BIZOTIC

JAVA

CODE: TECH 201 to 215
TRAINER HANDOUT

TABLE OF CONTENTS

MODULE 1 – TECH 201 - INTRODUCTION TO JAVA

THEORY CONCEPTS:

1. Introduction,Application of java,Features of Java - **25 Minutes**
2. JDK,JRE and JVM - **15 Minutes**
3. Data types, variables, and constants - **20 Minutes**

PROGRAM 1: 15 MINUTES

Ritik is working on a science project where he wants to build a magic board. This board should display a corresponding character for a given number input. However, Ritik wants the program to convert exactly four numbers into their respective ASCII characters without using any loops or iteration. Your task is to help Ritik develop this application.

Sample Input 1:

```
Enter the digits:
65
66
67
68
```

Sample Output 1:

```
65-A
66-B
67-C
68-D
```

Solution:

```
public class MagicBoard {
    public static void main(String[] args) {
        // Initialize the four digits directly
        int first = 65;
        int second = 66;
        int third = 67;
        int fourth = 68;
        // Print each number with its corresponding character
        System.out.println(first + " - " + (char) first);
        System.out.println(second + " - " + (char) second);
        System.out.println(third + " - " + (char) third);
        System.out.println(fourth + " - " + (char) fourth);
    }
}
```

PROGRAM 2: 15 MINUTES

Arjun is leading a software development project at a tech company in Bangalore. To ensure consistency, he asks each team member, including Priya, Sameer, and Kavya, to check their JDK version to confirm they are all using at least Java 11. Arjun needs to provide them with a utility program that can be run on their machines to check their JDK version and display a message recommending an upgrade if they are on an older version.

Task:

Develop a Java program that each team member can use to check their JDK version. If the version is less than Java 11, the program should display a message advising them to upgrade.

Code:

```
public class JDKVersionCheck {
    public static void main(String[] args) {
        String version = System.getProperty("java.version");
        System.out.println("JDK Version: " + version);
        if (version.startsWith("1.") || Integer.parseInt(version.split("\\.")[0]) < 11) {
                    System.out.println("Warning: You are using an older version of Java. It's
recommended to upgrade to Java 11 or higher for better performance and features.");
        } else {
            System.out.println("You are using a compatible version of Java.");
        }
    }
}
```

MODULE 2 – TECH 202 - TAKING INPUT IN JAVA(SCANNER CLASS), OPERATORS AND EXPRESSION

THEORY CONCEPTS:

1. Using Scanner class for input handling,Reading different types of data (integers, doubles, strings) - **20 Minutes**
2. Operators and Expressions - **40 Minutes**
3. Operator Precedence and Associativity - **10 Minutes**
4. Implicit and explicit type casting. - **10 Minutes**

PROGRAM 1: 15 MINUTES

Cricket Stadium

There was a large ground in the centre of the city which is rectangular in shape. The Corporation decided to build a Cricket stadium in the area for school and college students, but the area was used as a car parking zone. In order to protect the land from being used as an unauthorised parking zone, the corporation wanted to protect the stadium by building a fence. In order to help the workers to build a fence, they planned to place a thick rope around the ground. They wanted to buy only the exact length of the rope that is needed. They also wanted to cover the entire ground with a carpet during the rainy season. They wanted to buy only the exact quantity of carpet that is needed. They requested your help. Can you please help them by writing a program to find the exact length of the rope and the exact quantity of carpet that is required? Input format: Input consists of 2 integers. The first integer corresponds to the length of the ground and the second integer corresponds to the breadth of the ground. Output Format: Output Consists of two integers. The first integer corresponds to the length. The second integer corresponds to the quantity of carpet required.

Sample Input:

```
50
20
```

Sample Output:

```
140
```

```
1000
```

Code:

```
import java.util.Scanner;

class Main
{
    public static void main(String args[])
    {
        Scanner in = new Scanner(System.in);
        int l = in.nextInt();
        int b = in.nextInt();
        int perimeter = 2*(l+b);
        int area = l*b;
        System.out.println(perimeter+"\n"+area);
    }
};
```

PROGRAM 2: 15 MINUTES

The newspaper Agency

Each Sunday, a newspaper agency sells w copies of a special edition newspaper for Rs.x per copy. The cost to the agency of each newspaper is Rs.y. The agency pays a fixed cost for storage, delivery and so on of Rs.100 per Sunday. The newspaper agency wants to calculate the profit which it obtains only on Sundays. Can you please help them out by writing a program to compute the profit if w, x, and y are given.

Input Format: Input consists of 3 integers: w, x, and y. w is the number of copies sold, x is the cost per copy and y is the cost the agency spends per copy.
Output Format: The output consists of a single integer which corresponds to the profit obtained by the newspaper agency.

SAMPLE INPUT:

```
1000
2
1
```

SAMPLE OUTPUT:

```
900
```

Code:

```
import java.util.Scanner;

class Main
{
    public static void main(String args[])
    {
        Scanner in = new Scanner(System.in);
        int copies = in.nextInt();
        int SP = in.nextInt();
        int CP = in.nextInt();
        int profit = copies*(SP-CP)-100;
        System.out.println(profit);
    }
};
```

ADDITIONAL QUESTIONS:

<u>PROGRAM 3: 20 MINUTES</u>

Dept Repay

Alice wanted to start a business and she was looking for a venture capitalist. Through her friend Bob, she met the owner of a construction company who is interested in investing in an emerging business. Looking at the business proposal, the owner was very much impressed with Alice's work. So he decided to invest in Alice's business and hence gave a green signal to go ahead with the project. Alice bought Rs. X for a period of Y years from the owner at R% interest per annum. Find the rate of interest and the total amount to be given by Alice to the owner. The owner, impressed by proper repayment of the financed amount, decides to give a special offer of 2% discount on the total interest at the end of the settlement. Find the amount given back by Alice and also find the total amount. (Note: All rupee values should be in two decimal points).

Input Format: Input consists of 3 integers. The first integer corresponds to the principal amount borrowed by Alice. The second integer corresponds to the rate of interest The third integer corresponds to the number of years.
Output Format: The output consists of 4 floating point values. The first value corresponds to the interest. The second corresponds to the amount. The third value corresponds to the discount. The last value corresponds to the final settlement. All floating point values are to be rounded off to two decimal places

Sample Input:

```
100
1
10
```

Sample Output:

```
10.00
110.00
0.20
109.80
```

Code:

```
import java.util.Scanner;

public class DebtRepay {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Input principal, rate, and time
        double P = scanner.nextDouble(); // Principal amount
        double R = scanner.nextDouble(); // Rate of interest
        double T = scanner.nextDouble(); // Time period in years

        // Step 1: Calculate Interest
        double interest = (P * R * T) / 100;

        // Step 2: Calculate Total Amount
        double amount = P + interest;

        // Step 3: Calculate Discount
        double discount = interest * 0.02;

        // Step 4: Calculate Final Settlement Amount
        double finalSettlement = amount - discount;

        // Print results, formatted to two decimal places
        System.out.printf("%.2f\n", interest);
        System.out.printf("%.2f\n", amount);
        System.out.printf("%.2f\n", discount);
        System.out.printf("%.2f\n", finalSettlement);

        scanner.close();
    }
}
```

PROGRAM 4: 20 MINUTES

The Chronicles of Narnia
Four kids Peter, Susan, Edmond and Lucy travel through a wardrobe to the land of Narnia. Narnia is a fantasy world of magic with mythical beasts and talking animals. While exploring

the land of Narnia Lucy found Mr. Tumnus the two-legged stag, and she followed it down a narrow path. She and Mr. Tumnus became friends and he offered a cup of coffee to Lucy in his small hut. It was time for Lucy to return to her family and so she bid good bye to Mr. Tumnus and while leaving Mr. Tumnus said that it is quite difficult to find the route back as it was already dark. He told her to see the trees while returning back and said that the first tree with two digits number will help her find the way and the way to go back to her home is the sum of digits of the tree and that numbered way will lead her to the tree next to the wardrobe where she can find the others. Lucy was already confused, so pls help her in finding the route to her home.... Input Format: Input consists of an integer corresponding to the 2-digit number. Output Format: Output consists of an integer corresponding to the sum of its digits.

SAMPLE INPUT:

87

SAMPLE OUTPUT:

15

Code:

```
import java.util.Scanner;

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

        int tens = number / 10;
        int ones = number % 10;
        int sum = tens + ones;
        System.out.println(sum);
    }
}
```

MODULE 3 – TECH 203 - CONDITIONAL STATEMENTS

THEORY CONCEPTS:

1. if, if-else, if-else-if,nested if. - **20 Minutes**
2. switch-case statement. - **10 Minutes**

PROGRAM 1 : 15 MINUTES

An investor wants to decide whether to invest in a stock based on these conditions:
If the stock price increased by more than 10% in the last month, the investor will buy the stock.
If the stock price stayed within 5-10% of its previous value, the investor will hold the stock.
If the stock price dropped by more than 5%, the investor will sell the stock.
Write a program that takes the stock's price from a month ago and the current price and determines whether the investor should "Buy," "Hold," or "Sell."

Input Format:

The stock price a month ago (float).

The current stock price (float).

Output Format:

"Buy," "Hold," or "Sell."

Example:

Input:

Previous Price: 100

Current Price: 115

Output:

Buy

Solution:

```
import java.util.Scanner;

public class StockInvestmentDecision {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the stock price a month ago: ");
        float previousPrice = scanner.nextFloat();

        System.out.print("Enter the current stock price: ");
        float currentPrice = scanner.nextFloat();

        float percentageChange = ((currentPrice - previousPrice) / previousPrice) * 100;


        if (percentageChange > 10) {
            System.out.println("Buy");
        } else if (percentageChange >= -5 && percentageChange <= 10) {
            System.out.println("Hold");
        } else if (percentageChange < -5) {
            System.out.println("Sell");
        }

        scanner.close();
    }
}
```

<u>PROGRAM 2: 20 MINUTES</u>

In the town of Lakeland, the Water Supply Company is working to promote efficient water usage while ensuring fair pricing for its residents. To achieve this, the company has introduced a tiered water tariff system to encourage customers to be mindful of their consumption. The company bills its customers based on the amount of water they use in a month, following this structured tariff:

For the first 100 litres of water, customers are charged at a rate of ₹2 per liter.
For any water usage between 101 and 500 litres, the rate is ₹1.5 per liter.
For consumption beyond 500 litres, the rate drops to ₹1 per liter.
Objective: Your task is to develop a billing system that calculates the total water bill for a customer based on their monthly water usage.

