S.No: 1	Exp. Name: ***Using Int datatype***	**Date: 2023-10-11**

Aim:
Ramesh went to a general Store and picked two items of x and y prices.Write a java program to calculate the total amount for Ramesh to pay.

```
Sample Test Case:
Enter x value: 10
Enter y value: 20
Sum: 30
```

Source Code:

q1148/IntExample.java

```
package q1148;
import java.util.Scanner;
public class IntExample {
    public static void main(String[] args) {
    //write your code here
        Scanner a = new Scanner(System.in);
        System.out.print("Enter x value: ");
        int x = a.nextInt();
        System.out.print("Enter y value: ");
        int y = a.nextInt();
        System.out.println("Sum: " + (x + y));





    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter x value:
10
Enter y value:
20
Sum: 30
```

Test Case - 2

User Output

```
Enter x value:
12
Enter y value:
```

```
50
Sum: 62
```

Test Case - 3

User Output

```
Enter x value:
700
Enter y value:
654
Sum: 1354
```

Test Case - 4

User Output

```
Enter x value:
99
Enter y value:
100
Sum: 199
```

S.No: 2	Exp. Name: ***Using double datatype***	Date: 2023-10-11

Aim:
A contractor need to build a house for which he has to buy Y number of bricks of price X each.Help him to display total amount required to buy Y number of bricks.
Write a java program based on above Scenario.
Assumptions:
- Price X may be integer or double datatype
- Price Y may be integer or double datatype
- Total amount should be in double datatype

Sample Test Case:
```
Enter price: 10.56
Enter quantity: 50.5
Total amount: 533.28
```
Source Code:

q1149/doubleExample.java
```
package q1149;
import java.util.Scanner;
public class doubleExample {
    public static void main(String[] args) {
        //write your code here
        Scanner a = new Scanner(System.in);
        System.out.print("Enter price: ");
        double x = a.nextDouble();
        System.out.print("Enter quantity: ");
        double y = a.nextDouble();
        System.out.println("Total amount: " + (x * y));




    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output
```
Enter price:
100
Enter quantity:
20
Total amount: 2000.0
```

Test Case - 2

User Output

```
Enter price:
10.56
Enter quantity:
50.5
Total amount: 533.28
```

Test Case - 3

User Output

```
Enter price:
87.5
Enter quantity:
65
Total amount: 5687.5
```

Test Case - 4

User Output

```
Enter price:
83.56
Enter quantity:
200.3
Total amount: 16737.068000000003
```

S.No: 3	Exp. Name: ***Using String datatype***	Date: 2023-10-11

Aim:

A Class contains 60 Students. Mr. Vinod,class teacher,wants to send a message to the students congratulating them for clearing the final tests.
But Vinod doesn't have time to send messages to each student separately.
Write a program which takes the student name as input and displays Student's name along with the Vinod's message.

Assumptions:

- Student name doesn't contain empty spaces.

Sample Test Case:

```
Enter Student name: Vinod
Hello, Vinod. Congratulations on passing your last exams.
```

Source Code:

q1152/StringExample.java

```
package q1152;
import java.util.Scanner;
public class StringExample {
	public static void main(String[] args) {
		//write your code here
		Scanner a = new Scanner(System.in);
		System.out.print("Enter Student name: ");
		String n = a.nextLine();
		System.out.println("Hello, " + n + ". Congratulations on passing your last 
exams.");


	}
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter Student name:
Shiva
Hello, Shiva. Congratulations on passing your last exams.
```

Test Case - 2

User Output

```
Enter Student name:
Vinod
Hello, Vinod. Congratulations on passing your last exams.
```

Test Case - 3

User Output

```
Enter Student name:
Josh_Putnam
Hello, Josh_Putnam. Congratulations on passing your last exams.
```

Test Case - 4

User Output

```
Enter Student name:
Divya
Hello, Divya. Congratulations on passing your last exams.
```

S.No: 4	Exp. Name: ***Basic programs on datatypes***	**Date: 2023-10-11**

Aim:

A student named Akash wants to find **sum** and **product** of two numbers. Every time Akash has to perform those tasks for different numbers.
Write a java program to help Akash which takes two numbers and prints sum and product of the given numbers.

Sample Test Case:

```
Enter the num1 value: 10
Enter the num2 value: 20
Sum of 10 and 20 = 30
Product of 10 and 20 = 200
```

Source Code:

q1144/SumAndProductOfInt.java

```
package q1144;
import java.util.Scanner;
public class SumAndProductOfInt
{
        public static void main (String[] args)
        {
                  //write your code here
                Scanner a = new Scanner(System.in);
                System.out.print("Enter the num1 value: ");
                int n1 = a.nextInt();
                System.out.print("Enter the num2 value: ");
                int n2 = a.nextInt();
                System.out.println("Sum of " + n1 + " and " + n2 + " = " + (n1 + n2));
                System.out.println("Product of " + n1 + " and " + n2 + " = " + (n1 * n2));

        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the num1 value:
10
Enter the num2 value:
20
Sum of 10 and 20 = 30
Product of 10 and 20 = 200
```

Test Case - 2

User Output

```
Enter the num1 value:
58
```

```
Enter the num2 value:
69
Sum of 58 and 69 = 127
Product of 58 and 69 = 4002
```

Test Case - 3

User Output

```
Enter the num1 value:
875
Enter the num2 value:
258
Sum of 875 and 258 = 1133
Product of 875 and 258 = 225750
```

Test Case - 4

User Output

```
Enter the num1 value:
235
Enter the num2 value:
548
Sum of 235 and 548 = 783
Product of 235 and 548 = 128780
```

S.No: 5	Exp. Name: ***Basic programs on datatypes***	Date: 2023-10-11

Aim:
Suresh is a civil engineer.He wants to find the perimeter of the plots which helps him to make blueprints of the land.
Write a java program to find the perimeter of the plots which helps suresh to execute his plans easily.

Sample Test Case:
```
Enter the length of the plot: 100
Enter the width of the plot: 50
Perimeter of the plot is: 300
```
Source Code:

q1153/RectanglePerimeter.java

```
package q1153;
import java.util.Scanner;
public class RectanglePerimeter {
    public static void main(String[] args) {
     //write your code here
        Scanner a = new Scanner(System.in);
        System.out.print("Enter the length of the plot: ");
        int l = a.nextInt();
        System.out.print("Enter the width of the plot: ");
        int w = a.nextInt();
        System.out.println("Perimeter of the plot is: " + (2 * (l + w)));









    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output
```
Enter the length of the plot:
100
Enter the width of the plot:
50
Perimeter of the plot is: 300
```

Test Case - 2

User Output

```
Enter the length of the plot:
58
Enter the width of the plot:
44
Perimeter of the plot is: 204
```

Test Case - 3

User Output

```
Enter the length of the plot:
99
Enter the width of the plot:
87
Perimeter of the plot is: 372
```

Test Case - 4

User Output

```
Enter the length of the plot:
20
Enter the width of the plot:
10
Perimeter of the plot is: 60
```

S.No: 6	Exp. Name: ***Basic programs on datatypes***	Date: 2023-10-11

Aim:

Raju is feverish. Although the thermometer only displays temperatures in Fahrenheit, his companion Ravi wants to check Raju's body temperature in Celsius.

