  
    public static void main(String[] args) {  
        //..... YOUR CODE STARTS HERE .....
```

Scanner scanner = new Scanner(System.in);
UserActivityTracker tracker = new UserActivityTracker();
int n = scanner.nextInt();
scanner.nextLine();
for(int i = 0; i < n; i++) {
 String operation = scanner.nextLine();
 String[] parts = operation.split(" ");

if(parts[0].equals("ADD_ACTIVITY")) {

String userId = parts[1];
String activity = parts[2];
tracker.addActivity(userId,activity);

}else if(parts [0].equals("DISPLAY_ACTIVITIES"))

{
String userId = parts[1];

tracker.displayActivities(userId);
}

}

scanner.close();
//..... YOUR CODE ENDS HERE

}

}

Compilation Details:

TestCase1:

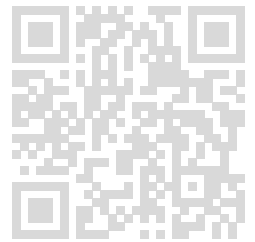
Input:

< hidden >

Expected Output:

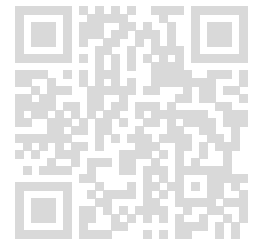
< hidden >

Output:



Shaurya Singh (singhshaurya851@gmail.com)

login
post
login



Compilation Status: Passed

Execution Time:

0.108s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

login
post

Compilation Status: Passed

Execution Time:

0.097s

18. Dynamic Vehicle Registration System

Problem Statement: You need to design a dynamic Vehicle Registration System where each vehicle can be of different types (Car, Motorcycle, or Truck) and should be able to report its registration details. Implement a base class Vehicle and three subclasses Car, Motorcycle, and Truck.

Description:

Create a base class Vehicle with:

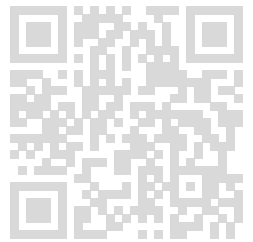
A constructor to initialize the vehicle's registration number and owner's name. A method displayDetails() to display the vehicle's details. Extend this base class with three subclasses:

Car with an additional attribute for the number of doors. Motorcycle with an additional attribute for engine capacity. Truck with an additional attribute for cargo capacity.

Input Format: First line: Integer N (Number of operations to perform) Next N lines: Each line will describe an operation in the format REGISTER <vehicle_type> <registration_number> <owner_name> <specific_attribute>. **Output Format:** For each operation, output the details of the registered vehicle.

Sample Input: 3REGISTER Car ABC123 John 4REGISTER Motorcycle XYZ789 Alice 600ccREGISTER Truck LMN456 Bob 10000kg
Sample Output: Vehicle: CarRegistration

Number: ABC123Owner: JohnNumber of Doors: 4Vehicle: MotorcycleRegistration
Number: XYZ789Owner: AliceEngine Capacity: 600ccVehicle: TruckRegistration
Number: LMN456Owner: BobCargo Capacity: 10000kg



Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

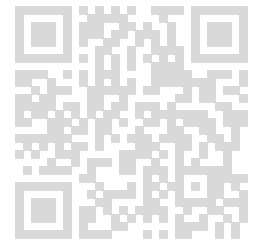
```
import java.util.Scanner;

class Vehicle {
//..... YOUR CODE STARTS HERE .....
String registrationNumber;
String ownerName;
// Constructor to initialize registration number and ownerName
public Vehicle(String registrationNumber, String ownerName) {
this.registrationNumber = registrationNumber;
this.ownerName = ownerName;
}
// Method to display Vehicle details
public void displayDetails() {
System.out.println("Registration Number: "+ registrationNumber);
System.out.println("Owner: "+ ownerName);
}
//..... YOUR CODE ENDS HERE .....
}

class Car extends Vehicle {
//..... YOUR CODE STARTS HERE .....
int numberOfDoors; //Constructor to initialize the car..
public Car(String registrationNumber, String ownerName, int numberOfDoors) {
super(registrationNumber,ownerName);
this.numberOfDoors = numberOfDoors;
}

// overriding displayDetails method to include Car-specific details
@Override

public void displayDetails() {
System.out.println("Vehicle: Car");
super.displayDetails();
System.out.println("Number of Doors: " + numberOfDoors);
}
//..... YOUR CODE ENDS HERE .....
}
```



```
class Motorcycle extends Vehicle {
//..... YOUR CODE STARTS HERE .....
String engineCapacity; // Constructor to initialize Motorcycle attributes..
```

```
public Motorcycle(String registrationNumber, String ownerName,String
engineCapacity) {
super(registrationNumber, ownerName);
this.engineCapacity = engineCapacity;
}
```

```
// overriding displayDetails()
@Override
```

```
public void displayDetails() {
System.out.println("Vehicle: Motorcycle");
super.displayDetails();
System.out.println("Engine Capacity: "+ engineCapacity);
}
//..... YOUR CODE ENDS HERE .....
}
```