Input format: The program will prompt the user to input the total number of litres consumed during the month.
Output Format: The system should compute the total bill according to the tariff structure and display it in a user-friendly format.

Explanation:
If a customer uses 550 litres of water in a month, their total bill will be calculated as follows:
For the first 100 litres: ₹2 per litre → ₹200
For the next 400 litres (101 to 500): ₹1.5 per litre → ₹600
For the remaining 50 litres (above 500): ₹1 per litre → ₹50
Total Bill: ₹200 + ₹600 + ₹50 = ₹900

Input:
Enter the number of litres consumed: 550

Output:
Total Water Bill: ₹900.0

Solution:

```
import java.util.Scanner;

public class WaterBillCalculator {
```

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

    System.out.println("Enter the number of litres consumed: ");
    int litres = scanner.nextInt();
    double bill = 0.0;

    if (litres > 100) {
        bill += 100 * 2.0;
        litres -= 100;
    } else {
        bill += litres * 2.0;
        litres = 0;
    }

    if (litres > 0) {
        if (litres > 400) {
            bill += 400 * 1.5;
            litres -= 400;
        } else {
            bill += litres * 1.5;
            litres = 0;
        }
    }

    if (litres > 0) {
        bill += litres * 1.0;
    }

    System.out.println("Total Water Bill: ₹" + bill);
    scanner.close();
}
```

}

PROGRAM 3: 20 MINUTES

Minimum Travel Time

The renowned book fair of the season "Publishers Federation Book Expo" is back, it promises to be bigger and better with a spread of about a million books on display. It is organised in a wide space this year on the topmost floor N of Hotel Grand Regency.
Williams, an ardent book lover visits the fair and wants to minimise the time it takes him to go from the N-th floor to ground floor. He can either take the elevator or the stairs.

The stairs are at an angle of 45 degrees and Williams's velocity is V1 m/s when taking the stairs down. The elevator on the other hand moves with a velocity V2 m/s. Whenever an elevator is called, it always starts from ground floor and goes to N-th floor where it collects Williams (collecting takes no time), it then makes its way down to the ground floor with Williams in it.

The elevator cross a total distance equal to N metres when going from N-th floor to ground floor or vice versa, while the length of the stairs is sqrt(2) * N because the stairs are at an angle 45 degrees. Williams has requested your help to decide whether he should use stairs or the elevator to minimise his travel time. Can you help him out?

Input Format

The first line of the input contains three space-separated integers N, V1, V2.

Output Format

Output a single line with string Elevator or Stairs, denoting the answer to the problem.

Sample Input	**Sample Output**
5 10 15	Elevator
Sample Input	**Sample Output**
2 10 14	Stairs

Solution:

```
import java.util.Scanner;
class Time
{
    public static void main(String args[])
    {
        Scanner ob = new Scanner(System.in);
        double n,v1,v2,t1,t2;
        n = ob.nextFloat();
        v1 = ob.nextFloat();
        v2 = ob.nextFloat();
        t1 = (Math.sqrt(2)*n)/v1;
        t2 = (2*n)/v2;
        if(t1>t2)
            System.out.println("Elevator");
        else
            System.out.println("Stairs");
    }
}
```

<u>PROGRAM 4: 25 MINUTES</u>

Co-Partners in Train

Tim and Bob are off to a famous Education Fair "Knowledge Forum 2017" at Uzhlanda. This time they have to travel without their guardians. Tim got very interested in the arrangement of seats inside the train coach.

The entire coach could be viewed as an arrangement of consecutive blocks of size 8.

BerthNumber Compartment

1-8 1

9-16 2

17-24 3

... and so on

Each of these size-8 blocks are further arranged as:

1LB, 2MB, 3UB, 4LB, 5MB, 6UB, 7SL, 8SU

9LB, 10MB, ...

......

....

Here LB denotes lower berth, MB middle berth and UB upper berth.

The following berths are called Co-Partners in Train:

3 UB 6 UB

2 MB 5 MB

1 LB 4 LB

7 SL 8 SU

and the pattern is repeated for every set of 8 berths.

Tim and Bob are playing this game of finding the co-partner in train of each berth. Write a program to do the same.

Input Format

The input consists of an integer N, which corresponds to the berth number whose neighbour is to be found out.

Output Format

The output is to display the berth of the neighbour of the corresponding seat.

Sample Input	**Sample Output**
1	4LB

Sample Input	**Sample Output**
5	2MB

Solution:

```
import java.io.*;
import java.util.*;
class copartnerstrain {
        public static void main(String [] args) {
                int a,count;
                Scanner sc = new Scanner(System.in);
                a = sc.nextInt();
                count=a%8;
                if(count==0){ a=a-1;}
                else if(count<4){a=a+3;}
                else if(count<7){a=a-3;}
                else if(count==7){a=a+1;}
                switch(count)
                {
                case 0:System.out.println(a+"SL");break;
                case 1:System.out.println(a+"LB");break;
                case 2:System.out.println(a+"MB");break;
                case 3:System.out.println(a+"UB");break;
                case 4:System.out.println(a+"LB");break;
                  case 5:System.out.println(a+"MB");break;
                case 6:System.out.println(a+"UB");break;
                case 7:System.out.println(a+"SU");break;
                }
        }
}
```

MODULE 4 – TECH 204 - CONTROL STATEMENTS

<u>**THEORY CONCEPTS:**</u>

1. for loop, while loop, do-while loop - **20 Minutes**
2. break and continue statements - **5 Minutes**
3. Nested loops - **20 Minutes**

PROGRAM 1: 25 MINUTES

Kaprekar number

A Kaprekar number is a number whose square when divided into two parts and such that the sum of parts is equal to the original number and none of the parts has value 0.

Input : n = 45

Output : Yes

Explanation : 45*45 = 2025 and 20 + 25 is 45

Input : n = 13

Output : No

Explanation : 13^2 = 169. Neither 16 + 9 nor 1 + 69 is equal to 13

Input : n = 297

Output : Yes

Explanation: 297^2 = 88209 and 88 + 209 is 297

Input : n = 10

Output : No

Explanation: 102 = 100. It is not a Kaprekar number even if sum of 100 + 0 is 100. This is because of the condition that none of the parts should have value 0.

Solutions:

```
public class CapricornNumber
{
    public static void main(String[] args)
    {
    Scanner scanner = new Scanner(System.in);
System.out.print("Enter a number=");
    int n = scanner.nextInt();
 boolean isCapricorn = false;
    int square = n * n;
    int temp = square;
    int contDigits = 0;
    while (temp > 0)
    {
    contDigits++;
    temp /= 10;
    }
    for (int i = 1; i < contDigits; i++)
    {
    int divisor = (int) Math.pow(10, i);
    int quotient = square / divisor;
    int remainder = square % divisor;
    if (quotient + remainder == n)
    {
        isCapricorn = true;
    }
    }
    if (isCapricorn)
    {
    System.out.println("Capricorn/Kaprekar number");
    } else
    {
    System.out.println("Not Capricorn/Kaprekar number");
    }
    }
```

```
}
```

PROGRAM 2: 25 MINUTES

Super Quiz Bee is a famous quiz Competition that tests students on a wide variety of academic subjects. This week's competition was a Team event and students who register for the event will be given a unique registration code N. The participants are teamed into 10 teams and the team to which a participant is assigned depends on the absolute difference between the first and last digit in the registration code.

The event organisers wanted your help in writing an automated program that will ease their job of assigning teams to the participants. If the registration number given is less than 10, then the program should display "Invalid Input".

Input Format

The only line of input contains an integer N.

Output Format

Output the absolute difference between the first and last digit of N**.**

Sample Input	Sample Output
345	2

Sample Input	Sample Output
9	Invalid Input

Solution:

```
import java.io.*;
import java.util.*;
class Teamevent {
        public static void main(String [] args) {
                int num,n,i,l;
                Scanner sc = new Scanner(System.in);
                num = sc.nextInt();
                if(num>=10)
                {
                n=num%10;
                l=num;
                for(i=0;num>0;i++)
                {
                num=num/10;
                if(num>0){l=num;}
                }
                if(l>=n) {
                     System.out.println(l-n);
                }
                else {
                     System.out.println(n-l);
                }
                }
                else
                {
                System.out.println("Invalid Input");
                }
        }
}
```

PROGRAM 3: 20 MINUTES

The IT giant "SoftCompInfo" has decided to transfer its message through the network using a new encryption technique. The company has decided to encrypt the data using the non-prime number concept. The message is in the form of a number and the sum of non-prime digits present in the message is used as the encryption key.
Write an algorithm to determine the encryption key.

Example
Input : 45673
Output:10

Explanation

The non-prime digits are 4 and 6. Hence the output is 4+6 = 10.

```
import java.util.*;
class Main
{
        public static int fun(int n)
        {
        int digit,i,sum=0,m;
        while(n!=0)
        {
        digit=n%10;
        n=n/10;

        for(i=2;i<digit;i++)
        {
                if(digit%i==0)
                {
                sum=sum+digit;
                break;
                }
        }
        }
         return sum;
        }
        public static void main(String args[])
        {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.print(fun(n));
        }
}
```

MODULE 5 – TECH 205 - CLASSES AND OBJECTS

THEORY CONCEPTS:

1. Introduction to Classes and Objects,Instance Variables and Methods,Object Creation and Instantiation - **25 Minutes**
2. Encapsulation=>Access modifiers (private, public, protected, default) - **15 Minutes**
3. Getters and setters methods to access and modify private data - **10 Minutes**

PROGRAM 1: 20 MINUTES

Employee Data Management using Encapsulation

Arjun works in an HR department and needs to store employee data securely. He wants to create a class Employee where name and salary are private instance variables. Provide getter and setter methods to access and modify the private data. Implement encapsulation to ensure the employee's salary can be updated only through the setter.

Input Format:

The first input is the employee name (string).

The second input is the salary of the employee (double).

The third input is the new salary (double).

Output Format:

Display the employee's name and salary after updating it.

Sample Input 1:

```
Rajesh
50000
55000
```

Sample Output 1:

```
Employee Name: Rajesh, Salary: 55000.0
```

Sample Input 2:

```
Suresh
```

```
60000
62000
```

Sample Output 2:

```
Employee Name: Suresh, Salary: 62000.0
```

Topic: Encapsulation, Getters, and Setters

Solution:

```
class Employee {
    // Private instance variables
    private String name;
    private double salary;