Write a java program to help ravi which converts temperature from Fahrenheit to Celsius scale.

Use the formula:

```
    Celsius=(Fahrenheit - 32) * 5/9
```

Sample Test Case:

```
Enter temperature in Fahrenheit: 100.5
Temperature in Celsius: 38.055557
```

Source Code:

q1146/TemperatureConverter.java

```
package q1146;
import java.util.Scanner;
public class TemperatureConverter {
    public static void main(String[] args) {
        // write your code here
        Scanner a =new Scanner(System.in);
        System.out.print("Enter temperature in Fahrenheit: ");
        float f = a.nextFloat();
        System.out.println("Temperature in Celsius: " + ((f - 32) * 5/9));







    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter temperature in Fahrenheit:
100.5
Temperature in Celsius: 38.055557
```

Test Case - 2

User Output

```
Enter temperature in Fahrenheit:
95.2
Temperature in Celsius: 35.11111
```

Test Case - 3

User Output

```
Enter temperature in Fahrenheit:
99
Temperature in Celsius: 37.22222
```

Test Case - 4

User Output

```
Enter temperature in Fahrenheit:
96.58
Temperature in Celsius: 35.87778
```

S.No: 7	Exp. Name: ***Basic programs on datatypes***	Date: 2023-10-11

Aim:
Ganesh owns a paper plant and recently received a large order to make paper plates with various radii.Ganesh needs to compute the area of every plate with a different radius each time.
Create a Java programme that calculates the area of the circular plates using the radius to aid Ganesh.

Note: Take **pi**value as **3.14**

Sample Test Case:
```
Enter the radius: 20
Area: 1256.0
```
Source Code:

q1154/AreaOfCircle.java

```
package q1154;
import java.util.Scanner;
public class AreaOfCircle {
    public static void main(String[] args) {
    //write your code here
                Scanner a = new Scanner(System.in);
                System.out.print("Enter the radius: ");
                double r = a.nextDouble();
                System.out.println("Area: " + (3.14 *r*r));




    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output
```
Enter the radius:
20
Area: 1256.0
```

Test Case - 2

User Output
```
Enter the radius:
50
Area: 7850.0
```

Test Case - 3

User Output

```
Enter the radius:
5
Area: 78.5
```

Test Case - 4

User Output

```
Enter the radius:
6
Area: 113.03999999999999
```

S.No: 8	Exp. Name: ***Typecasting***	Date: 2023-10-11

Aim:
Write a java program to Convert **int** to following datatypes

- int to long
- long to float
- float to double
- Print all the converted values.

Sample Test Case:

```
Enter an Integer: 78
After widening values are:
Long value: 78
Float value: 78.0
Double value: 78.0
```

Source Code:

q1140/TypeConversion.java

```
package q1140;
import java.util.Scanner;
public class TypeConversion {
        public static void main(String[] args) {
       //write your code here
       Scanner a = new Scanner(System.in);
       System.out.print("Enter an Integer: ");
       int x = a.nextInt();
       System.out.println("After widening values are:");
       long y = x;
       System.out.println("Long value: " + y);
       float z = y;
       System.out.println("Float value: " + z);
       double w = z;
       System.out.println("Double value: " + w);

  }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter an Integer:
78
After widening values are:
Long value: 78
Float value: 78.0
Double value: 78.0
```

Test Case - 2

User Output

```
Enter an Integer:
-123
After widening values are:
Long value: -123
Float value: -123.0
Double value: -123.0
```

Test Case - 3

User Output

```
Enter an Integer:
-4
After widening values are:
Long value: -4
Float value: -4.0
Double value: -4.0
```

Test Case - 4

User Output

```
Enter an Integer:
265
After widening values are:
Long value: 265
Float value: 265.0
Double value: 265.0
```

S.No: 9	Exp. Name: ***Typecasting***	Date: 2023-10-11

Aim:
Imagine having a stock trading application that only shows double datatype stock prices.Users occasionally request estimated stock prices in the form of float, long, and int datatypes.
Write a java program which takes the stock price and converts the price to display approximately(Observe the Sample Test Case).

Sample Test Case:

```
Enter the number: 126.321
After narrowing values are:
Float value: 126.321
Long value: 126
Int value: 126
```

Source Code:

q1141/ExplicitTypeConversion.java

```
package q1141;
import java.util.Scanner;
public class ExplicitTypeConversion
{
  public static void main(String[] args)
  {
        //write your code here
        Scanner a = new Scanner(System.in);
        System.out.print("Enter the number: ");
        double x = a.nextDouble();
        System.out.println("After narrowing values are:");
        float y = (float)x;
        System.out.println("Float value: " + y);
        long z = (long)y;
        System.out.println("Long value: " + z);
        int w = (int)z;
        System.out.println("Int value: " + w);
  }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output

```
Enter the number:
126.321
After narrowing values are:
Float value: 126.321
Long value: 126
Int value: 126
```

Test Case - 2

User Output

```
Enter the number:
23.54
After narrowing values are:
Float value: 23.54
Long value: 23
Int value: 23
```

Test Case - 3

User Output

```
Enter the number:
89.546
After narrowing values are:
Float value: 89.546
Long value: 89
Int value: 89
```

Test Case - 4

User Output

```
Enter the number:
2.31
After narrowing values are:
Float value: 2.31
Long value: 2
Int value: 2
```

S.No: 10	Exp. Name: ***Typecasting***	Date: 2023-10-11

Aim:

You are working on a software project that involves handling user input for various calculations. The user enters numeric values as strings, and you need to convert these strings to decimals for the calculations.

Note: Use **Double.parseDouble(String)** to convert String to double.

Sample Test Case:

```
Enter a double value: 14.36
Double value entered: 14.36
```

Source Code:

q1157/TypeCasting.java

```
package q1157;
import java.util.Scanner;
public class TypeCasting{
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
                //write your code here
        System.out.print("Enter a double value: ");
        String d = scanner.nextLine();
        System.out.println("Double value entered: " + Double.parseDouble(d));









    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter a double value:
14.36
Double value entered: 14.36
```

Test Case - 2

User Output

```
Enter a double value:
32.65
Double value entered: 32.65
```

Test Case - 3

User Output

```
Enter a double value:
46.85
Double value entered: 46.85
```

Test Case - 4

User Output

```
Enter a double value:
1.254
Double value entered: 1.254
```

S.No: 11	Exp. Name: ***Typecasting***	Date: 2023-10-11

Aim:

You are developing software for an inventory management system in a retail store. Part of the system involves entering and processing product prices, which are represented as decimal values. To optimise storage and facilitate calculations, Write a program to convert these prices to integers, rounding them to the nearest cent.