```
class Truck extends Vehicle {
//..... YOUR CODE STARTS HERE .....
String cargoCapacity;
// Constructor to initialize
```

```
public Truck (String registrationNumber,String ownerName, String cargoCapacity) {
super(registrationNumber,ownerName);
this.cargoCapacity = cargoCapacity;
```

```
}
// overriding displayDetails method..
```

```
@Override
public void displayDetails() {
System.out.println("Vehicle: Truck");
super.displayDetails();
System.out.println("Cargo Capacity: " + cargoCapacity);
}
//..... YOUR CODE ENDS HERE .....
}
```

```
public class Main {
public static void main(String[] args) {
//..... YOUR CODE STARTS HERE .....
Scanner scanner = new Scanner(System.in);
int N = scanner.nextInt();
scanner.nextLine(); // New line..
for (int i = 0; i<N; i++) {
String operation = scanner.nextLine();
String[] parts = operation.split(" ");
```

```
String vehicleType = parts[1];
```



```
String registrationNumber = parts[2];
String ownerName = parts[3];
String specificAttribute = parts[4];

switch(vehicleType.toLowerCase()) {
case "car":
int numberOfDoors = Integer.parseInt(specificAttribute);
Car car = new Car(registrationNumber, ownerName,numberOfDoors);
car.displayDetails();
break;
case "motorcycle":
Motorcycle motorcycle = new
Motorcycle(registrationNumber,ownerName,specificAttribute);
motorcycle.displayDetails();
break;
case "truck":
Truck truck = new Truck(registrationNumber,ownerName,specificAttribute);
truck.displayDetails();
break;
default:
System.out.println("Unknown vehicleType!");

break;
}
}
scanner.close();
//..... YOUR CODE ENDS HERE .....
}
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Vehicle: Car
Registration Number: ABC123
Owner: John
Number of Doors: 4
Vehicle: Motorcycle
Registration Number: XYZ789
Owner: Alice
Engine Capacity: 600cc
Vehicle: Truck
Registration Number: LMN456

Owner: Bob
Cargo Capacity: 10000kg

Compilation Status: Passed

Execution Time:

0.115s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

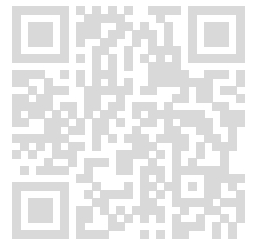
Output:

Vehicle: Car
Registration Number: DEF456
Owner: Emily
Number of Doors: 2
Vehicle: Truck
Registration Number: GHI789
Owner: Dave
Cargo Capacity: 8000kg
Vehicle: Motorcycle
Registration Number: JKL012
Owner: Carol
Engine Capacity: 400cc
Vehicle: Car
Registration Number: MNO345
Owner: Frank
Number of Doors: 4

Compilation Status: Passed

Execution Time:

0.112s



19. Exception Handling with Multiple Catch Blocks

Problem Statement: Write a Java program to handle multiple exceptions. The program should take two integers as input from the user and perform division. If the input is invalid, throw a `NumberFormatException`. If the second integer is zero, throw an `ArithmeticException`. Ensure that the program uses multiple catch blocks to handle these exceptions and always prints a final message using the `finally` block.

Description: The program should prompt the user to enter two integers. It should then attempt to divide the first integer by the second. If the user enters a non-integer value, handle the `NumberFormatException`. If the second integer is zero, handle the

ArithmeticException. Regardless of whether an exception occurs or not, print "Operation Completed" using the finally block.

Input Format: The first line contains an integer, a. The second line contains an integer b. Output Format: If a and b are valid integers and b is not zero, print the result of a / b. If a or b is not a valid integer, print "Invalid input". If b is zero, print "Cannot divide by zero". Always print "Operation Completed" at the end.

Sample Input: 10 2 Sample Output: 5 Operation Completed

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

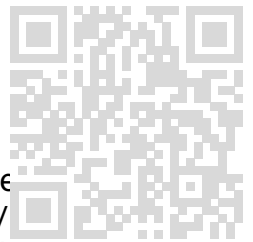
```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        //..... YOUR CODE STARTS HERE .....
        int a;
        int b;
        try{
            a = Integer.parseInt(scanner.nextLine());
            b = Integer.parseInt(scanner.nextLine());
            int result = a / b;
            System.out.println(result);
        } catch (NumberFormatException e) {
            System.out.println("Invalid input");
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by Zero");
        } finally {
            System.out.println("Operation Completed");
        }

        scanner.close();
        // scanner close after taken input from user...
        //..... YOUR CODE ENDS HERE .....
    }
}
```

Compilation Details:

TestCase1:



Input:

< hidden >

Expected Output:

< hidden >

Output:

5
Operation Completed

Compilation Status: Passed

Execution Time:

0.094s

TestCase2:**Input:**

< hidden >

Expected Output:

< hidden >

Output:

Invalid input
Operation Completed

Compilation Status: Passed

Execution Time:

0.089s

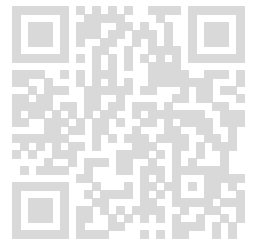
20. Custom Exception for Age Validation

Problem Statement:Create a custom exception called `InvalidAgeException`. Write a Java program that takes the age of a user as input and throws this custom exception if the age is less than 18. Ensure the program catches the exception and prints an appropriate message.