    // Constructor
    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    // Getter method for name
    public String getName() {
        return name;
    }

    // Setter method for name
    public void setName(String name) {
        this.name = name;
    }

    // Getter method for salary
    public double getSalary() {
        return salary;
    }
    // Setter method for salary
    public void setSalary(double salary) {
        this.salary = salary;
    }

    // Method to display employee details
    public void displayDetails() {
        System.out.println("Employee Name: " + name + ", Salary: " + salary);
    }
}

public class Main {
    public static void main(String[] args) {
        Employee emp1 = new Employee("Rajesh", 50000);
        emp1.setSalary(55000);  // Updating salary
        emp1.displayDetails();  // Output: Employee Name: Rajesh, Salary: 55000.0

        Employee emp2 = new Employee("Suresh", 60000);
        emp2.setSalary(62000);  // Updating salary
        emp2.displayDetails();  // Output: Employee Name: Suresh, Salary: 62000.0
    }
}
```

PROGRAM 2: 20 MINUTES

Madhav is creating a bank account management system where each BankAccount should have an accountHolderName and balance as private data members. Create getter and setter methods for the account holder’s name but prevent direct modification of the balance. Instead, provide methods for deposit and withdrawal operations to modify the balance.

Input Format:

- The first input is the account holder's name (string).

- The second input is the initial balance (double).
- The third input is an operation (deposit or withdraw).
- The fourth input is the amount to be deposited or withdrawn (double).

Output Format:

- Display the account holder's name and the updated balance after the transaction.

Sample Input 1:

```
Madhav
20000
deposit
5000
```

Sample Output 1:

```
Account Holder: Madhav, Updated Balance: 25000.0
```

Sample Input 2:

```
Suman
30000
withdraw
10000
```

Sample Output 2:

```
Account Holder: Suman, Updated Balance: 20000.0
```

Solution:

```
class BankAccount {
    // Private instance variables
    private String accountHolderName;
    private double balance;

    // Constructor
    public BankAccount(String accountHolderName, double balance) {
        this.accountHolderName = accountHolderName;
        this.balance = balance;
    }

    // Getter for account holder name
    public String getAccountHolderName() {
        return accountHolderName;
    }

    // Setter for account holder name
    public void setAccountHolderName(String accountHolderName) {
        this.accountHolderName = accountHolderName;
    }

    // Method to deposit money
    public void deposit(double amount) {
        balance += amount;
    }

    // Method to withdraw money
    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        } else {
            System.out.println("Insufficient balance.");
        }
    }

    // Method to display account details
    public void displayDetails() {
         System.out.println("Account Holder: " + accountHolderName + ", Updated Balance: " +
balance);
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        BankAccount account1 = new BankAccount("Madhav", 20000);
        account1.deposit(5000);  // Deposit money
            account1.displayDetails();   // Output: Account Holder: Madhav, Updated Balance:
25000.0

        BankAccount account2 = new BankAccount("Suman", 30000);
        account2.withdraw(10000);  // Withdraw money
            account2.displayDetails();   // Output: Account Holder: Suman, Updated Balance:
20000.0
    }
}
```

PROGRAM 3: 20 MINUTES

Create a class with the following attributes:

int rollno,

int mark1,

int mark2,

int mark3.

Create an array of objects for the above class.

Create the main class and in the main method calculate and print the following.

Total marks obtained by each student.

The highest mark in each subject with the roll number of the student who scored it.

The student who obtained the highest total mark.

Input Format

The first line of the input consists of the value of n.

The second lines consist of a integer (roll number, mark1, mark2, and mark3)

Output Format

First n lines print the total marks of each student.

The next 3 lines print the student's roll number and highest marks in each subject, separated by a space.

The last line prints the roll number of the student and the highest total marks scored.

Sample Input

```
5
1 98 85 76
2 85 74 65
3 85 96 75
4 52 65 79
5 52 75 65
```

Sample Output

```
259
224
256
196
```

Solution

```
import java.io.*;
import java.util.*;
class Student{
    public int  rno;
    public int  mark1;
    public int  mark2;
    public int  mark3;
    public void setRno(int r) {
        this.rno = r;
    }
    public void setMark1(int m1) {
        this.mark1 = m1;
    }
    public void setMark2(int m2) {
        this.mark2 = m2;
    }
    public void setMark3(int m3) {
        this.mark3 = m3;
    }
    public int getRno() {
        return rno;
    }
    public int getMark1() {
        return mark1;
    }
    public int getMark2() {
        return mark2;
    }
    public int getMark3() {
        return mark3;
    }
}
class main {
    public static void main(String [] args) {
        int n,i;
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt();
        Student [] s = new Student[n];
        for(i=0;i<n;i++) {
            s[i] = new Student();
            s[i].setRno(sc.nextInt());
            s[i].setMark1(sc.nextInt());
            s[i].setMark2(sc.nextInt());
            s[i].setMark3(sc.nextInt());
```

```
}
int [] totalmarks = new int[n];
for(i=0;i<n;i++) {
   totalmarks[i] = s[i].getMark1()+s[i].getMark2()+s[i].getMark3();
   System.out.println(totalmarks[i]);
}
int max1 = 0,max2 =0,max3=0;
int r1=0, r2=0, r3=0;
for(i=0;i<n;i++) {
   if(s[i].getMark1() > max1) {
      max1 = s[i].getMark1();
      r1 = s[i].getRno();
   }
}
for(i=0;i<n;i++) {
   if(s[i].getMark2() > max2) {
      max2 = s[i].getMark2();
      r2 = s[i].getRno();
   }
}
for(i=0;i<n;i++) {
   if(s[i].getMark3() > max3) {
      max3 = s[i].getMark3();
      r3 = s[i].getRno();
   }
}
System.out.println(r1+" "+max1);
System.out.println(r2+" "+max2);
System.out.println(r3+" "+max3);
int maxtotal = 0, r=0;
for(i=0;i<n;i++) {
   if(totalmarks[i] > maxtotal) {
```

```
                maxtotal = totalmarks[i];
                r = s[i].getRno();
            }
        }
        System.out.println(r+" "+maxtotal);
    }
}
```

MODULE 6 – TECH 206 - CONSTRUCTOR, STATIC

<u>THEORY CONCEPTS:</u>

1. Constructor types (default, parameterized) - 20 Minutes
2. Constructor Overloading - 5 Minutes
3. this keyword usage - 10 Minutes
4. Static Variables and Methods ,Static Blocks - 15 Minutes

<u>PROGRAM 1: 20 MINUTES</u>

Create a class Main with the following member variables int length int breadth int height int radius In Main class, get the details of the values of all variables and pass it to the shape class Create a class Shape with the following member variables int length int breadth int height int radius The values are passed from the Main class as parameters to the constructors in Shape class. In Shape class, create four constructors. The first constructor is the default constructor.Shape(){} The second constructor is for the cube, which receives length, breadth, and height as arguments. Shape(int length, int breadth, int height){constructor body} The third constructor is for the sphere, which receives radius as arguments. Shape(int radius){constructor body} The fourth constructor is for the cylinder, which receives radius and height as arguments. Shape(int radius, int height){Constructor body} Include three methods in Shape class namely public float calculateAreaOfCylinder() public float calculateAreaOfSphere() public Integer calculateAreaOfCube() to calculate the area of each shapes.

Sample Input and Output 1 :

```
1.Cube
2.Sphere
3.Cylinder
Enter the choice
1
length
```

10

breadth

20

height

30

Area of Cube is 6000

Sample Input and Output 2:

1.Cube

2.Sphere

3.Cylinder

Enter the choice

2

radius

10

Area of Sphere is 4186.67

Sample Input and Output 3:

1.Cube

2.Sphere

3.Cylinder

Enter the choice

3

radius

10

height

5

Area of Cylinders is 1570.0

- Case 1

Input (stdin)

1

10

```
20
30
```

Output (stdout)

```
1.Cube
2.Sphere
3.Cylinder
Enter the choice
length
breadth
height
Area of Cube is 6000
```

Code:

```
import java.util.Scanner;
class Shape
{
 static int length;
 static int breadth;
 static int height;
 static int radius;
 void Shape()
 {
 }
 public int calculateAreaOfCube()
 {
 return (length*breadth*height);
 }
 public float calculateAreaOfSphere()
 {
 return (float)(((float)(4.0/3.0))*(3.14)*(Math.pow(radius,3)));
 }
 public float calculateAreaOfCylinder()
 {
 return (float)((3.14)*(Math.pow(radius,2))*height);
 }
 void Shape(int length,int breadth,int height)
 {
 length=length;
 breadth=breadth;
 height=height;
 System.out.println("Area of Cube is "+calculateAreaOfCube());
 }
 void Shape(int radius)
 {
 radius=radius;
 System.out.printf("Area of Sphere is %.2f",calculateAreaOfSphere());
 }
void Shape(int radius,int height)
 {
 radius=radius;
 height=height;
 System.out.println("Area of Cylinders is "+calculateAreaOfCylinder());
 }

}
class Main extends Shape
{

 public static void main (String[] args) {
 Scanner s=new Scanner(System.in);
 Main obj=new Main();
 Shape ob=new Shape();
 System.out.println("1.Cube\n2.Sphere\n3.Cylinder");
 System.out.println("Enter the choice");
 int x=s.nextInt();
 if(x==1)
 {
 System.out.println("length");
 length=s.nextInt();
 System.out.println("breadth");
 breadth=s.nextInt();
 System.out.println("height");
 height=s.nextInt();
 ob.Shape(length,breadth,height);
 }
 else if(x==2)
 {
 System.out.println("radius");
 radius=s.nextInt();
 ob.Shape(radius);
 }
 else if(x==3)
 {
 System.out.println("radius");
 radius=s.nextInt();
 System.out.println("height");
 height=s.nextInt();
 ob.Shape(radius,height);
 }
 }
}
```

PROGRAM 2: 30 MINUTES

Write a program to get the employee id, name, and salary of N professors and print the details of the professors whose salary is greater or equal to 20000. Create a class Professor with the following public attributes,

id - int

name - string

salary - int

Include default constructor Professor() and parameterized constructor Professor (int id, string name, int salary) and a method

display() which prints the details.

In the main method create N objects for the class Professor and call the necessary functions.

Input Format

The first line consists of the total number of professors(N).

The next N lines consist of Employee id, Name, and Salary separated by space in each line.

Output Format

The output prints the details of the professors whose salary is greater or equal to 20000.

Refer to the sample input and output for formatting specifications

Sample Input