Sample Test case:

```
Enter a double value: 12.35
Original Value before Casting: 12.35
After Type Casting to short: 12
After Type Casting to int: 12
```

Source Code:

q1159/NarrowingTypeCasting.java

```java
package q1159;
import java.util.Scanner;
public class NarrowingTypeCasting {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a double value: ");
        double inputDouble = scanner.nextDouble();

        // write your code here
        System.out.println("Original Value before Casting: " + inputDouble);
        short s = (short)inputDouble;
        System.out.println("After Type Casting to short: " + s);
        int i = (int)inputDouble;
        System.out.println("After Type Casting to int: " + i);





    }

}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter a double value: 
12.35
Original Value before Casting: 12.35
After Type Casting to short: 12
After Type Casting to int: 12
```

Test Case - 2

User Output

```
Enter a double value:
45.123
Original Value before Casting: 45.123
After Type Casting to short: 45
After Type Casting to int: 45
```

Test Case - 3

User Output

```
Enter a double value:
3.21
Original Value before Casting: 3.21
After Type Casting to short: 3
After Type Casting to int: 3
```

Test Case - 4

User Output

```
Enter a double value:
100.23
Original Value before Casting: 100.23
After Type Casting to short: 100
After Type Casting to int: 100
```

S.No: 12	Exp. Name: ***Typecasting***	Date: 2023-10-11

Aim:

You are working on a software project for an international language education program. The project involves creating a program that allows users to enter keyboard shortcuts for various language characters and symbols. These shortcuts are represented as **short** data types, and you need to convert them to **char** data types to display the corresponding characters.

Sample Test Case:

```
Enter a short value: 65
Original short value: 65
Converted char value: A
```

Source Code:

q1161/ShortToCharConversion.java

```
package q1161;
import java.util.Scanner;

public class ShortToCharConversion {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
                //write your code here
        System.out.print("Enter a short value: ");
        short s = scanner.nextShort();
        System.out.println("Original short value: " + s);
        char c = (char)s;
        System.out.println("Converted char value: " + c);









    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter a short value:
65
Original short value: 65
Converted char value: A
```

Test Case - 2

User Output

```
Enter a short value:
```

```
67
Original short value: 67
Converted char value: C
```

Test Case - 3

User Output

```
Enter a short value:
78
Original short value: 78
Converted char value: N
```

Test Case - 4

User Output

```
Enter a short value:
100
Original short value: 100
Converted char value: d
```

Aim:

A task is assigned to a student,in which the student has to categorise the metal blocks in different boxes based on their weight.He can weigh only two metal blocks, at a time.
Write a program which takes weights of two metal boxes and displays a list which includes all relations between the weights.(Observe the sample test case).

Sample Test Case:

```
Enter the weight of object1: 50.5
Enter the weight of object2: 30.6
Relation between object1 and object2:
object1 == object2 = false
object1 != object2 = true
object1 > object2 = true
object1 < object2 = false
object1 >= object2 = true
object1 <= object2 = false
```

Source Code:

q1143/RelationalOperations.java

```
package q1143;
import java.util.Scanner;
public class RelationalOperations{
public static void main(String []args){
         // write your code below
         Scanner a = new Scanner(System.in);
     System.out.print("Enter the weight of object1: ");
     float o1 = a.nextFloat();
     System.out.print("Enter the weight of object2: ");
     float o2 = a.nextFloat();
     System.out.println("Relation between object1 and object2:");
     System.out.println("object1 == object2 = " + (o1==o2));
     System.out.println("object1 != object2 = " + (o1!=o2));
     System.out.println("object1 > object2 = " + (o1>o2));
     System.out.println("object1 < object2 = " + (o1<o2));
     System.out.println("object1 >= object2 = " + (o1>=o2));
     System.out.println("object1 <= object2 = " + (o1<=o2));


        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the weight of object1:
50.5
Enter the weight of object2:
```

```
30.6
Relation between object1 and object2:
object1 == object2 = false
object1 != object2 = true
object1 > object2 = true
object1 < object2 = false
object1 >= object2 = true
object1 <= object2 = false
```

Test Case - 2

User Output

```
Enter the weight of object1:
20.896
Enter the weight of object2:
54.69
Relation between object1 and object2:
object1 == object2 = false
object1 != object2 = true
object1 > object2 = false
object1 < object2 = true
object1 >= object2 = false
object1 <= object2 = true
```

Test Case - 3

User Output

```
Enter the weight of object1:
55.5
Enter the weight of object2:
55.5
Relation between object1 and object2:
object1 == object2 = true
object1 != object2 = false
object1 > object2 = false
object1 < object2 = false
object1 >= object2 = true
object1 <= object2 = true
```

Test Case - 4

User Output

```
Enter the weight of object1:
452.369
Enter the weight of object2:
500.258
Relation between object1 and object2:
object1 == object2 = false
```

```
object1 != object2 = true
object1 > object2 = false
object1 < object2 = true
object1 >= object2 = false
object1 <= object2 = true
```

S.No: 14	Exp. Name: ***Operators***	Date: 2023-10-18

Aim:

Raju is a farmer. He wants to take a loan from a bank that offers a low Simple Interest Rate on a certain amount of capital. Raju needs help in calculating S.I with thegiven parameters p,r,t (prime, rate, time).

Write a java program based on above scenario to help Raju.

Sample Test Case:

```
Enter the principal amount: 850000.56
Enter the rate of interest (in percentage): 3.5
Enter the time (in years): 2.3
Simple Interest: 68425.04508
```

Source Code:

q1169/SimpleInterestCalculator.java

```
package q1169;
import java.util.Scanner;

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

    //write your code here
    Scanner a =new Scanner(System.in);
    System.out.print("Enter the principal amount: ");
    double p = a.nextDouble();
    System.out.print("Enter the rate of interest (in percentage): ");
    double r = a.nextDouble();
    System.out.print("Enter the time (in years): ");
    double t = a.nextDouble();
    double SI = (p*r*t)/100;
    System.out.println("Simple Interest: " + SI);
    a.close();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the principal amount:
10000
Enter the rate of interest (in percentage):
3
Enter the time (in years):
2
Simple Interest: 600.0
```

Test Case - 2

User Output

```
Enter the principal amount:
5000.658
Enter the rate of interest (in percentage):
5
Enter the time (in years):
5
Simple Interest: 1250.1645
```

Test Case - 3

User Output

```
Enter the principal amount:
560000
Enter the rate of interest (in percentage):
3
Enter the time (in years):
2
Simple Interest: 33600.0
```

Test Case - 4

User Output

```
Enter the principal amount:
850000.56
Enter the rate of interest (in percentage):
3.5
Enter the time (in years):
2.3
Simple Interest: 68425.04508
```

S.No: 15	Exp. Name: ***Write a Java program that prints all real solutions to the quadratic equation ax2+bx+c=0. Read in a, b, c and use the quadratic formula.***	Date: 2023-10-11

Aim:
Write a Java program that prints all real solutions to the quadratic equation $ax^2 + bx + c$. Read in a, b, c and use the quadratic formula.