Description:Define a custom exception `InvalidAgeException` that extends the `Exception` class. The program should prompt the user to enter their age. If the age is less than 18, throw the `InvalidAgeException` with a message "Age must be 18 or older". Catch this exception and print the message to the user.

Input Format:The first line contains an integer, age.**Output Format:**If age is 18 or older, print "Age is valid".If age is less than 18, print "Age must be 18 or older".

Sample Input:20**Sample Output:**Age is valid



Shaurya Singh (singhshaurya851@gmail.com)

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.Scanner;

class InvalidAgeException extends Exception {
//..... YOUR CODE STARTS HERE .....
public InvalidAgeException(String message) {
super(message);
}
//..... YOUR CODE ENDS HERE .....
}

public class Main {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
//..... YOUR CODE STARTS HERE .....
int age = scanner.nextInt();
try {
if(age < 18) {
throw new InvalidAgeException("Age must be 18 or older");
}

System.out.println("Age is valid");

} catch (InvalidAgeException e) {
System.out.println(e.getMessage());
}
//..... YOUR CODE ENDS HERE .....
}
}
```

Compilation Details:

TestCase1:

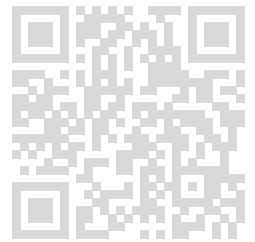
Input:

< hidden >

Expected Output:

< hidden >

Output:



Age is valid

Compilation Status: Passed

Execution Time:

0.09s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Age must be 18 or older

Compilation Status: Passed

Execution Time:

0.089s

21. Nested Exception Handling

Problem Statement: Write a Java program that reads a list of integers from the user and calculates their sum. If any non-integer value is entered, handle the exception and continue reading the next value. Additionally, ensure that the program terminates gracefully by printing the final sum, even if an exception occurs during the input process.

Description: Your task is to implement nested exception handling in Java. The program should continue reading integers from the user until a non-integer value is entered. If a non-integer value is encountered, the program should catch the exception and continue reading the next value. Use nested try-catch blocks to achieve this. Finally, ensure that the program prints the sum of all entered integers using a finally block.

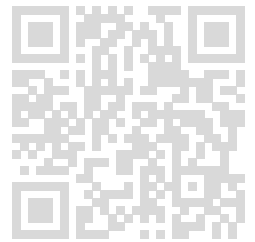
Input Format: The input consists of a sequence of integers and non-integer values entered by the user. **Output Format:** The output should display the sum of all entered integers.

Sample Input: 1 2 3 a 4 **Sample Output:** Sum of entered integers: 10

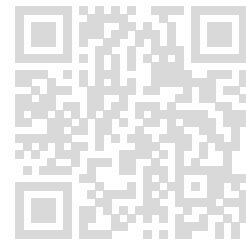
Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming



Language Used: JAVA



Source Code:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        //..... YOUR CODE STARTS HERE .....
        int sum = 0;

        while (true) {

            try {

                String input = scanner.next();

                try {

                    int number = Integer.parseInt(input);

                    sum += number;

                } catch (NumberFormatException e) {
                    continue;
                }

                } catch (Exception e) {

                    break;

                }

            }

            System.out.println("Sum of entered integers: " + sum);

            scanner.close();
            //..... YOUR CODE ENDS HERE .....
        }
    }
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Sum of entered integers: 10

Compilation Status: Passed**Execution Time:**

0.098s

TestCase2:**Input:**

< hidden >

Expected Output:

< hidden >

Output:

Sum of entered integers: 25

Compilation Status: Passed**Execution Time:**

0.096s

22. Exception Handling in Array Operations

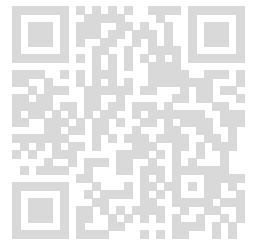
Problem Statement: Write a Java program that performs array operations. The program should read an integer n from the user, create an array of size n , and allow the user to fill the array. If the user tries to access an index outside the array bounds, handle the `ArrayIndexOutOfBoundsException` and display an appropriate message.

Description: Your task is to implement exception handling for array operations. The program should read the size of the array from the user and then allow the user to fill the array with integers. If the user tries to access or modify an index outside the bounds of the array, catch the `ArrayIndexOutOfBoundsException` and display a message indicating the error. Ensure the program continues to run gracefully after handling the exception.

Input Format: The first input is an integer n representing the size of the array. The next n inputs are the elements of the array. The next input is the index of the array. **Output Format:** The output should display the elements of the array or an appropriate error message if an out-of-bounds access occurs.

Sample Input 1: 1 1002 **Sample Output 1:** `ArrayIndexOutOfBoundsException: Index 2 out of bounds for length 1`

Sample Input 2: 3 10 20 30 **Sample Output 2:** Element at index 1: 20



Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Read the size of the array
        int n = scanner.nextInt();
        int[] array = new int[n];

        // Read the elements of the array
        for (int i = 0; i < n; i++) {
            array[i] = scanner.nextInt();
        }

        // Read the index to access
        int index = scanner.nextInt();

        try {
            // Try to access and print the element at the given index
            System.out.println("Element at index " + index + ": " + array[index]);
        } catch (ArrayIndexOutOfBoundsException e) {
            // Catch the ArrayIndexOutOfBoundsException and print an error message
            System.out.println("ArrayIndexOutOfBoundsException: Index " + index + " out of
            bounds for length " + n);
        }
    }
}
```