```
3
 1001 Akil 25000
 1002 Elon 30000
 1003 Musk 12000
```

Sample Output

```
1001 Akil 25000
 1002 Elon 30000
```

Sample Input

```
2
 100123 Jibran 100000
 100124 Rahman 200000
```

Sample Output

```
100123 Jibran 100000
100124 Rahman 200000
```

Code:

```
import java.util.Scanner;

class Professor {
    int id;
    String name;
    int salary;

    public Professor() {}

    public Professor(int id, String name, int salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }

    public void display() {
        System.out.println(id + " " + name + " " + salary);
    }
}

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

        int N = scanner.nextInt();
        scanner.nextLine();
        Professor[] professors = new Professor[N];
        for (int i = 0; i < N; i++) {
            int id = scanner.nextInt();
            String name = scanner.next();
            int salary = scanner.nextInt();
            professors[i] = new Professor(id, name, salary);
        }

        for (int i = 0; i < N; i++) {
            if (professors[i].salary >= 20000) {
                professors[i].display();
            }
        }
        scanner.close();
    }
}
```

ADDITIONAL QUESTIONS

PROGRAM 3: 25 MINUTES

Construct a class to convert an integer into a roman numeral.

Class name: intToromanConvertor

Method name: int_to_Roman

Hint: Find the below integer and its corresponding roman numeral.

(Example: integer 1000 = roman numeral M, integer 400 = roman CD)

integer= [1000, 900, 500, 400,100, 90, 50, 40,10, 9, 5, 4,1]

roman_equal = ["M", "CM", "D", "CD","C", "XC", "L", "XL","X", "IX", "V", "IV","I"]

Input Format

Input contains integers.

Output Format

Output displays equivalent roman numeral

Sample Input

5000

Sample Output

MMMMM

Sample Input

4

Sample Output

IV

Code:

```java
import java.util.Scanner;

class IntToRomanConvertor {

    public static String int_to_Roman(int num) {

        int[] integerValues = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
          String[] romanEqual = {"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V",
"IV", "I"};

        String result = "";

        for (int i = 0; i < integerValues.length; i++) {
            while (num >= integerValues[i]) {
                result += romanEqual[i];
                num -= integerValues[i];
            }
        }

        return result;
    }

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

        System.out.print("Enter an integer: ");
        int num = scanner.nextInt();

        String romanNumeral = int_to_Roman(num);
        System.out.println("Roman Numeral: " + romanNumeral);

        scanner.close();
    }
}
```

MODULE 7 – TECH 207 - INHERITANCE

THEORY CONCEPTS:

1. Introduction to Inheritance,Superclasses and Subclasses - **10 Minutes**
2. Types of inheritance - **30 Minutes**
3. Access Modifiers in Inheritance - **10 Minutes**
4. Constructor chaining using super -**10 Minutes**

PROGRAM 1: 15 MINUTES

A company maintains a database that has the details of all the employees.

There are two levels of employees where level 1 is the top management having a salary more than 1000 dollars and level 2 is the staff who are getting a salary less than 1000 dollars. Create a base class named "Employee" with empId and salary as attributes. Create a subclass, empLevel, that extends the employee and categorises the employee into various levels, and implements the following concept.

Input Format

The input should contain integer the employee ID and salary of the employee

Note: Employee id should be of integer type and salary float type.

Output Format

The output of the program must display the employee id, salary, and level of the employee, one below the other in the same order.

Sample Input

```
253 5.6
```

Sample Output

```
253
5.6
2
```

Answer:

```
 #include <iostream>
using namespace std;
class Employee {
public: int empId;       float salary;

public:
    Employee(int id, float sal)
    {
        empId = id;
        salary = sal;
    }
};

class empLevel : public Employee {
public:

    empLevel(int id, float sal) : Employee(id, sal) {}
    int determineLevel() {
        if (salary > 1000)
            return 1;
        else
            return 2;
    }
};

int main() {
    int id;
    float sal;
    cout << "Enter employee ID and salary: ";
    cin >> id >> sal;
    empLevel emp(id, sal);
    cout << emp.empId << endl;
    cout << emp.salary << endl;
    cout << emp.determineLevel() << endl;
    return 0;
}
```

PROGRAM 2: 20 MINUTES

Pooja is developing a system to track students and their education history. She creates a superclass Person with attributes name and age. The Student subclass inherits from Person and adds an attribute university. Another subclass, Graduate, inherits from Student and adds an attribute graduationYear. Implement these classes and print the details of the graduate.

Input Format:

The first line contains the name of the person (string).

The second line contains the age of the person (integer).

The third line contains the university the student attended (string).

The fourth line contains the graduationYear of the graduate (integer).

Output Format:

Display the details of the graduate in the following format: "Graduate <name>, Age: <age>, University: <university>, Year: <graduationYear>"

Sample Input 1:

```
Neeraj
24
IIT Delhi
2023
```

Sample Output 1:

```
Graduate Neeraj, Age: 24, University: IIT Delhi, Year: 2023
```

Sample Input 2:

```
Shreya
22
Anna University
2022
```

Sample Output 2:

```
Graduate Shreya, Age: 22, University: Anna University, Year: 2022
```

```
// Superclass Person
class Person {
    String name;
    int age;

    // Constructor for Person
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

// Subclass Student (inherits from Person)
class Student extends Person {
    String university;

    // Constructor for Student (uses super to chain Person constructor)
    public Student(String name, int age, String university) {
        super(name, age);
        this.university = university;
    }
}

// Subclass Graduate (inherits from Student)
class Graduate extends Student {
    int graduationYear;

    // Constructor for Graduate (uses super to chain Student constructor)
    public Graduate(String name, int age, String university, int graduationYear) {
        super(name, age, university);
        this.graduationYear = graduationYear;
    }

    // Method to display graduate details
    public void displayDetails() {
        System.out.println("Graduate " + name + ", Age: " + age + ", University: " + university
+ ", Year: " + graduationYear);
    }
}

// Main Class
public class Main {
    public static void main(String[] args) {
        Graduate grad1 = new Graduate("Neeraj", 24, "IIT Delhi", 2023);
        grad1.displayDetails();

        Graduate grad2 = new Graduate("Shreya", 22, "Anna University", 2022);
        grad2.displayDetails();
    }
}
```

PROGRAM 3: 25 MINUTES

Develop a program for a banking system for account management. Each account has the following attributes: AccountID, HolderName and Balance. Declare one constructor with three parameters that initialises the three attributes to some default values. Attributes must be validated.

AccountBalance must be greater than or equal to zero. If not, it is set to zero.
AccountID must be between 100 and 999. If not, set to -1 to indicate that it invalid.

Use the method setAccountBalance (…) to print account balance. Write one method: Credit to deposit money into the account. The method should return the new balance after the money deposit. Then create a class VIPAccount that inherits from class Account. The VIPAccount class overrides the method setAccountBalance (…) such that it prints the balance can be negative but no less than – 10000. The constructor of the VIPAccount class must call the constructor of the Account class.

Input Format

The first line of the input consists of the account id.

Next input is the account holder name.

The third input is the initial balance.

Fourth input is the amount to be credited.

The last input is a negative balance (Argument to setAccountBalance in overridden method).

Output Format

The first line of the output prints the account details.

The next line prints the new balance after the amount is credited.

Next output is the result of setAccountBalance (First base class method then derived class method).

Sample Input

```
120
Alice
48200
500
-15000
```

Sample Output

```
120 Alice 48200
48700
48700
The balance can be negative but no less than -10000
```

Solution:

```
import java.util.Scanner;

class Account {
    protected int AccountID;
    protected String HolderName;
    protected double Balance;

    // Constructor
    public Account(int id, String name, double balance) {
        // Validate and set AccountID
        if (id >= 100 && id <= 999) {
            AccountID = id;
        } else {
            AccountID = -1;
        }

        HolderName = name;
        Balance = (balance >= 0) ? balance : 0;
    }

    // Method to display account balance
    public void setAccountBalance() {
        System.out.println(AccountID + " " + HolderName + " " + Balance);
    }

    // Method to credit amount to the account
    public double Credit(double amount) {
        Balance += amount;
        return Balance;
    }
}
```

```
class VIPAccount extends Account {

    // Constructor
    public VIPAccount(int id, String name, double balance) {
        super(id, name, balance);
    }

    // Overridden method to set account balance for VIPAccount
    @Override
    public void setAccountBalance() {
        if (Balance < -10000) {
            Balance = -10000;
        }
        System.out.println(Balance);
    }
}

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

        // Read inputs from user
        System.out.print("Enter Account ID: ");
        int id = scanner.nextInt();

        scanner.nextLine(); // Consume the newline character left by nextInt()

        System.out.print("Enter Account Holder Name: ");
        String name = scanner.nextLine();

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

        System.out.print("Enter Amount to be Credited: ");
        double creditAmount = scanner.nextDouble();

        System.out.print("Enter Negative Balance: ");
        double negativeBalance = scanner.nextDouble();

        // Create Account and VIPAccount objects
        Account acc = new Account(id, name, initialBalance);
        VIPAccount vipAcc = new VIPAccount(id, name, initialBalance);

        // Display Account details and credit amount for the Account object
        System.out.println(acc.AccountID + " " + acc.HolderName + " " + acc.Balance);
        System.out.println(acc.Credit(creditAmount));
        acc.setAccountBalance(); // Call base class method

        // Display Account details and credit amount for the VIPAccount object
        System.out.println(vipAcc.Credit(creditAmount));
        vipAcc.setAccountBalance(); // Call overridden method
    }
}
```

MODULE 8 – TECH 208 – POLYMORPHISM

THEORY CONCEPTS:

1. Introduction to Polymorphism,Compile-time and runtime polymorphism - **5 Minutes**
2. Method overriding - **15 Minutes**
3. Method overloading - **15 minutes**
4. Final classes, methods, and variables - **20 Minutes**

PROGRAM 1: 25 MINUTES

Ravi wants to build a simple banking system that can calculate interest for different types of bank accounts. The types of accounts are:

- Savings account with an interest rate of 4%.
- FixedDeposit account with an interest rate of 6%.
- RecurringDeposit account with an interest rate of 5%.

To calculate the interest, Ravi creates a class Bank with overloaded methods, where the method signature is based on the account type and the amount deposited. The method overloading is done by having methods with the same name but different parameter types or numbers.

The method getInterestRate() will calculate the interest based on the account type and amount. The account type can be passed as a String, int, or double, depending on the overloaded method.

Requirements:

- Implement the overloaded methods of getInterestRate() based on account type.
- Method overloading should demonstrate using different parameter types:
- One method should accept String (for account type) and double (for amount).
- Another method should accept an int (for account type code) and double (for amount).
- Another method should accept two double parameters (for a fixed amount and **deposited amount).**

Input Format:

The first input will be the account type (either "Savings", "FixedDeposit", or "RecurringDeposit").
The second input will be the amount deposited (double).

Output Format:

Display the calculated interest based on the account type and the deposited amount.

Sample Input 1:

Savings

10000

Sample Output 1:

makefile

Interest: 400.0

Sample Input 2:

FixedDeposit

15000

Sample Output 2:

makefile

Interest: 900.0

Solution:

```
import java.util.Scanner;

class Bank {
    // Method to calculate interest for Savings Account
    public double getInterestRate(String accountType, double amount) {
        if (accountType.equalsIgnoreCase("Savings")) {
            return amount * 0.04;  // 4% interest for Savings
        }
        return 0;
    }

    // Overloaded method to calculate interest for Fixed Deposit Account
    public double getInterestRate(int accountTypeCode, double amount) {
        if (accountTypeCode == 1) {  // 1 represents Fixed Deposit
            return amount * 0.06;  // 6% interest for Fixed Deposit
        }
        return 0;
    }

    // Overloaded method to calculate interest for Recurring Deposit Account
    public double getInterestRate(double fixedAmount, double amount) {
        if (fixedAmount == 5000) {  // 5000 represents Recurring Deposit
            return amount * 0.05;  // 5% interest for Recurring Deposit
        }
        return 0;
    }
}

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

        // Taking inputs
        System.out.print("Enter Account Type (Savings/FixedDeposit/RecurringDeposit): ");
        String accountType = scanner.nextLine();

        System.out.print("Enter the Amount Deposited: ");
        double amount = scanner.nextDouble();

        // Creating an instance of the Bank class
        Bank bank = new Bank();
        // Using the overloaded methods based on the account type
        double interest = 0;
        if (accountType.equalsIgnoreCase("Savings")) {
            interest = bank.getInterestRate(accountType, amount);
        } else if (accountType.equalsIgnoreCase("FixedDeposit")) {
            interest = bank.getInterestRate(1, amount);  // 1 for Fixed Deposit
        } else if (accountType.equalsIgnoreCase("RecurringDeposit")) {
            interest = bank.getInterestRate(5000, amount);  // 5000 for Recurring Deposit
        }

        // Display the calculated interest
        System.out.println("Interest: " + interest);
    }
}
```

PROGRAM 2: 25 MINUTES

Sunil is building a system for an office where employees and managers get different bonuses. He creates a superclass Employee with a method calculateBonus() that returns a 10% bonus of the salary. The Manager subclass overrides the method to return a 15% bonus of the salary. Implement method overriding to display the bonus amount for both employee and manager.

Input Format:

- The first input is the role of the employee ("Employee" or "Manager").
- The second input is the salary of the employee (double).

Output Format:

- Display the bonus amount for the given employee.

Sample Input 1:

```
Employee
50000
```

Sample Output 1:

```
Bonus: 5000.0
```

Sample Input 2:

```
Manager
70000
```

Sample Output 2:

```
Bonus: 10500.0
```

Solution:

```
// Superclass Employee
class Employee {
    double salary;

    // Constructor
    public Employee(double salary) {
        this.salary = salary;
    }

    // Method to calculate bonus (10% for regular employees)
    public double calculateBonus() {
        return salary * 0.10;
    }
}

// Subclass Manager (overrides calculateBonus method)
class Manager extends Employee {

    public Manager(double salary) {
        super(salary);
    }
    public double calculateBonus() {
        return salary * 0.15;
    }
}

public class Main {
    public static void main(String[] args) {
        Employee emp1 = new Employee(50000);
        System.out.println("Bonus: " + emp1.calculateBonus());  // Output: 5000.0

        Manager mgr1 = new Manager(70000);
        System.out.println("Bonus: " + mgr1.calculateBonus());  // Output: 10500.0
    }
}
```

PROGRAM 3: 25 MINUTES

Bread Butter Jam Fuji likes bread so much. She eats bread daily. She eats bread with butter and jam or sometimes jam alone. Rarely does she eat bread alone. But she wants to know which gives her more energy. So she wants to calculate how much energy she gains from the bread that she eats. Help her in finding the amount of energy gained from given calories. 1 kcal = 4.1868 kJ Bread = 74 calories Butter = 102 calories Jam = 26 calories Write a program to calculate the energy and implement it using Method overloading concept. Print energy with three decimal places. Method Overloading is a feature that allows a class to have two or more methods having the same name, but their arguments type or the return type of the method or the number of arguments are different. Create two classes namely, Main and Calories. Get all the details in Main class. Include a method called,calculateCalories() which is overloaded three times. First calculateCalories() is to calculate the calorie when she eats only bread. Second calculateCalories() is to calculate the calorie when she eats bread and jam. Third, calculateCalories() is to calculate the calorie when she eats bread, butter and Jam. Include a method called returnCalories() that returns the calculated calories. Include a method called calculateEnergy() that calculates the energy and returns the energy value.

Sample input and output 1:

1.Bread only

```
2.Bread+Jam
3.Bread+Jam+Butter
Enter the choice
2
Enter the number of Slice of bread
2
Enter the number of teaspoon of Jam
4
1055.074 kJ of energy generated from 252.0 calories
```

Sample input and output 2:

```
1.Bread only
2.Bread+Jam
3.Bread+Jam+Butter
Enter the choice
1
Enter the number of Slice of bread
2
619.646 kJ of energy generated from 148.0 calories
```

Sample input and output 3:

```
1.Bread only
2.Bread+Jam
3.Bread+Jam+Butter
Enter the choice
3
Enter the number of Slice of bread
3
Enter the number of teaspoon of Jam
4
Enter the number of teaspoon of Butter
6
```

```
3927.219 kJ of energy generated from 938.0 calories
```

Case 1:

Input (stdin)

```
1
2
```

Output (stdout)

```
1.Bread only
2.Bread+Jam
3.Bread+Jam+Butter
Enter the choice
Enter the number of Slice of bread
619.646 kJ of energy generated from 148.0 calories
```

Code:

```
import java.util.*;
class Main
{
  public static void main(String args[])
  { int kcal;
    int bread;
     int jam;
     int butter;
    Calories c=new Calories();
    Scanner s=new Scanner(System.in);
 System.out.println("1.Bread only\n2.Bread+Jam\n3.Bread+Jam+Butter");
    System.out.println("Enter the choice");
    int choice=s.nextInt();
    switch(choice)
    {
      case 1:
```

```
      System.out.println("Enter the number of Slice of bread");
      bread=s.nextInt();
      kcal= c.calculateCalories(bread);
      c.calculateEnergy(kcal);
      break;
     case 2:
      System.out.println("Enter the number of Slice of bread");
      bread=s.nextInt();
      System.out.println("Enter the number of teaspoon of Jam");
      jam=s.nextInt();
      kcal=c.calculateCalories(bread,jam);
      c.calculateEnergy(kcal);
      break;
     case 3:
      System.out.println("Enter the number of Slice of bread");
      bread=s.nextInt();
      System.out.println("Enter the number of teaspoon of Jam");
      jam=s.nextInt();
      System.out.println("Enter the number of teaspoon of Butter");
      butter=s.nextInt();
      kcal= c.calculateCalories(bread,jam,butter);
      c.calculateEnergy(kcal);
      break;
  }
}}

class Calories
{
  int calories;
  int calculateCalories(int bread)
  {
   calories = (bread*74);
   return calories;
  }
   int calculateCalories(int bread,int jam)
  {
   calories=((74*bread)+(26*jam));


 return calories;
  }
 int calculateCalories(int bread,int jam,int butter)
  {
   calories=((74*bread)+(26*jam)+(102*butter));
   return calories;
  }
  void calculateEnergy( int calories)
  { float cal=(calories);
   // Float Kcal=(calories)/1000f;
   float energy=(4.1868f*calories);
   System.out.print(String.format("%.3f",energy));
   System.out.print(" kJ of energy generated from "+cal+" calories");
  }
}
```

MODULE 9 – TECH 209 - ABSTRACT CLASSES AND INTERFACES

THEORY CONCEPTS:

1. Introduction to Abstract Classes, Abstract methods and concrete methods in abstract classes - **15 Minutes**
2. Introduction to Interfaces,Implementing interfaces in classes, default and static methods in interfaces, inheritance with interfaces. - **20 Minutes**

PROGRAM 1: 30 MINUTES

Consider a Banking Scenario, there are many accounts, like Savings Account, Current Account, Demat Account and so on. We have a base Class Account which contains all the basic properties and methods of an Account. We do have some Maintenance Charges that apply to only some of the accounts. If you would like to enforce that the Savings Account & Current Account should have maintenance charges, then the simplest way is to ask your class to implement the interface. If you do not implement the method in the class, it would raise a compilation error. So, Java Interfaces essentially acts like a contract where it's given that the methods declared in the interface have to be implemented in the class. Let's code the above scenario. Create MaintainanceCharge interface with computeMaintainanceCharge method. Create a base Class Account with the fields - name, number, balance, and startDate. Create two subclasses CurrentAccount & SavingsAccount which extends Account and implements Maintenance Charge interface. In the Savings Account the maintenance amount will be 2mn+50. In checking Account, the maintenance amount will be mn+200. where m is the maintenance charge per year and n is the number of years. Follow the IO pattern for the input & output. Note: Maintenance charge Rs.50 for saving account and 100 for Current account.

Sample input and output 1:

```
1.Current Account
2.Savings Account
1
Name
```

```
SB
Account Number
12345
Account Balance
5000
Enter the Start Date(yyyy-mm-dd)
2013-04-22
Enter the Years
2
Maintenance Charge For Current Account 400.00
```

Sample input and output 2:

```
1.Current Account
2.Savings Account
2
Name
SB
Account Number
54321
Account Balance
3000
Enter the Start Date(yyyy-mm-dd)
2014-04-12
Enter the Years
5
Maintenance Charge For Savings Account 550.00
```

- Case 1

Input (stdin)

```
1
SB
12345
```

```
5000
2013-04-22
2
```

Output (stdout)