Source Code:

realSolutions.java

```
import java.util.Scanner;
public class realSolutions {
	public static void main(String[] args) {
		Scanner x = new Scanner(System.in);
		System.out.print("Enter Value for a: ");
		double a = x.nextDouble();
		System.out.print("Enter Value for b: ");
		double b = x.nextDouble();
		System.out.print("Enter Value for c: ");
		double c = x.nextDouble();
		double d = b*b-4*a*c;
		if (d>=0) {
			double s1 = (-b+Math.pow(d,0.5))/(2*a);
			double s2 = (-b-Math.pow(d,0.5))/(2*a);
			System.out.println("Real Solutions are: " + s1 + "," + s2);
		}
		else {
			System.out.println("No Real Solutions");
		}
	}
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter Value for a:
1
Enter Value for b:
4
Enter Value for c:
6
No Real Solutions
```

Test Case - 2

User Output

```
Enter Value for a:
1
Enter Value for b:
6
Enter Value for c:
6
Real Solutions are: -1.2679491924311228,-4.732050807568877
```

Test Case - 3

User Output

```
Enter Value for a:
1
Enter Value for b:
-3
Enter Value for c:
2
Real Solutions are: 2.0,1.0
```

S.No: 16	Exp. Name: ***Fibonacci sequence***	Date: 2023-10-18

Aim:
Sarah is learning about the Fibonacci sequence in her maths class. Her teacher asked to start the sequence with two initial values, which are 0 & 1. Sarah is fascinated by this sequence and wants to explore it further.
Write a java program that generates the first **n** numbers in the Fibonacci sequence in Recursive(with using recursion) and Non-recursive way(without using recursion), starting with the initial values of 0 and 1, and then prints them out.

Sample Test Case:

```
Enter the range: 10
Fibonacci Sequence (Recursive):
0 1 1 2 3 5 8 13 21 34
Fibonacci Sequence (Non-Recursive):
0 1 1 2 3 5 8 13 21 34
```

Source Code:

```
q1172/Fibonacci.java
```

```
package q1172;
import java.util.Scanner;

public class Fibonacci {

    // Recursive function to calculate Fibonacci number
    public static int fibonacciRecursive(int n) {
       //write code with using recursion
       if (n<=1) {
                return n;
       }
       return fibonacciRecursive(n-1) + fibonacciRecursive(n-2);


    }

    // Non-recursive (iterative) function to calculate Fibonacci number
    public static int fibonacciIterative(int n) {
       //write code without using recursion
       int f[] = new int[n + 2];
       int i;
       f[0] = 0;
       f[1] = 1;
       for (i=2;i<=n;i++) {
        f[i] = f[i-1] + f[i-2];
       }
       return f[n];
    }

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

        System.out.print("Enter the range: ");
        int n = scanner.nextInt();

        System.out.println("Fibonacci Sequence (Recursive):");
        for (int i = 0; i < n; i++) {
            System.out.print(fibonacciRecursive(i) + " ");
        }

        System.out.println("\nFibonacci Sequence (Non-Recursive):");
        for (int i = 0; i < n; i++) {
            System.out.print(fibonacciIterative(i) + " ");
        }
        System.out.println();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the range:
```

```
5
Fibonacci Sequence (Recursive):
0 1 1 2 3
Fibonacci Sequence (Non-Recursive):
0 1 1 2 3
```

Test Case - 2

User Output

```
Enter the range:
10
Fibonacci Sequence (Recursive):
0 1 1 2 3 5 8 13 21 34
Fibonacci Sequence (Non-Recursive):
0 1 1 2 3 5 8 13 21 34
```

Test Case - 3

User Output

```
Enter the range:
4
Fibonacci Sequence (Recursive):
0 1 1 2
Fibonacci Sequence (Non-Recursive):
0 1 1 2
```

Test Case - 4

User Output

```
Enter the range:
9
Fibonacci Sequence (Recursive):
0 1 1 2 3 5 8 13 21
Fibonacci Sequence (Non-Recursive):
0 1 1 2 3 5 8 13 21
```

S.No: 17	Exp. Name: ***Conditional Statements-if, if-else, Nested-if, switch***	**Date: 2023-10-18**

Aim:

Ravi is a pension officer. Ravi's job is to collect the person's name,age and prepare a data which should be in a below format :

```
Enter your name: srividya
Enter your age: 28
Hello, srividya!
You are 28 years old.
You were born in the year 1995.
You are an adult.
```

Ravi wants to declares them as

- Children if they are under 18 years of age,
- Adults if they are between 18 to 65.
- Remaining as senior citizens.

Assume you are ravi. How would you solve the problem?

Note: "You are a minor."
"You are an adult."
"You are a senior citizen." use these sentences in print statements appropriately.

Source Code:

```
q1158/ScannerWithIfElse.java
```

```java
package q1158;
import java.util.Scanner;

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

        // Prompt the user for their name
        System.out.print("Enter your name: ");
        String name = scanner.nextLine();

        // Prompt the user for their age
        System.out.print("Enter your age: ");
        int age = scanner.nextInt();

        // Calculate the year of birth
        int birthYear = java.time.Year.now().getValue() - age;

        // Complete below statements
        System.out.println("Hello, " + name + "!");
        System.out.println("You are " + age + " years old.");
        System.out.println("You were born in the year " + birthYear + ".");

        // Add conditional logic based on age
        if (age<18) {
                System.out.println("You are a minor.");
        }
        else if (age>=18 & age<=65) {
                System.out.println("You are an adult.");
        }
        else {
                System.out.println("You are a senior citizen.");
        }




    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter your name:
balu
Enter your age:
76
Hello, balu!
```

```
You are 76 years old.
You were born in the year 1947.
You are a senior citizen.
```

Test Case - 2

User Output

```
Enter your name:
harsha
Enter your age:
34
Hello, harsha!
You are 34 years old.
You were born in the year 1989.
You are an adult.
```

Test Case - 3

User Output

```
Enter your name:
preethi
Enter your age:
13
Hello, preethi!
You are 13 years old.
You were born in the year 2010.
You are a minor.
```

Test Case - 4

User Output

```
Enter your name:
srividya
Enter your age:
28
Hello, srividya!
You are 28 years old.
You were born in the year 1995.
You are an adult.
```

S.No: 18	Exp. Name: ***Conditional Statements-if, if-else, Nested-if, switch***	**Date: 2023-10-18**

Aim:

Create a class InterviewProcess.Write a Java program that models an interview process with multiple rounds. The interview process consists of the following rounds: written test, group discussion, technical round, and HR round. The program should ask the candidate for their results at each stage and determine if they are eligible to proceed to the next round or if they should go home.

Sample Test Case:

```
Did you clear the written test? (yes/no): yes
You are eligible for the group discussion round
Did you pass the group discussion round? (yes/no): yes
You are eligible for the technical round
Did you pass the technical round? (yes/no): yes
Congrats! You are eligible for the HR round
```

Note: You must write "Sorry,You can go home" in else block.