Compilation Details:

TestCase1:

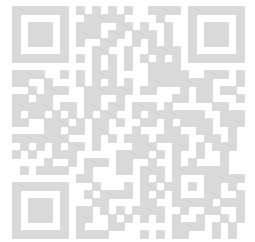
Input:

< hidden >

Expected Output:

< hidden >

Output:



ArrayIndexOutOfBoundsException: Index 2 out of bounds for length 1

Compilation Status: Passed

Execution Time:

0.112s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Element at index 1: 20

Compilation Status: Passed

Execution Time:

0.129s

23. Custom Exception for Age Verification

Problem Statement:Create a Java program that verifies the age entered by the user. If the age is below 18, throw a custom exception called UnderageException. Handle the exception and display an appropriate message.

Description:Your task is to implement a custom exception in Java. The program should read the age from the user. If the age is below 18, throw a custom exception called UnderageException. Catch the exception and display a message indicating that the user is underage. Ensure that the program continues to execute gracefully after handling the exception.

Input Format:The input consists of a single integer representing the age.

Output Format:The output should display an appropriate message based on the age verification.

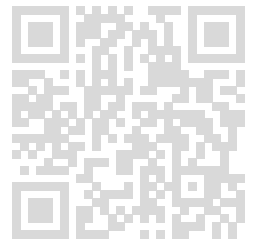
Sample Input 1:17**Sample output 1:**UnderageException: Age 17 is below the legal age limit.

Sample Input 2:20**Sample output 2:**Age verification successful.

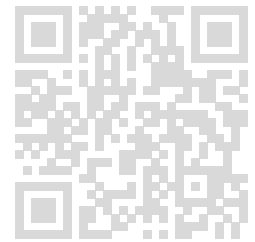
Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming



Language Used: JAVA



Source Code:

```
import java.util.Scanner;

class UnderageException extends Exception {
//..... YOUR CODE STARTS HERE .....

public UnderageException(String message) {
super(message);
}
//..... YOUR CODE ENDS HERE .....
}

public class Main {
public static void main(String[] args) {
//..... YOUR CODE STARTS HERE .....
Scanner scanner = new Scanner(System.in);

int age = scanner.nextInt();

try {

if (age < 18) {

throw new UnderageException("UnderageException: Age " + age + " is below the legal
age limit.");

} else {

System.out.println("Age verification successful.");
}

} catch ( UnderageException e) {

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

}

//..... YOUR CODE ENDS HERE .....
}
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

UnderageException: Age 17 is below the legal age limit.

Compilation Status: Passed

Execution Time:

0.114s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Age verification successful.

Compilation Status: Passed

Execution Time:

0.089s

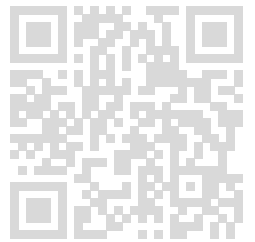
24. Custom Exception for Invalid User Input

Problem Statement:You are developing a user management system where users can register with a unique username. Implement a system where a custom exception is thrown when a user attempts to register with a username that contains forbidden characters or is too short.

Description:Create a custom exception named `InvalidUsernameException` that extends the `Exception` class. This exception should be thrown if the username is less than 5 characters long or contains any non-alphanumeric characters. The `UserManager` class should handle this exception and inform the user of the error.

Input Format:A single line of input, the username to be validated.**Output Format:**If the username is valid, output "Username registered successfully."If the username is invalid, output "Invalid username: [error details]."

Sample Input:john_doe**Sample Output:**Invalid username: Contains non-alphanumeric characters.



Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.Scanner;

class InvalidUsernameException extends Exception {
//..... YOUCoDE STARTS HERE .....

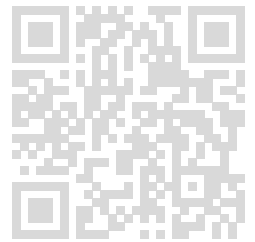
public InvalidUsernameException(String message) {
super(message);
}

//..... YOUR CODE ENDS HERE .....
}
// UserManager Class
class UserManager {
//..... YOUR CODE STARTS HERE .....

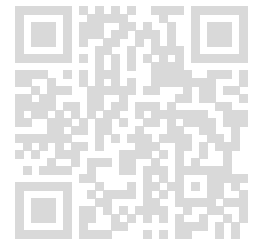
public void registerUser(String username) throws InvalidUsernameException {
if (username.length() < 5) {
throw new InvalidUsernameException("Too short.");
}
if (!username.matches("[a-zA-Z0-9]+")) {
throw new InvalidUsernameException("Contains non-alphanumeric characters.");
}
System.out.println("Username registered successfully.");
}

//..... YOUR CODE ENDS HERE .....
}
// Main Class
public class Main {
public static void main(String[] args) {
//..... YOUR CODE STARTS HERE .....

Scanner scanner = new Scanner(System.in);
UserManager userManager = new UserManager();
String username = scanner.nextLine();
try {
userManager.registerUser(username);
} catch (InvalidUsernameException e) {
System.out.println("Invalid username: " + e.getMessage());
}
}
```



```
//..... YOUR CODE ENDS HERE .....  
}  
}
```



Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Invalid username: Contains non-alphanumeric characters.

Compilation Status: Passed

Execution Time:

0.089s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Invalid username: Too short.

Compilation Status: Passed

Execution Time:

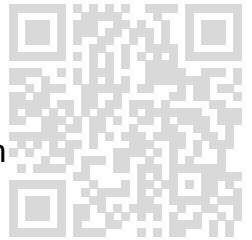
0.087s

25. Handling Multiple Exceptions

Problem Statement:Create a Java program that reads two integers from the user and performs division. Handle `ArithmeticException` for division by zero and `NumberFormatException` for invalid integer input. Ensure the program prints a message for each exception and continues execution gracefully.

Description:Your task is to handle multiple exceptions in a Java program. The program should read two integers from the user and perform division. If the user

enters a non-integer value, catch the `NumberFormatException` and display an appropriate message. If the user attempts to divide by zero, catch the `ArithmeticException` and display an appropriate message. Ensure that the program continues to execute gracefully after handling each exception.



Input Format:The first input is the dividend.The second input is the divisor.**Output Format:**The output should display the result of the division or an appropriate error message if an exception occurs.

Sample Input 1:102Sample Output 1:Result: 5

Sample Input 2:10 0Sample Output 2:ArithmeticException: Division by zero is not allowed.

Sample Input 3:a 5Sample Output 3:NumberFormatException: Invalid input. Please enter integers only.

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        //..... YOUR CODE STARTS HERE .....

        try {

            String dividendInput = scanner.next();
            String divisorInput = scanner.next();

            int dividend = Integer.parseInt(dividendInput);
            int divisor = Integer.parseInt(divisorInput);

            int result = dividend / divisor;

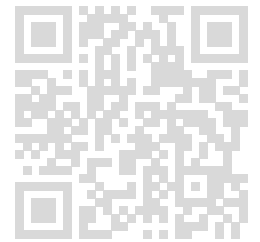
            System.out.println("Result: " + result);

        } catch (NumberFormatException e) {

            System.out.println("NumberFormatException: Invalid input. Please enter integers only.");

        } catch (ArithmeticException e) {
```

```
System.out.println("ArithmeticException: Division by zero is not allowed.");
}  
//..... YOUR CODE ENDS HERE .....  
}  
}
```



Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Result: 5

Compilation Status: Passed

Execution Time:

0.098s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

ArithmeticException: Division by zero is not allowed.

Compilation Status: Passed

Execution Time:

0.084s

26. Reordering Elements in a List

Problem Statement: Given a list of integers, reorder the list so that all odd numbers come before all even numbers while maintaining the relative order of odd and even numbers.

Description: You need to read a list of integers from the user and reorder it such that all odd numbers come before all even numbers. The relative order of odd and even numbers should be preserved. You must use Java's List interface and iterators.

Input Format: The first line contains an integer, N ($1 \leq N \leq 100$), denoting the number of elements in the list. The second line contains N space-separated integers. **Output Format:** Print the reordered list, with all odd numbers appearing before all even numbers, maintaining their relative order.

Sample Input: 5 4 3 2 1 5 **Sample Output:** 3 1 5 4 2

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        //..... YOUR CODE STARTS HERE .....

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        List<Integer> numbers = new ArrayList<>();

        for(int i = 0; i < n; i++) {
            numbers.add(scanner.nextInt());
        }

        List<Integer> odds = new ArrayList<>();
        List<Integer> evens = new ArrayList<>();

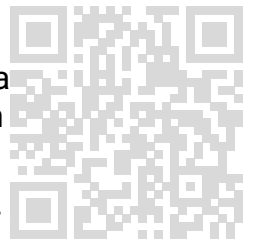
        for(int number : numbers) {

            if(number % 2 != 0) {

                odds.add(number);

            } else {

                evens.add(number);
            }
        }
    }
}
```