```
1.Current Account
2.Savings Account
Name
Account Number
Account Balance
Enter the Start Date(yyyy-mm-dd)
Enter the Years
Maintenance Charge For Current Account 400.00
```

Code:

```
import java.util.Scanner;
interface Maintainancecharge
{
   int compute();
}
 class Account
{

   String name;
    int number;
    int balance;
    String Startdate;
}
class Currentaccount extends Account
{
    int r;
     int compute(int n)
    {
        r=100*n+200;
        return r;
    }
}
class Savingsaccount extends Account
{
    int g;
     int  compute(int n)
    {
        g=(2*50*n)+50;
        return g;
    }
}
class Main
{
    public static void main(String args[])
    {
        int n;
        String na;
        int an;
        int ab;
        String sd;
        int y;
        Scanner sc=new Scanner(System.in);
        System.out.println("1.Current Account");
        System.out.println("2.Savings Account");
        n=sc.nextInt();
        System.out.println("Name");
         na=sc.nextLine();
          na=sc.nextLine();
        System.out.println("Account Number");
```

```
         an=sc.nextInt();
        System.out.println("Account Balance");
         ab=sc.nextInt();
        System.out.println("Enter the Start Date(yyyy-mm-dd)");
         sd=sc.nextLine();
          sd=sc.nextLine();
        System.out.println("Enter the Years");
         y=sc.nextInt();
         if(n==1)
         {
         Currentaccount obj1=new Currentaccount();
                              System.out.println("Maintainence   Charge   For   Current   Account
"+obj1.compute(y)+".00");
         }
         else
         {
         Savingsaccount obj2=new Savingsaccount();
                              System.out.println("Maintainence   Charge   For   savings   Account
"+obj2.compute(y)+".00");
         }
    }

}
```

PROGRAM 2: 30 MINUTES

Write a Program to calculate the current bill
Create a class currentBill with a virtual method double amount().
Create a Fan
Create a class Fan that extends currentBill with watts and hours as its public attributes and overrides the virtual function.

Create a class Light that extends currentBill with watts and hours as its public attributes and overrides the virtual function.

Create a class TV that extends currentBill with watts and hours as its public attributes and overrides the virtual function.

In the main method, prompt the user to enter the power rate of the appliance and the total hours used then create the necessary
objects and call the methods.

Input Format

The first line consists of the power rating of the fan and the total hours used separated by space.

The second line consists of the power rating of Light and the total hours used separated by space.

The third line consists of the power rating of the TV and the total hours used separated by space.

Output Format Sample

The output prints the bill amount. Refer to the sample input and output for formatting specifications.

Sample Input

```
40 123
55 200
33 400
```

Sample output:

```
43.68
```

Sample Input

```
60 300
54 360
30 720
```

Sample output:

```
88.56
```

Solution:

```
import java.util.Scanner;

// Base class with a virtual method
class CurrentBill {
    public double amount() {
        return 0;
    }
}

// Derived class Fan
class Fan extends CurrentBill {
    double watts;
    double hours;

    public Fan(double watts, double hours) {
        this.watts = watts;
        this.hours = hours;
    }

    @Override
    public double amount() {
        return watts * hours * 0.1; // Assume rate is 0.1 per watt-hour
    }
}

// Derived class Light
class Light extends CurrentBill {
    double watts;
    double hours;
    public Light(double watts, double hours) {
        this.watts = watts;
        this.hours = hours;
    }

    @Override
    public double amount() {
        return watts * hours * 0.11; // Assume rate is 0.11 per watt-hour
    }
}

// Derived class TV
class TV extends CurrentBill {
    double watts;
    double hours;

    public TV(double watts, double hours) {
        this.watts = watts;
        this.hours = hours;
    }

    @Override
    public double amount() {
        return watts * hours * 0.12; // Assume rate is 0.12 per watt-hour
    }
}

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

        // Get input for Fan
        int fanWatts = scanner.nextInt();
        int fanHours = scanner.nextInt();

        // Get input for Light
        int lightWatts = scanner.nextInt();
        int lightHours = scanner.nextInt();

        // Get input for TV
        int tvWatts = scanner.nextInt();
        int tvHours = scanner.nextInt();

        // Create objects for each appliance
        Fan fan = new Fan(fanWatts, fanHours);
        Light light = new Light(lightWatts, lightHours);
        TV tv = new TV(tvWatts, tvHours);

        // Calculate and print the bills
        System.out.printf("Fan Bill: %.2f\n", fan.amount());
        System.out.printf("Light Bill: %.2f\n", light.amount());
        System.out.printf("TV Bill: %.2f\n", tv.amount());

        scanner.close();
    }
}
```

PROGRAM 3:

Alphabetics Game: You have to enter four characters. For each uppercase letter, you will get 10 marks, and for each lowercase letter, you will get -5 marks.

Write a program to calculate the total score:

Create a base class with a method void game().

Define this method in the derived class to calculate the total score based on the letter cases.

Input Format:

The input consists of four characters separated by spaces.

Output Format:

The output prints the total score calculated based on the conditions specified.

Sample Input

```
A F K R
```

Sample Output

```
Score : 40
```

Sample Input

```
A b D f
```

Sample Output

```
Score : 10
```

Code:

```java
import java.util.Scanner;

// Base class with a virtual method
class Game {
    public void game() {
        // Base class method (does nothing)
    }
}

// Derived class to calculate the total score
class AlphabeticsGame extends Game {
    char[] letters;

    public AlphabeticsGame(char[] letters) {
        this.letters = letters;
    }
    @Override
    public void game() {
        int score = 0;

        // Loop through the array of characters and calculate the score
        for (char letter : letters) {
            if (Character.isUpperCase(letter)) {
                score += 10; // 10 points for uppercase
            } else if (Character.isLowerCase(letter)) {
                score -= 5;  // -5 points for lowercase
            }
        }

        // Print the total score
        System.out.println("Total Score: " + score);
    }
}

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

        // Take input for four characters separated by space
        System.out.println("Enter four characters separated by space:");
        String input = scanner.nextLine();

        // Split the input string into an array of characters
        String[] inputArray = input.split(" ");

        // Ensure that the input consists of exactly four characters
        if (inputArray.length != 4) {
            System.out.println("Invalid input. Please enter exactly four characters.");
            return;
        }

        // Convert the string array to a char array
        char[] letters = new char[4];
        for (int i = 0; i < 4; i++) {
            letters[i] = inputArray[i].charAt(0); // Only the first character of each input part
        }

        // Create an object of AlphabeticsGame and calculate the score
        AlphabeticsGame game = new AlphabeticsGame(letters);
        game.game();

        scanner.close();
    }
}
```

MODULE 10 – TECH 210 - ARRAY

THEORY CONCEPTS:

1. Introduction to Arrays - **10 Minutes**
2. Single-dimensional arrays, Declaring, initialising, and accessing arrays,Array operations - **20 Minutes**
3. Multi-dimensional arrays - **15 Minutes**

PROGRAM 1: 20 MINUTES

Ravi, Sita, Mohan, and Geeta are in a relay race, and their speeds are represented as an integer array. After the race, you want to determine a special score for each of them. This score is the product of the speeds of all participants except their own speed. The challenge is to calculate this without using division.

Problem Statement:

Given an integer array speeds representing the running speeds of Ravi, Sita, Mohan, and Geeta, return a new array score where score[i] is the product of all elements of speeds except speeds[i].

Example 1:

Input:

speeds = [1, 2, 3, 4]

Output:

score = [24, 12, 8, 6]

Example 2:

Input:

speeds = [-1,1,0,-3,3]

Output:

score =[0,0,9,0,0]

Solution:

```
import java.util.Scanner;

class Solution {
    public int[] productExceptSelf(int[] nums) {
        int n = nums.length;
        int[] ans = new int[n];

        for (int i = 0; i < n; i++) {
            int product = 1;
            for (int j = 0; j < n; j++) {
                if (i != j) {
                    product *= nums[j];
                }
            }

            ans[i] = product;
        }

        return ans;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of elements in the array:");
        int n = scanner.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt();
        }
        Solution sol = new Solution();
        int[] result = sol.productExceptSelf(nums);
        System.out.println("Output: " + java.util.Arrays.toString(result));

        scanner.close();
    }
}
```

PROGRAM 2: 20 MINUTES

Rahul, Priya, and Sanjay are planning to organise a secret event, and they have a budget for each item they plan to purchase. They have a list of possible costs, and they want to find three items that, together, perfectly match a specific total cost. If they can find these three items, they can proceed with their plan. If not, they'll need to change their strategy.

Help Rahul, Priya, and Sanjay by writing a program that finds a combination of three item prices from a given list that add up to the total budget.

Problem Statement:

Given an array of integers prices[] representing the cost of different items and an integer budget, your task is to find three distinct items (a triplet) whose sum equals the budget. If such a triplet exists, return it; otherwise, return that no such triplet is found.

Input Format:

An integer array prices[] where each element represents the price of an item.

An integer budget representing the total cost they want to achieve.

Output Format:

If such a triplet exists, print the three item prices.

If no such triplet exists, print a message indicating no valid combination.

Example:

Input:

prices = [12, 3, 4, 1, 6, 9]

budget = 24

Output:

Triplet found: 12, 3, 9

Example 2:

Input:
prices = [1, 2, 3, 4, 5]
budget = 12

Output:

No valid triplet found

Solution:

```
import java.util.Arrays;

public class TripletSum {
    public static void findTriplet(int[] prices, int budget) {
        int n = prices.length;
        Arrays.sort(prices);

        for (int i = 0; i < n - 2; i++) {
            int left = i + 1;
            int right = n - 1;

            while (left < right) {
                int sum = prices[i] + prices[left] + prices[right];

                if (sum == budget) {
                            System.out.println("Triplet found: " + prices[i] + ", " + prices[left] + ", " +
prices[right]);
                    return;
                }
                else if (sum < budget) {
                    left++;
                }

                else {
                    right--;
                }
            }
        }

        System.out.println("No valid triplet found");
    }

    public static void main(String[] args) {
        int[] prices1 = {12, 3, 4, 1, 6, 9};
        int budget1 = 24;
        System.out.println("Test case 1:");
        findTriplet(prices1, budget1);
        int[] prices2 = {1, 2, 3, 4, 5};
        int budget2 = 12;
        System.out.println("\nTest case 2:");
        findTriplet(prices2, budget2);
    }
}
```

<u>PROGRAM 3: 30 MINUTES</u>

Imagine a group of treasure hunters, Sarah, Tom, and Jake, who have discovered a mysterious ancient temple. Inside the temple, they find a grid-like chamber filled with precious jewels and artefacts arranged in a matrix format. The entrance is located at the top-left corner, and they want to collect the treasures in a specific order as they navigate through the temple.

To make their treasure hunt more exciting, they decide to collect the treasures in a **spiral order** starting from the entrance (top-left corner) and moving around the matrix in a circular manner until they gather all the items.

Help Sarah, Tom, and Jake by writing a program that performs a spiral traversal on the given matrix and returns the list of treasures in the order they should collect them.

Problem Statement:

Given a 2D matrix matrix[][] containing integers, your task is to return a list of integers representing the elements of the matrix collected in a spiral order

Input: mat[][] = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15,16]]

Output: [1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10]

Solution:

```
import java.util.Scanner;
public class SpiralTraversal {
    public static void spiralTraverse(int rows, int cols, int[][] matrix) {
        int top = 0, bottom = rows - 1;
        int left = 0, right = cols - 1;

        while (top <= bottom && left <= right) {
            // Traverse from left to right
            for (int i = left; i <= right; i++) {
                System.out.print(matrix[top][i] + " ");
            }
            top++;

            // Traverse downwards
            for (int i = top; i <= bottom; i++) {
                System.out.print(matrix[i][right] + " ");
            }
            right--;

            // Traverse from right to left
            if (top <= bottom) {
                for (int i = right; i >= left; i--) {
                    System.out.print(matrix[bottom][i] + " ");
                }
                bottom--;
            }

            // Traverse upwards
            if (left <= right) {
                for (int i = bottom; i >= top; i--) {
                    System.out.print(matrix[i][left] + " ");
                }
                left++;
            }
        }
    }
    public static void main(String[] args) {
        int rows = 5, cols = 4;
        int[][] matrix = {
            {1, 2, 3, 4},
            {5, 6, 7, 8},
            {9, 10, 11, 12},
            {13, 14, 15, 16},
            {17, 18, 19, 20}
        };
        System.out.println("Spiral Traversal of the Matrix:");
        spiralTraverse(rows, cols, matrix);
    }
}
```

MODULE 11 – TECH 211 - STRINGS

THEORY CONCEPTS:

1. Introduction to Strings, String Methods - **45 Minutes**
2. StringBuilder and StringBuffer - **30 Minutes**

PROGRAM 1: 25 MINUTES

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol	**Value**
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, 2 is written as II in Roman numerals, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

I can be placed before V (5) and X (10) to make 4 and 9.
X can be placed before L (50) and C (100) to make 40 and 90.
C can be placed before D (500) and M (1000) to make 400 and 900.
Given a roman numeral, convert it to an integer.

Example 1:

Input: s = "III"

Output: 3

Explanation: III = 3.

Example 2:

Input: s = "LVIII"

Output: 58

Explanation: L = 50, V= 5, III = 3.

Example 3:

Input: s = "MCMXCIV"

Output: 1994

Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

<u>Solution:</u>

```
public class RomanToInteger {

    public static int romanToInt(String s) {
        int total = 0;
        int prevValue = 0;
        for (int i = s.length() - 1; i >= 0; i--) {
            char currentChar = s.charAt(i);
            int currentValue = getValue(currentChar);
            if (currentValue < prevValue) {
                total -= currentValue;
            } else {
                total += currentValue;
            }

            prevValue = currentValue;
        }

        return total;
    }
    private static int getValue(char ch) {
        switch (ch) {
            case 'I': return 1;
            case 'V': return 5;
            case 'X': return 10;
            case 'L': return 50;
            case 'C': return 100;
            case 'D': return 500;
            case 'M': return 1000;
            default: return 0;
        }
    }

    public static void main(String[] args) {
        String s1 = "III";
        System.out.println("Input: " + s1 + " | Output: " + romanToInt(s1)); // Output: 3

        String s2 = "LVIII";
        System.out.println("Input: " + s2 + " | Output: " + romanToInt(s2)); // Output: 58

        String s3 = "MCMXCIV";
        System.out.println("Input: " + s3 + " | Output: " + romanToInt(s3)); // Output: 1994
    }
}
```

PROGRAM 2: 15 MINUTES

There is a robot starting at the position (0, 0), the origin, on a 2D plane. Given a sequence of its moves, judge if this robot ends up at (0, 0) after it completes its moves.

You are given a string move that represents the move sequence of the robot where moves[i] represents its ith move. Valid moves are 'R' (right), 'L' (left), 'U' (up), and 'D' (down).

Return true if the robot returns to the origin after it finishes all of its moves, or false otherwise.

Note: The way that the robot is "facing" is irrelevant. 'R' will always make the robot move to the right once, 'L' will always make it move left, etc. Also, assume that the magnitude of the robot's movement is the same for each move.

Example 1:

Input: moves = "UD"

Output: true

Explanation: The robot moves up once, and then down once. All moves have the same magnitude, so it ended up at the origin where it started. Therefore, we return true.

Example 2:

Input: moves = "LL"

Output: false

Explanation: The robot moves left twice. It ends up two "moves" to the left of the origin. We return false because it is not at the origin at the end of its moves.

Solution:

```
public class RomanToInteger {
    public static int romanToInt(String s) {
        int total = 0;
        int prevValue = 0;
        for (int i = s.length() - 1; i >= 0; i--) {
            char currentChar = s.charAt(i);
            int currentValue = getValue(currentChar);
            if (currentValue < prevValue) {
                total -= currentValue;
            } else {
                total += currentValue;
            }
            prevValue = currentValue;
        }

        return total;
    }

    private static int getValue(char ch) {
        switch (ch) {
            case 'I': return 1;
            case 'V': return 5;
            case 'X': return 10;
            case 'L': return 50;
            case 'C': return 100;
            case 'D': return 500;
            case 'M': return 1000;
            default: return 0;
        }
    }

    public static void main(String[] args) {
        String s1 = "III";
        System.out.println("Input: " + s1 + " | Output: " + romanToInt(s1)); // Output: 3

        String s2 = "LVIII";
        System.out.println("Input: " + s2 + " | Output: " + romanToInt(s2)); // Output: 58

        String s3 = "MCMXCIV";
        System.out.println("Input: " + s3 + " | Output: " + romanToInt(s3)); // Output: 1994
    }
}
```

MODULE 12 – TECH 212 - EXCEPTION HANDLING

THEORY CONCEPTS:

1. Introduction to Exception Handling - **10 Minutes**
2. Types of exceptions (checked vs. unchecked) - **10 Minutes**
3. try, catch, finally blocks - **15 Minutes**
4. Java exception keywords (throw, throws) - **15 Minutes**
5. Custom exceptions - **15 Minutes**

PROGRAM 1: 25 MINUTES

Create a class Student with attributes roll no, name, age and course. Initialize values through the parameterized constructor.

If the age of a student is not between 15 and 21 then generate a user-defined exception "AgeNotWithinRangeException". If the name contains numbers or special symbols raise exception

"NameNotValidException". Define the two exception classes to display the message as shown in sample output.

Example1

Input

```
100
Babu
20
MCA
```

Output

```
100 Babu 20 MCA
```

Explanation

Since the input values satisfy the constraint, print the given data.

Example2

Input

```
100
Babu
24
MCA
```

Output

```
Age is not between 15 and 21
100 Babu 24 MCA
```

Explanation

Since the age input value did not satisfy the constraint, print the appropriate exception.

Solution:

```java
import java.util.Scanner;

// Custom Exception for Age
class AgeNotWithinRangeException extends Exception {
    public AgeNotWithinRangeException(String message) {
        super(message);
    }
}

// Custom Exception for Name Validation
class NameNotValidException extends Exception {
    public NameNotValidException(String message) {
        super(message);
    }
}

// Student Class
class Student {
    private int rollNo;
    private String name;
    private int age;
    private String course;

    // Parameterized Constructor
    public Student(int rollNo, String name, int age, String course)
            throws AgeNotWithinRangeException, NameNotValidException {
        if (age < 15 || age > 21) {
            throw new AgeNotWithinRangeException("Age is not between 15 and 21");
        }

        if (!isValidName(name)) {
            throw new NameNotValidException("Name is not valid");
        }

        this.rollNo = rollNo;
        this.name = name;
        this.age = age;
        this.course = course;
    }

    // Method to check if the name is valid (without regex)
    private boolean isValidName(String name) {
        for (char c : name.toCharArray()) {
            // Check if character is not a letter or space
            if (!Character.isLetter(c) && !Character.isWhitespace(c)) {
                return false; // Invalid name
            }
        }
        return true; // Valid name
    }

    // Method to display student details
```

```
    public void display() {
        System.out.println(rollNo + " " + name + " " + age + " " + course);
    }
}

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

        // Input
        System.out.println("Enter roll number:");
        int rollNo = scanner.nextInt();
        scanner.nextLine(); // Consume the newline
        System.out.println("Enter name:");
        String name = scanner.nextLine();
        System.out.println("Enter age:");
        int age = scanner.nextInt();
        scanner.nextLine(); // Consume the newline
        System.out.println("Enter course:");
        String course = scanner.nextLine();

        try {
            // Create Student object
            Student student = new Student(rollNo, name, age, course);
            // If no exceptions are thrown, display student details
            student.display();
        } catch (AgeNotWithinRangeException | NameNotValidException e) {
            // Handle exceptions
            System.out.println(e.getMessage());
            System.out.println(rollNo + " " + name + " " + age + " " + course);
        } finally {
            scanner.close(); // Close the scanner
        }
    }
}
```

PROGRAM 2: 20 MINUTES

Imagine you're developing a banking application where users can enter the amount they wish to deposit. To enhance user experience and prevent errors, you need to ensure that the application gracefully handles invalid inputs.

Problem Statement: The application should prompt users to enter a deposit amount. If the user enters a valid integer, the application will proceed with the deposit. If the input is invalid (e.g., a string or special characters), the application should catch the exception and inform the user without crashing.

Requirements:

The application prompts the user for an input amount.

It attempts to convert the input string to an integer.

If the conversion fails due to a NumberFormatException, it should inform the user that the input is invalid and prompt them to try again.

The program should continue to request input until a valid integer is provided.

Example Interactions:

Valid Input:

Input: "500"

Output: "Deposit successfully: $500 has been added to your account."

Invalid Input:

Input: "abc"

Output: "Invalid input. Please enter a valid integer."

Another Invalid Input:

Input: "45.67"

Output: "Invalid input. Please enter a valid integer."

Solution:

```
import java.util.Scanner;

public class BankingApp {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.print("Enter the amount you wish to deposit: ");
            String userInput = scanner.nextLine();
            try {
                int depositAmount = Integer.parseInt(userInput);
                  System.out.println("Deposit successful: $" + depositAmount + " has been added to
your account.");
                break;
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter a valid integer.");
            }
        }
        scanner.close();
    }
}
```

MODULE 13 – TECH 213 - COLLECTIONS I

THEORY CONCEPTS:

1. Overview of Collections Framework - 15 Minutes
2. Lists - 30 Minutes
3. Set - 20 Minutes
4. Queue - 20 Minutes

PROGRAM 1: 20 MINUTES

Using Java Library ArrayList as a List Interface implementation, input N integers from standard input and add to the list only if they form an increasing
sequence.

1. Take a number, N > 0 as input
2. Accept N integers as input

a. Add the number to the list only if it forms an increasing sequence else ignore

3. Print the list

Input Format

Input number of elements, N > 0

Enter each integer on the next N lines

Output Format

List of integers in increasing sequence ignoring out of order elements

Sample Input

```
7
3
5
9
4
11
13
2
```

Sample Output

```
[3, 5, 9, 11, 13]
```

Code:

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

public class IncreasingSequenceList {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
         int N = scanner.nextInt();
        List<Integer> list = new ArrayList<>();
         int previousNumber = Integer.MIN_VALUE;
           for (int i = 0; i < N; i++) {
             int number = scanner.nextInt();
             if (number > previousNumber) {
                 list.add(number);
                 previousNumber = number;
             }
         }
         System.out.println(list);
         scanner.close();
    }
}
```

PROGRAM 2: 20 MINUTES

In computer science, a set is an abstract data type that can store certain values, without any particular order, and no repeated values(Wikipedia). is an example of a set, but is not a set. Today you will learn how to use sets in java by solving this problem.

You are given pairs of strings. Two pairs are identical if and . That also implies is not same as . After taking each pair as input, you need to print the number of unique pairs you currently have.

Input Format

In the first line, there will be an integer denoting the number of pairs. Each of the next lines will contain two strings separated by a single space.

Constraints:

Length of each string is at most and will consist of lower case letters only.

Output Format

Print lines. In the line, print the number of unique pairs you have after taking pair as input.

Sample Input

```
5
john tom
john mary
john tom
mary anna
mary anna
```

Sample Output

```
1
2
2
3
3
```

Explanation

After taking the first input, you have only one pair: (john,tom)

After taking the second input, you have two pairs: (john, tom) and (john, mary)

After taking the third input, you still have two unique pairs.

After taking the fourth input, you have three unique pairs: (john,tom), (john, mary) and (mary, anna)

After taking the fifth input, you still have three unique pairs.

Solution:

```
import java.util.*;

public class UniquePairs {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        scanner.nextLine();
        Set<String> uniquePairs = new HashSet<>();
        for (int i = 0; i < n; i++) {
            String input = scanner.nextLine();
            String[] pair = input.split(" ");
            Arrays.sort(pair);
            uniquePairs.add(pair[0] + " " + pair[1]);
            System.out.println(uniquePairs.size());
        }
        scanner.close();
    }
}
```

MODULE 14 – TECH 214 - COLLECTIONS II

THEORY CONCEPTS:

1. Map Interface - **15 Minutes**
2. Classes - **15 Minutes**
3. Comparators - **20 Minutes**

PROGRAM 1: 30 MINUTES

Harini is a content analyst at an online media company that tracks trending topics on social media. Her team receives large blocks of text data daily, filled with various user comments and hashtags. To quickly identify frequently mentioned words or topics, Harini is tasked with creating a tool that counts the occurrences of each word in a given text block.

How can Harini implement this Java program to efficiently count each word's occurrences in a given text block, using the Map interface and HashMap? Ensure the implementation is case-insensitive and ignores punctuation.

Sample Input:

```
Enter a string:

This is a test. This test is only a test.
```

Sample Output:

```
Word occurrences:
a: 2
test: 1
this: 2
only: 1
is: 2
test.: 2
```

Code:

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

class WordOccurrence {
    public static Map<String, Integer> countWordOccurrences(String input) {
        // Create a HashMap to store the words and their occurrences
        Map<String, Integer> wordCountMap = new HashMap<>();

        // Split the input string into words using whitespace as the delimiter
        String[] words = input.split("\\s+");

        // Iterate through each word
        for (String word : words) {
            // Convert word to lowercase to ensure case-insensitive counting
            word = word.toLowerCase();

            // Update the word count in the map
            wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1);
        }

        return wordCountMap;
    }

    public static void main(String[] args) {
        // Create a Scanner object for input
        Scanner scanner = new Scanner(System.in);

        // Prompt the user for a string
        System.out.println("Enter a string:");
        String input = scanner.nextLine();
        // Count word occurrences
        Map<String, Integer> wordOccurrences = countWordOccurrences(input);

        // Print the occurrences
        System.out.println("Word occurrences:");
        for (Map.Entry<String, Integer> entry : wordOccurrences.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }

        // Close the scanner
        scanner.close();
    }
}
```

<u>**PROGRAM 2: 30 MINUTES**</u>

Likith is an HR assistant at a startup, and he is tasked with organising employee data for a company-wide team-building event. To create a fair distribution of teams by age group, Likith decides to sort the list of employees by age. How can Likith write a Java program that uses a Comparator to sort employee records by age, with the ability to take dynamic input for names and ages?

Sample Input:

```
Enter the number of people:
3
Enter name for person 1:
Alice
Enter age for person 1:
30
Enter name for person 2:
Bob
Enter age for person 2:
25
```

```
Enter name for person 3:
Charlie
Enter age for person 3:
35
```

Sample Output:

```
People sorted by age:
Bob (25)
Alice (30)
Charlie (35)
```

Code:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.Scanner;

class Person
{
   String name;
   int age;

   Person(String name, int age)
   {
      this.name = name;
      this.age = age;
   }

   @Override
   public String toString() {
      return name + " (" + age + ")";
```

```
    }
}

public class ComparatorExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Person> people = new ArrayList<>();

        System.out.println("Enter the number of people:");
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        // Taking input for each person
        for (int i = 0; i < n; i++) {
            System.out.println("Enter name for person " + (i + 1) + ":");
            String name = scanner.nextLine();

            System.out.println("Enter age for person " + (i + 1) + ":");
            int age = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            // Add the person to the list
            people.add(new Person(name, age));
        }

        // Define a Comparator to sort by age
        Comparator<Person> ageComparator = new Comparator<Person>() {
            @Override
            public int compare(Person p1, Person p2) {
                return Integer.compare(p1.age, p2.age);
            }
        };
        // Alternatively, you can use lambda for Comparator
        // Comparator<Person> ageComparator = (p1, p2) -> Integer.compare(p1.age, p2.age);

        // Sort the people list by age
        Collections.sort(people, ageComparator);

        // Print the sorted list
        System.out.println("People sorted by age:");
        for (Person person : people) {
            System.out.println(person);
        }
        // Close the scanner
        scanner.close();
    }
}
```

MODULE 15 – TECH 215 - LAMBDA EXPRESSIONS, STREAMS

THEORY CONCEPTS:

1. Introduction to Lambda Expressions - **10 Minutes**
2. Functional Interfaces - **15 Minutes**
3. Method References, Types of method references: static methods, instance methods, constructor reference - **20 Minutes**
4. Stream API->Creating Streams, Intermediate Operations, Terminal Operations - **50 Minutes**

PROGRAM 1: 20 MINUTES

Given a string columnTitle that represents the column title as appears in an Excel sheet, return its corresponding column number.

For example:

A -> 1
B -> 2
C -> 3
...
Z -> 26
AA -> 27
AB -> 28
...

Example 1:

Input: columnTitle = "A"
Output: 1

Example 2:

Input: columnTitle = "AB"

Output: 28

Example 3:

Input: columnTitle = "ZY"

Output: 701

Constraints:

1 <= columnTitle.length <= 7

columnTitle consists only of uppercase English letters.

columnTitle is in the range ["A", "FXSHRXW"].

Solution:

```
public class ExcelColumnToNumber {
    public static void main(String[] args) {
        String columnTitle = "AB";

        // Lambda to convert columnTitle to column number
        int columnNumber = columnTitle.chars() // Stream of characters (int values)
                .map(c -> c - 'A' + 1)  // Convert character to its respective column number (A -> 1,
B -> 2, ...)
                     .reduce(0, (acc, num) -> acc * 26 + num);  // Accumulate the result (base-26
conversion)

        System.out.println("Column number: " + columnNumber);
    }
}
```

PROGRAM 2: 15 MINUTES

Consider a case where an e-commerce platform displays a list of product IDs on the homepage. Occasionally, due to database issues or filtering errors, some product IDs appear multiple times. The goal is to clean up the list by removing duplicates, while ensuring that the order in which the product IDs appear is maintained.

A programmer needs to write code that removes duplicates from a list of product IDs, while keeping the original order intact. The task involves using Java Streams to perform this operation efficiently.

Input Format

A list of integers representing product IDs, which may contain duplicates.

Output Format

A list of integers with duplicates removed, maintaining the order of their first occurrence.

Sample Input

[101, 102, 103, 102, 104, 101, 105]

Sample output:

[101, 102, 103, 104, 105]

Solution:

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

public class RemoveDuplicates {
    public static void main(String[] args) {
        List<Integer> productIds = Arrays.asList(101, 102, 103, 102, 104, 101, 105);

        List<Integer> result = productIds.stream()
                .distinct() // Removes duplicates
                .collect(Collectors.toList()); // Collect the result into a list

        System.out.println(result); // Print the result
    }
}
```

PROGRAM 3:

Einstein is trying to count the number of specific coloured balls from a bucket, all the balls are coloured from a to z. He is supposed to count the number of C coloured balls(i.e., One colour from a-z) using Stream API methods.

Input Format

First Line consists of String S(Bucket of Balls)

Second Line has an alphabet Character C(one character from a-z)

Output Format

Integer or Long value mentioning the count of C in S

Sample Input

```
Gokul is in God mode
o
```

Sample Output

```
3
```

Code:

```
class main {
    public static void main(String args[])
    {
 Scanner sc=new Scanner(System.in);
        String str = sc.nextLine();
        char c =sc.next().charAt(0);
        Long l= str.chars()
            .filter(val -> val == c)
            .count();
        System.out.println(l);
    }
 }
```