Source Code:

q1162/InterviewProcess.java

```
package q1162;
import java.util.Scanner;

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

        // write the code here
        System.out.print("Did you clear the written test? (yes/no): ");
        String wtr = scanner.next();
        if (wtr.equalsIgnoreCase("yes")){
                System.out.println("You are eligible for the group discussion round");
                System.out.print("Did you pass the group discussion round? (yes/no): ");
                String gdr = scanner.next();
                if (gdr.equalsIgnoreCase("yes")){
                        System.out.println("You are eligible for the technical round");
                        System.out.print("Did you pass the technical round? (yes/no): ");
                        String tr = scanner.next();
                        if (tr.equalsIgnoreCase("yes")){
                                System.out.println("Congrats! You are eligible for the HR 
round");
                        }else{
                                System.out.println("Sorry,You can go home");
                        }
                }else{
                        System.out.println("Sorry,You can go home");
                }
        }else{
                System.out.println("Sorry,You can go home");
        }
        scanner.close();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Did you clear the written test? (yes/no):
yes
You are eligible for the group discussion round
Did you pass the group discussion round? (yes/no):
no
Sorry,You can go home
```

Test Case - 2

User Output

```
Did you clear the written test? (yes/no):
yes
You are eligible for the group discussion round
Did you pass the group discussion round? (yes/no):
yes
You are eligible for the technical round
Did you pass the technical round? (yes/no):
yes
Congrats! You are eligible for the HR round
```

S.No: 19	Exp. Name: ***Conditional Statements-if, if-else, Nested-if, switch***	**Date: 2023-10-18**

Aim:
In a class, the teacher conducted an exam for all the students. Based on that exam, their ranks were decided like this

- Marks greater than or equal to 90 and Marks less than or equal to 100 as 1st Rank,
- Marks greater than or equal to 75 and Marks less than 90 as 2nd Rank,
- Marks greater than or equal to 50 and Marks less than 75 as 3rd Rank,
- Marks greater than or equal to 35 and Marks less than 50 as just passed,
- Marks greater than or equal to 10 and Marks less than 35 as failed
- Any Other Marks as "Invalid credentials. Please enter valid marks."

Sample Test Case:

```
Enter your marks: 75
You got 2nd rank.
```

Note: "Congratulations! You got 1st rank."
"You got 2nd rank."
"You got 3rd rank."
"You just passed."
"You failed."
"Invalid credentials. Please enter valid marks." use these sentences in print statements

Source Code:

```
q1166/StudentResult.java
```

```
package q1166;
import java.util.Scanner;
public class StudentResult {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter your marks: ");
        // Prompt the user to enter marks
        int m = scanner.nextInt();
        // Determine the student's result based on marks
        if (m>=90 && m<=100){
                System.out.println("Congratulations! You got 1st rank.");
        }
        else if (m>=75 && m<90){
                System.out.println("You got 2nd rank.");
        }
        else if (m>=50 && m<75){
                System.out.println("You got 3rd rank.");
        }
        else if (m>=35 && m<50){
                System.out.println("You just passed.");
        }
        else if (m>=10 && m<35){
                System.out.println("You failed.");
        }
        else{
                System.out.println("Invalid credentials. Please enter valid marks.");
        }
 }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter your marks:
23
You failed.
```

Test Case - 2

User Output

```
Enter your marks:
8
Invalid credentials. Please enter valid marks.
```

Test Case - 3

User Output

```
Enter your marks:
95
Congratulations! You got 1st rank.
```

Test Case - 4

User Output

```
Enter your marks:
35
You just passed.
```

Test Case - 5

User Output

```
Enter your marks:
90
Congratulations! You got 1st rank.
```

S.No: 20	Exp. Name: ***Conditional Statements-if, if-else, Nested-if, switch***	Date: 2023-10-18

Aim:

Ravi possesses three geometric figures: a rectangle, a square, and a circle. He has the measurements for these shapes but lacks the knowledge to compute their respective areas. Create a program that assists Ravi in determining the areas of these different shapes.

Note: "Invalid choice. Please choose 1, 2, or 3." Use these sentences in default

Sample Test Case:

```
Choose a shape:
1. Rectangle
2. Square
3. Circle
Enter your choice (1/2/3): 1
Enter the length of the rectangle: 12
Enter the width of the rectangle: 13
The area of the rectangle is: 156.0
```

Source Code:

```
q1168/AreaCalculator.java
```

```
package q1168;
import java.util.Scanner;
public class AreaCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Choose a shape:");
        System.out.println("1. Rectangle");
        System.out.println("2. Square");
        System.out.println("3. Circle");
        System.out.print("Enter your choice (1/2/3): ");
                int choice = scanner.nextInt();

        switch (choice) {
            //write code here
            case 1:
                System.out.print("Enter the length of the rectangle: ");
                double l = scanner.nextDouble();
                System.out.print("Enter the width of the rectangle: ");
                double w = scanner.nextDouble();
                System.out.println("The area of the rectangle is: " + (l*w));
                break;
            case 2:
                System.out.print("Enter the side length of the square: ");
                double s = scanner.nextDouble();
                System.out.println("The area of the square is: " + (s*s));
                break;
            case 3:
                System.out.print("Enter the radius of the circle: ");
                double r = scanner.nextDouble();
                System.out.println("The area of the circle is: " + (Math.PI*r*r));
                break;
            default:
                System.out.println("Invalid choice. Please choose 1, 2, or 3.");
    }

    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Choose a shape:
1. Rectangle
2. Square
3. Circle
Enter your choice (1/2/3):
9
Invalid choice. Please choose 1, 2, or 3.
```

Test Case - 2

User Output

```
Choose a shape:
1. Rectangle
2. Square
3. Circle
Enter your choice (1/2/3):
1
Enter the length of the rectangle:
20.56
Enter the width of the rectangle:
10.365
The area of the rectangle is: 213.1044
```

Test Case - 3

User Output

```
Choose a shape:
1. Rectangle
2. Square
3. Circle
Enter your choice (1/2/3):
2
Enter the side length of the square:
30.569
The area of the square is: 934.463761
```

Test Case - 4

User Output

```
Choose a shape:
1. Rectangle
2. Square
3. Circle
Enter your choice (1/2/3):
3
Enter the radius of the circle:
50.64
The area of the circle is: 8056.33036015514
```

Test Case - 5

User Output

```
Choose a shape:
1. Rectangle
2. Square
3. Circle
```

```
Enter your choice (1/2/3):
1
Enter the length of the rectangle:
256.25
Enter the width of the rectangle:
365.2
The area of the rectangle is: 93582.5
```

S.No: 21	Exp. Name: ***Conditional Statements-if, if-else, Nested-if, switch***	**Date: 2023-10-18**

Aim:

Surya owns a digital watch and wishes to include a feature that provides information about the number of days in each month, ranging from 1 to 12. When he presses a single digit(month), the watch should display the corresponding number of days, such as 31. Assist Surya in implementing this update.

Note:
Use the below printing statements appropriately
"Enter a month number (1 to 12): "
"31 days in this month."
"30 days in this month."
"28 or 29 days in this month (leap year dependent)."
"Invalid month number. Please enter a number from 1 to 12."

Source Code:

q1170/DayscheckWithNumbers.java

```
package q1170;
import java.util.Scanner;

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

        System.out.print("Enter a month number (1 to 12): ");
        int month = scanner.nextInt();
//write a code here
                switch (month){
                        case 1: case 3: case 5: case 7: case 8: case 10: case 12:
                                System.out.println("31 days in this month.");break;
                        case 2:
                                System.out.println("28 or 29 days in this month (leap year
dependent).");break;
                        case 4: case 6: case 9: case 11:
                                System.out.println("30 days in this month.");break;
                        default:
                                System.out.println("Invalid month number. Please enter a
number from 1 to 12.");break;
                }


//for your reference observe the below comment(you can use one printing statement for
multiple cases)
 //case 4: case 6: case 9: case 11:
 //               System.out.println("30 days in this month.");

    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter a month number (1 to 12):
0
Invalid month number. Please enter a number from 1 to 12.
```

Test Case - 2

User Output

```
Enter a month number (1 to 12):
9
30 days in this month.
```

Test Case - 3

User Output

```
Enter a month number (1 to 12):
3
31 days in this month.
```

Test Case - 4

User Output

```
Enter a month number (1 to 12):
11
30 days in this month.
```

S.No: 22	Exp. Name: ***Loops-while, do-while, for***	**Date: 2023-10-18**

Aim:
Write a Java Program that prompts the user for an integer and then prints out all the prime numbers up to that Integer.