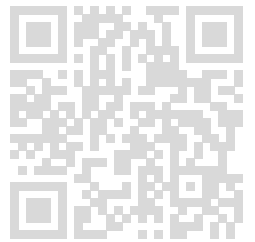

```
odds.addAll(evens);

for (int num : odds) {

System.out.print(num + " ");

}

//..... YOUR CODE ENDS HERE .....
}
}
```



Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3 1 5 4 2

Compilation Status: Passed

Execution Time:

0.107s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

21 43 65 10 32 54

Compilation Status: Passed

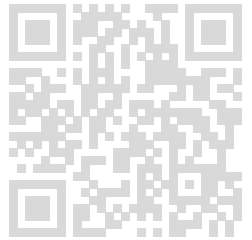
Execution Time:

0.107s

Shaurya Singh (singhshaurya851@gmail.com)

27. Custom Exception for Bank Account Balance

Problem Statement: You are designing a banking application where users can withdraw money from their account. Implement a custom exception that is thrown if a withdrawal amount exceeds the account balance.



Description: Create a custom exception named `InsufficientFundsException` that extends `Exception`. This exception should be thrown if a withdrawal attempt exceeds the account balance. The `BankAccount` class should handle this exception and provide appropriate feedback.

Input Format: The initial balance of the account (a floating-point number). The withdrawal amount (a floating-point number). **Output Format:** If the withdrawal amount is valid, output "Withdrawal successful. Remaining balance: [amount]". If the withdrawal amount exceeds the balance, output "Insufficient funds: [error details]".

Sample Input 1: 500600 **Sample Output 1:** Insufficient funds: Withdrawal exceeds balance.

Sample Input 2: 200200 **Sample Output 2:** Withdrawal successful. Remaining balance: 0.0

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.Scanner;

//Custom Exception Class

class InsufficientFundsException extends Exception {
//..... YOUR CODE STARTS HERE .....

public InsufficientFundsException(String message) {
super(message);
}

//..... YOUR CODE ENDS HERE .....
}

// BankAccount Class
class BankAccount {
//..... YOUR CODE STARTS HERE .....

private double balance;
public BankAccount(double balance) {
this.balance = balance;
```

```
}  
public void withdraw(double amount) throws InsufficientFundsException {  
    if (amount > balance) {  
        throw new InsufficientFundsException("Withdrawal exceeds balance.");  
    }  
    balance -= amount;  
    System.out.println("Withdrawal successful. Remaining balance: " + balance);  
}
```

```
//..... YOUR CODE ENDS HERE .....
```

```
}  
// Main Class  
public class Main {  
    public static void main(String[] args) {  
        //..... YOUR CODE STARTS HERE .....
```

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

```
        // Input for initial balance and withdrawal amount  
        double initialBalance = scanner.nextDouble();  
        double withdrawalAmount = scanner.nextDouble();
```

```
        BankAccount account = new BankAccount(initialBalance);  
        try {  
            account.withdraw(withdrawalAmount);  
        } catch (InsufficientFundsException e) {  
            System.out.println("Insufficient funds: " + e.getMessage());  
        }
```

```
//..... YOUR CODE ENDS HERE .....
```

```
}  
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

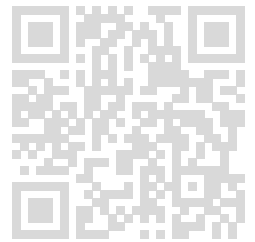
Output:

Insufficient funds: Withdrawal exceeds balance.

Compilation Status: Passed

Execution Time:

0.103s



TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Withdrawal successful. Remaining balance: 500.0

Compilation Status: Passed

Execution Time:

0.118s

28. Counting Unique Words

Problem Statement:Read a string from the user and count the number of unique words using a Set.

Description:You need to read a string of text from the user and count how many unique words are present in it. Words are considered case-insensitive and separated by spaces. Use Java's Set interface to achieve this.

Input Format:The input consists of a single line containing a string of words.**Output Format:**Print the number of unique words in the string.

Sample Input:Hello world hello**Sample Output:**2

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

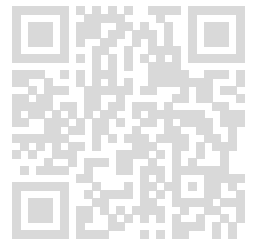
Language Used: JAVA

Source Code:

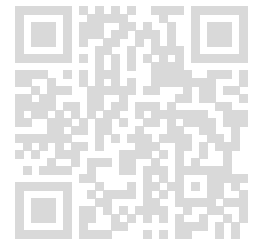
```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        //..... YOUR CODE STARTS HERE .....
        Scanner scanner = new Scanner(System.in);

        String input = scanner.nextLine().toLowerCase();
```



```
String[] words = input.split("\\s+");  
  
Set <String> uniqueWords = new HashSet <>(Arrays.asList(words));  
  
System.out.println(uniqueWords.size());  
  
//..... YOUR CODE ENDS HERE .....  
}  
}
```



Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

2

Compilation Status: Passed

Execution Time:

0.086s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

5

Compilation Status: Passed

Execution Time:

0.088s

29. Intersection of Two Sets

Problem Statement: Find the intersection of two sets of integers.