Sample Test Case:
```
Enter the end of the range: 10
2 3 5 7
```

Source Code:

q1167/PrimeNumber.java

```
package q1167;
import java.util.Scanner;
public class PrimeNumber {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the end of the range: ");
       // write your logic here
        int n = input.nextInt();
        for (int i=2;i<=n;i++){
                boolean isPrime = true;
                for (int j=2;j<=Math.sqrt(i);j++){
                        if (i%j==0){
                                isPrime = false;
                                break;
                        }
                }
                if (isPrime){
                        System.out.print(i+" ");
                }
        }
        input.close();
                        }
    }
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the end of the range:
10
2 3 5 7
```

Test Case - 2

User Output

```
Enter the end of the range:
20
2 3 5 7 11 13 17 19
```

Test Case - 3

User Output

```
Enter the end of the range:
100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Test Case - 4

User Output

```
Enter the end of the range:
30
2 3 5 7 11 13 17 19 23 29
```

S.No: 23	Exp. Name: ***Loops-while, do-while, for***	Date: 2023-10-18

Aim:

Write a program for the below requirements
*** calculate sum and product of the given number**
*** if sum equals to product print it is a spy number and print the given number in reverse order**
*** if sum and product not equal, print it is not a spy number.**

Note : use only while loop

Sample Test Case : 1

```
123
sum of the given number is: 6
product of the given number is: 6
it is a spy number
Given number in reverse order: 321
```

Sample Test Case : 2

```
456
sum of the given number is: 15
product of the given number is: 120
it is not a spy number
```

Source Code:

```
q1165/SpyNumber.java
```

```java
package q1165;
import java.util.Scanner;
class SpyNumber {
  public static void main(String[] args) {
   Scanner s1 = new Scanner(System.in);

   // write your code here
   int n = s1.nextInt();
   int sum = 0;
   int product = 1;
   int originalnum = n;
   while (n>0){
        int d = n%10;
        sum+=d;
        product*=d;
        n/=10;
   }
   System.out.println("sum of the given number is: "+sum);
   System.out.println("product of the given number is: "+product);
   if (sum==product){
        System.out.println("it is a spy number");
        System.out.print("Given number in reverse order: ");
        while (originalnum>0){
                int d = originalnum%10;
                System.out.print(d);
                originalnum/=10;
        }
        System.out.println();
   }else{
        System.out.println("it is not a spy number");
   }
   s1.close();
  }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
123
sum of the given number is: 6
product of the given number is: 6
it is a spy number
Given number in reverse order: 321
```

Test Case - 2

User Output

```
456
sum of the given number is: 15
product of the given number is: 120
```

```
it is not a spy number
```

Test Case - 3

User Output

```
1124
sum of the given number is: 8
product of the given number is: 8
it is a spy number
Given number in reverse order: 4211
```

Test Case - 4

User Output

```
1234
sum of the given number is: 10
product of the given number is: 24
it is not a spy number
```

S.No: 24	Exp. Name: ***Loops-while, do-while, for***	Date: 2023-10-18

Aim:
Rani needs to create a Java program that operates a printing machine to produce a pyramid-shaped pattern using alphabetic characters. This program should take a parameter 'n' to determine the number of rows in the pyramid and then navigate the printing machine accordingly to achieve the desired pyramid pattern.
Sample Test Case:

```
4
  A
 A B
 A B C
A B C D
```

Source Code:

q1171/AlphabetPyramid.java

```
package q1171;
import java.util.Scanner;
public class AlphabetPyramid
{
    public static void main(String[] args)
    {
        Scanner s1=new Scanner(System.in);



       //write ypur logic here
       int n = s1.nextInt();
       for (int i=0;i<n;i++){
        for (int j=0;j<n-i;j++){
                System.out.print(" ");
        }
        char ch='A';
        for (int j=0;j<=i;j++){
                System.out.print(ch+" ");
                ch++;
        }
        System.out.println();
       }
       s1.close();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
4
    A
   A B
  A B C
 A B C D
```

Test Case - 2

User Output

```
3
  A 
 A B 
A B C 
```

Test Case - 3

User Output

```
10
         A 
        A B 
       A B C 
      A B C D 
     A B C D E 
    A B C D E F 
   A B C D E F G 
  A B C D E F G H 
 A B C D E F G H I 
A B C D E F G H I J 
```

Test Case - 4

User Output

```
7
      A 
     A B 
    A B C 
   A B C D 
  A B C D E 
 A B C D E F 
A B C D E F G 
```

S.No: 25	Exp. Name: ***Arrays***	**Date: 2023-11-04**

Aim:
Write a Program to remove an Element from given Array

Problem Statement:

- The function should return the array excluding the given element

Function Rules:
Fill the missing logic in**function** `RemoveElementFromArray` with**return type**int[]and**parameters**as listed below:

- `int[] arr`
- `int element`

Source Code:

```
q1145/RemoveElementFromArray.java
```

```
package q1145;
import java.util.Arrays;
import java.util.Scanner;

public class RemoveElementFromArray {
 public static int[] removeElement(int[] arr, int element) {
        //write you code here
        int count=0;
        for (int value:arr){
                if (value!=element){
                        count++;
                }
        }

        int[] newArray=new int[count];
        int index=0;
        for (int value:arr){
                if (value!=element){
                        newArray[index]=value;
                        index++;
                }
        }
        return newArray;

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

        // Input the array size
        System.out.print("Enter the size of the array: ");
        int size = scanner.nextInt();

        int[] arr = new int[size];

        // Input array elements
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < size; i++) {
            arr[i] = scanner.nextInt();
        }

        // Input the element to be removed
        System.out.print("Enter the element to remove: ");
        int elementToRemove = scanner.nextInt();

        // Call the removeElement method to remove the element
        int[] modifiedArray = removeElement(arr, elementToRemove);

        // Display the modified array
        System.out.println("Array after removing " + elementToRemove + ":");
        for (int i = 0; i < modifiedArray.length; i++) {
            System.out.print(modifiedArray[i] + " ");
        }
        System.out.println();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the size of the array:
5
Enter the elements of the array:
2 3 4 5 6
Enter the element to remove:
5
Array after removing 5:
2 3 4 6
```

Test Case - 2

User Output

```
Enter the size of the array:
6
Enter the elements of the array:
3 4 5 6 7 77
Enter the element to remove:
1
Array after removing 1:
3 4 5 6 7 77
```

Test Case - 3

User Output

```
Enter the size of the array:
10
Enter the elements of the array:
1 2 3 4 5 6 7 8 9 11
Enter the element to remove:
9
Array after removing 9:
1 2 3 4 5 6 7 8 11
```

Test Case - 4

User Output

```
Enter the size of the array:
4
Enter the elements of the array:
5 6 89 63
Enter the element to remove:
5
Array after removing 5:
```

6 89 63

S.No: 26	Exp. Name: ***Arrays***	**Date: 2023-11-04**

Aim:

A group of people, including Deeksha, attended an exhibition. Deeksha, the organizer, is concerned about the safety of the attendees. To address this concern, she needs a program that can identify the two individuals in the group with the highest and lowest ages.
Write a program that will assist Deeksha in ensuring the well-being of the exhibition's participants.(Observe the Sample Test Case)

Sample Test Case:

```
Enter the number of people: 5
Enter the age of each person:
55 69 52 20 10
Maximum age: 69
Minimum age: 10
```

Source Code:

q1147/FindMinAndMax.java

```
package q1147;
import java.util.Scanner;
public class FindMinAndMax {
    public static void main(String[] args) {
        //write your code here
        Scanner a = new Scanner(System.in);
        System.out.print("Enter the number of people: ");
        int num = a.nextInt();
        int arr[] = new int[num];
        System.out.println("Enter the age of each person:");
        for (int i=0;i<num;i++){
                arr[i]=a.nextInt();
        }
        int max=arr[0];
        int min=arr[0];
        for (int i=0;i<num;i++){
                if (arr[i]<min){
                        min=arr[i];
                }
        }
        for (int j=0;j<num;j++){
                if (arr[j]>max){
                        max=arr[j];
                }
        }
        System.out.println("Maximum age: "+max);
        System.out.println("Minimum age: "+min);
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output

```
Enter the number of people:
5
Enter the age of each person:
55 69 52 20 10
Maximum age: 69
Minimum age: 10
```

Test Case - 2

User Output

```
Enter the number of people:
6
Enter the age of each person:
22 23 63 10 55 44
Maximum age: 63
Minimum age: 10
```

Test Case - 3

User Output

```
Enter the number of people:
2
Enter the age of each person:
56 45
Maximum age: 56
Minimum age: 45
```

Test Case - 4

User Output

```
Enter the number of people:
6
Enter the age of each person:
2 54 66 33 44 41
Maximum age: 66
Minimum age: 2
```

S.No: 27	Exp. Name: ***Arrays***	**Date: 2023-11-05**

Aim:
Write a java program to print **Leaders** from the given array

Explaination:

In the context of an array, "leaders" are elements that are greater than all the elements to their right in the array. In other words, a leader is an element that does not have any element to its right that is greater than itself. Leaders "stand out" in the sense that they are the largest elements towards the right end of the array.

Here's an example to illustrate the concept of leaders in an array:

Suppose you have an array:

arr = [16, 17, 4, 3, 5, 2]

In this example, the leaders in the array are 17, 5, and 2.

The task of finding leaders in an array involves iterating through the array from right to left and identifying elements that are greater than or equal to the maximum element seen so far to their right. If an element meets this condition, it is considered a leader and can be printed.
Print the leaders in the **reverse order**(right to left) of the array.

Sample Test Case:

```
Enter the number of elements in the array: 5
Enter the elements of the array:
78 85 6 45 55
Leaders in the array:
55 85
```

Source Code:

```
q1160/LeadersInArray.java
```

```
package q1160;
import java.util.Scanner;

public class LeadersInArray {
        public static void findLeaders(int[] arr) {
        //write your code here
        int n = arr.length;
        int maxRight = arr[n-1];
        System.out.println("Leaders in the array:");
        for (int i=n-1;i>=0;i--){
                if (arr[i]>=maxRight){
                        maxRight=arr[i];
                        System.out.print(maxRight+" ");
                }
        }
        System.out.println();

        }

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

        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();

        int[] arr = new int[n];

        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        findLeaders(arr);
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of elements in the array:
5
Enter the elements of the array:
78 85 6 45 55
Leaders in the array:
55 85
```

Test Case - 2

User Output

```
Enter the number of elements in the array:
```

```
6
Enter the elements of the array:
85 69 54 71 6 5
Leaders in the array:
5 6 71 85
```

Test Case - 3

User Output

```
Enter the number of elements in the array:
3
Enter the elements of the array:
2 8 6
Leaders in the array:
6 8
```

Test Case - 4

User Output

```
Enter the number of elements in the array:
4
Enter the elements of the array:
-54 -8 -75 -7
Leaders in the array:
-7
```

S.No: 28	Exp. Name: ***2D Arrays***	**Date: 2023-11-08**

Aim:

Suresh is an employee at a medical store, and he stores delicate and expensive medicines in a wooden box. This box is divided into **numRows** rows and **numCols** columns.Each medicine inside the box has a label indicating its price.

You are asked to create a program that calculates and displays the **total price** of all the medicines stored in this box.

Sample Test Case:

```
Enter the number of rows: 3
Enter the number of columns: 3
Enter the prices of each medicine:
856 698 547
693 574 693
569 658 547
Total amount: 5835
```

Source Code:

q1163/SumOf2DArray.java

```java
package q1163;
import java.util.Scanner;

public class SumOf2DArray {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of rows: ");
        int numRows = scanner.nextInt();
        System.out.print("Enter the number of columns: ");
        int numCols = scanner.nextInt();

        int[][] matrix = new int[numRows][numCols];

        System.out.println("Enter the prices of each medicine:");
        for (int i = 0; i < numRows; i++) {
            for (int j = 0; j < numCols; j++) {
                matrix[i][j] = scanner.nextInt();
            }
        }
        //write your code here
        int sum = 0;
        for (int i=0;i<numRows;i++){
                for (int j=0;j<numCols;j++){
                        sum = sum+matrix[i][j];
                }
        }
        System.out.println("Total amount: "+sum);


    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of rows:
3
Enter the number of columns:
3
Enter the prices of each medicine:
856 698 547
693 574 693
569 658 547
Total amount: 5835
```

Test Case - 2

User Output

```
Enter the number of rows:
3
Enter the number of columns:
2
Enter the prices of each medicine:
254 695
458 632
555 852
Total amount: 3446
```

Test Case - 3

User Output

```
Enter the number of rows:
3
Enter the number of columns:
1
Enter the prices of each medicine:
455
855
966
Total amount: 2276
```

Test Case - 4

User Output

```
Enter the number of rows:
2
Enter the number of columns:
```

3
Enter the prices of each medicine:
198 698 777
999 387 888
Total amount: 3947

S.No: 29	Exp. Name: ***2D Arrays***	**Date: 2023-11-05**

Aim:

Problem Statement:

The computer vision researchers are developing an advanced image enhancement algorithm for medical X-ray images. To achieve this, they need to transpose the pixel matrices of these X-ray images, which are stored in a database. Your task is to create a program that can efficiently transpose a given two-dimensional matrix, helping the researchers in their image enhancement process.