Description: You need to read two sets of integers from the user and print their intersection. Use Java's Set interface and its operations to achieve this.

Input Format: The first line contains an integer, n ($1 \leq n \leq 100$), denoting the number of elements in the first set. The second line contains n space-separated integers. The third line contains an integer, m ($1 \leq m \leq 100$), denoting the number of elements in the second set. The fourth line contains m space-separated integers. **Output Format:** Print the elements in the intersection of the two sets, one per line. If there are no common elements, print "No common elements."

Sample Input: 41 2 3 4 3 3 4 5 **Sample Output:** 3 4

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        //..... YOUR CODE STARTS HERE .....

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        Set<Integer> set1 = new HashSet<>();

        for (int i = 0; i < n; i++) {
            set1.add(scanner.nextInt());
        }
        int m = scanner.nextInt();

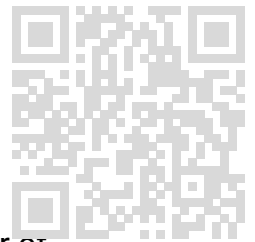
        Set<Integer> set2 = new HashSet<>();

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

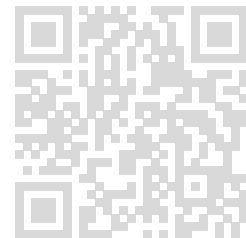
            set2.add(scanner.nextInt());
        }
        set1.retainAll(set2);

        if (set1.isEmpty()) {

            System.out.println("No common elements.");
```



```
} else {  
for (int element : set1) {  
  
System.out.println(element);  
}  
}  
//..... YOUR CODE ENDS HERE .....  
}  
}
```



Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

File "script-3.6.0.py", line 1
import java.util.*;
^
SyntaxError: invalid syntax

Runtime Error (NZEC)

Compilation Status: Failed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

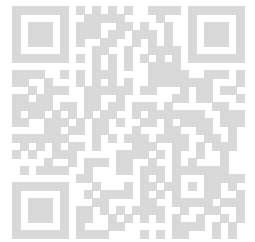
File "script-3.6.0.py", line 1
import java.util.*;
^
SyntaxError: invalid syntax

Runtime Error (NZEC)

Compilation Status: Failed

Execution Time:

0.011s



30. Creating a Frequency Map

Problem Statement: Read a list of words and create a frequency map of each word.

Description: You need to read a list of words from the user and create a map where the keys are the words and the values are their respective frequencies. Use Java's Map interface to store and calculate the frequencies.

Input Format: The first line contains an integer, n ($1 \leq n \leq 100$), denoting the number of words. The second line contains n space-separated words. **Output Format:** Print each word and its frequency, one per line.

Sample Input: 4 apple banana apple grape **Sample Output:** banana 1 apple 2 grape 1

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

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

```
public class Main {  
    public static void main(String[] args) {  
        //..... YOUR CODE STARTS HERE .....
```

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

```
        int n = scanner.nextInt();
```

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

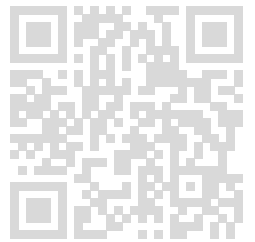
```
        String input = scanner.nextLine();
```

```
        String[] words = input.split("\\s+");
```

```
        Map <String,Integer> frequencyMap = new HashMap<>();
```



```
for(String word : words) {  
  
frequencyMap.put(word, frequencyMap.getOrDefault(word, 0) + 1);  
}  
  
for(Map.Entry<String, Integer> entry : frequencyMap.entrySet()) {  
  
System.out.println(entry.getKey() + " " + entry.getValue());  
}  
  
//..... YOUR CODE ENDS HERE .....  
}  
}
```



Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

banana 1
apple 2
grape 1

Compilation Status: Passed

Execution Time:

0.111s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

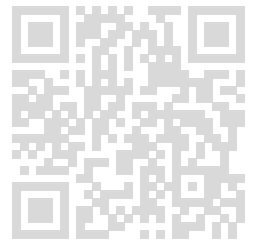
cat 1
fish 1
dog 3

Shaurya Singh (singhshaurya851@gmail.com)

Compilation Status: Passed

Execution Time:

0.111s



31. Unique Items in a Shopping List

Problem Statement: You are developing a shopping application where users can maintain their shopping lists. The application needs to handle scenarios where users can add, remove, and view items in their shopping list. The unique feature of the application is that it must manage items efficiently while ensuring that no duplicates are allowed and each item is stored in a way that preserves its insertion order.

Description: You need to implement a Java program that:

Allows users to add items to a shopping list. Allows users to remove items from the list. Displays the current shopping list without duplicates, maintaining the order of insertion.

Input Format: First line: Integer N (Number of operations to perform) Next N lines: Each line will contain an operation in the format "ADD <item>" or "REMOVE <item>". The item is a string with spaces. **Output Format:** After processing all operations, print the final shopping list with items in their insertion order, separated by commas.

Sample Input: 5ADD ApplesADD BananasADD ApplesREMOVE BananasADD Grapes
Sample Output: Apples, Grapes

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

```
import java.util.LinkedHashSet;
import java.util.Scanner;
import java.util.Set;
```

```
public class Main {
    public static void main(String[] args) {
        //..... YOUR CODE STARTS HERE .....
```

```
        Scanner scanner = new Scanner(System.in);
        int N = scanner.nextInt();
        scanner.nextLine();
        LinkedHashSet <String> shoppingList = new LinkedHashSet<>();
        for(int i = 0; i < N; i++) {
```

```
String operation = scanner.nextLine();
String[] parts = operation.split(" ", 2);
String command = parts[0];
String item = parts.length > 1 ? parts[1] : "";
```

```
if (command.equals("ADD")) {

shoppingList.add(item);

} else if (command.equals("REMOVE")) {

shoppingList.remove(item);
}

}

System.out.println(String.join(" ", shoppingList));

//..... YOUR CODE ENDS HERE .....
}
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Apples, Grapes

Compilation Status: Passed

Execution Time:

0.095s

TestCase2:

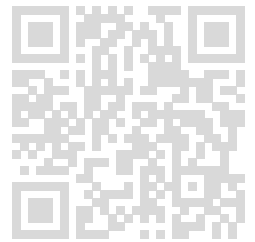
Input:

< hidden >

Expected Output:

< hidden >

Output:



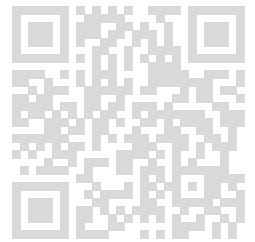
Shaurya Singh (singhshaurya851@gmail.com)

Apples, Bananas, Grapes

Compilation Status: Passed

Execution Time:

0.094s



32. Frequency of Words in Text

Problem Statement: You are tasked with implementing a feature that counts the frequency of each word in a given text. The twist is that the word frequencies need to be sorted in descending order of frequency, and in case of a tie, the words should be sorted lexicographically.

Description: You need to implement a Java program that:

Reads a block of text from the user. Counts the frequency of each word in the text. Outputs the words and their frequencies sorted by frequency (highest first) and lexicographically in case of ties.

Input Format: A single line of text (can contain multiple words and punctuation). **Output Format:** Each line should contain a word followed by its frequency, sorted by frequency in descending order and lexicographically for tied frequencies.

Sample Input: apple banana apple fruit banana apple **Sample Output:** apple 3 banana 2 fruit 1

Completion Status: Completed

Concepts Included:

gu 27 3rd semester java programming

Language Used: JAVA

Source Code:

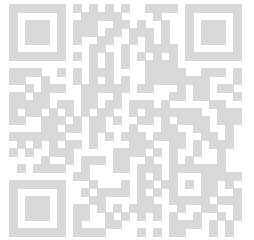
```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        //..... YOUR CODE STARTS HERE .....

        Scanner scanner = new Scanner(System.in);

        String input = scanner.nextLine();

        String[] words = input.split("\\W+");
        // Split by non-word characters
        Map<String, Integer> frequencyMap = new HashMap<>();
```



```
for (String word : words) {  
    if (!word.isEmpty()) {  
        frequencyMap.put(word, frequencyMap.getOrDefault(word, 0) + 1);  
    }  
}  
  
List<Map.Entry<String, Integer>> sortedEntries = new  
    ArrayList<>(frequencyMap.entrySet());  
  
sortedEntries.sort((entry1, entry2) -> {  
  
    int freqCompare = entry2.getValue().compareTo(entry1.getValue());  
  
    return (freqCompare != 0) ? freqCompare :  
        entry1.getKey().compareTo(entry2.getKey());  
});  
  
for (Map.Entry<String, Integer> entry : sortedEntries) {  
    System.out.println(entry.getKey() + " " + entry.getValue());  
}  
//..... YOUR CODE ENDS HERE .....  
}  
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

apple 3
banana 2
fruit 1

Compilation Status: Passed

Execution Time:

0.112s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

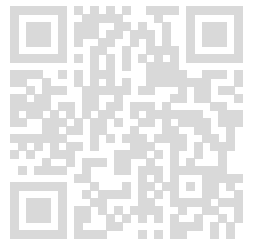
Output:

hello 3
world 2

Compilation Status: Passed

Execution Time:

0.114s



Shaurya Singh (singhshaurya851@gmail.com)