Sample Test Case:

```
Enter the number of rows: 3
Enter the number of columns: 2
Enter the elements of the matrix:
2 4
5 4
8 7
Transpose of the Matrix:
2 5 8
4 4 7
```

Source Code:

q1164/MatrixTranspose.java

```java
package q1164;
import java.util.Scanner;

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

        System.out.print("Enter the number of rows: ");
        int numRows = scanner.nextInt();
        System.out.print("Enter the number of columns: ");
        int numCols = scanner.nextInt();

        int[][] matrix = new int[numRows][numCols];

        System.out.println("Enter the elements of the matrix:");
        for (int i = 0; i < numRows; i++) {
            for (int j = 0; j < numCols; j++) {
                matrix[i][j] = scanner.nextInt();
            }
        }
        //write your code here
                System.out.println("Transpose of the Matrix:");
                for (int i=0;i<numCols;i++){
                        for (int j=0;j<numRows;j++){
                                System.out.print(matrix[j][i]+" ");
                        }
                        System.out.println();
                }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of rows:
3
Enter the number of columns:
3
Enter the elements of the matrix:
5 6 8
5 7 9
3 6 8
Transpose of the Matrix:
5 5 3
6 7 6
8 9 8
```

Test Case - 2

User Output

```
Enter the number of rows:
4
Enter the number of columns:
4
Enter the elements of the matrix:
5 6 9 8
6 3 9 8
5 7 2 1
4 4 2 7
Transpose of the Matrix:
5 6 5 4
6 3 7 4
9 9 2 2
8 8 1 7
```

Test Case - 3

User Output

```
Enter the number of rows:
2
Enter the number of columns:
2
Enter the elements of the matrix:
2 1
5 4
Transpose of the Matrix:
2 5
```

```
1 4
```

Test Case - 4

User Output

```
Enter the number of rows:
3
Enter the number of columns:
2
Enter the elements of the matrix:
2 4
5 4
8 7
Transpose of the Matrix:
2 5 8
4 4 7
```

S.No: 30	Exp. Name: ***Program to check the given String is Palindrome or not***	**Date: 2023-11-05**

Aim:
Create a class `PalindromeOrNot` with a **main** method. The method receives one command line argument. Check the given argument is palindrome or not.

For example:

```
Cmd Args : madam
The given string madam is a palindrome
```

```
Cmd Args : Godavari
The given string Godavari is not a palindrome
```

Note: Please don't change the package name.

Source Code:

q11184/PalindromeOrNot.java

```
package q11184;
public class PalindromeOrNot{
        public static void main(String[] args){
                String a = args[0];
                String b = "";
                int n = a.length();
                for (int i=n-1;i>=0;i--){
                        b = b+a.charAt(i);
                }
                if (a.equals(b)){
                        System.out.println("The given string "+a+" is a palindrome");
                }
                else{
                        System.out.println("The given string "+a+" is not a palindrome");
                }
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
The given string madam is a palindrome
```

Test Case - 2

User Output

```
The given string Godavari is not a palindrome
```

Test Case - 3

User Output

```
The given string malayalam is a palindrome
```

Test Case - 4

User Output

```
The given string 12345 is not a palindrome
```

S.No: 31	Exp. Name: ***Constructor Overloading***	Date: 2023-11-10

Aim:

Mr. Vikas is working for a school management system. The system maintains records of teachers. To efficiently manage these records, Vikas needs to display profiles for teachers. These profiles include the **name and their years of teaching experience.**

Implement a program that displays teachers name and their experience which helps Vikas to maintain records.

Note: if **Years of Experience** is not initialised it should automatically initialise with its default value **0.0**

Source Code:

q1332/Records.java

```
package q1332;

class Teacher {
        //Declare variables: name and Years of Experience
        String name;
        double yoe;
        // Constructor with name parameter
        public Teacher(String name){
                this(name, 0.0);
        }


        // Constructor with name and Years of Experience
        public Teacher(String name, double yoe){
                this.name = name;
                this.yoe = yoe;
        }

        void display() {
      System.out.println("Name: " + name + ", Years of Experience: " + yoe);
    }
}

public class Records {
    public static void main(String[] args) {
        String name = args[0];
        double yoe = 0.0;

        Teacher t = new Teacher(name, yoe);
    //if args.length is equal to 1 call the Constructor with 1 argument
        if (args.length==1){
                t.display();
        }

    //if args.length is equal to 2 call the Constructor with 2 arguments
        if (args.length==2){
                yoe = Double.parseDouble(args[1]);
                t = new Teacher(name, yoe);
                t.display();
        }


    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Name: Avinash, Years of Experience: 5.3
```

Test Case - 2

User Output

```
Name: Uma, Years of Experience: 0.0
```

Test Case - 3

User Output

```
Name: shivam, Years of Experience: 9.3
```

Test Case - 4

User Output

```
Name: Charles Savage, Years of Experience: 10.7
```

S.No: 32	Exp. Name: ***Method overloading***	**Date: 2023-11-08**

Aim:

Consider a situation where you have a list of exam scores for a class of students. You want to find the highest scores among the students to determine the top performers.
Write a program that takes **two or three** inputs from command-line arguments and displays **highest score**.

Sample Test Case:

For two Command line args:

```
Command line args 56 98
Maximum of two numbers: 98
```

For three Command line args:

```
Command line args 45 89 99
Maximum of three numbers: 99
```

Note: Scores are in int datatype.

Source Code:

```
q1329/MathOperations.java
```

```
package q1329;

public class MathOperations {
	// Method to find the maximum of two numbers
	public static void max(int a,int b){
		int max = a;
		if (a>b){
			max=a;
		}
		else{
			max=b;
		}
		System.out.println("Maximum of two numbers: "+max);
	}



	// Method to find the maximum of three numbers
	public static void max(int a,int b,int c){
		int max = a;
		if (a>b && a>c){
			max=a;
		}
		else if (b>a && b>c){
			max=b;
		}
		else{
			max=c;
		}
		System.out.println("Maximum of three numbers: "+max);
	}





	public static void main(String[] args) {

		int d = Integer.parseInt(args[0]);
		int e = Integer.parseInt(args[1]);

		MathOperations m = new MathOperations();
	//if args.length is equal to 2 call the method with two arguments
		if (args.length==2){
			m.max(d,e);
		}


	//else if args.length is equal to 3 call the method with three arguments
		if (args.length==3){
			int f = Integer.parseInt(args[2]);
			m.max(d,e,f);
		}
```

```
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Maximum of three numbers: 99
```

Test Case - 2

User Output

```
Maximum of two numbers: 98
```

Test Case - 3

User Output

```
Maximum of two numbers: 65
```

Test Case - 4

User Output

```
Maximum of three numbers: 5487
```

S.No: 33	Exp. Name: ***Method Overloading***	**Date: 2023-11-10**

Aim:
Imagine you are working in a retail company that sells products online. The company needs to calculate the **total price** of customer orders based on the items purchased. Different items have different prices, and some orders may contain multiple items.
You are tasked with creating a program to **calculate the total price** of customer orders.

Sample Test Case:
Sum of two products:

```
Command line args 5698.23 5872.36
Sum of two products: 11570.59
```

Sum of three products:

```
Command line args 5897.5 549.265 5698.3
Sum of three products: 12145.065
```

Source Code:

q1331/SumCalculator.java

```java
package q1331;

public class SumCalculator {
    // Method to find the sum of two products
    public static void Sum2(double a, double b){
        System.out.println("Sum of two products: "+(a+b));
    }



     // Method to find the sum of three products
    public static void Sum3(double a, double b, double c){
        System.out.println("Sum of three products: "+(a+b+c));
    }




     //main method
      public static void main(String[] args) {
        double x = Double.parseDouble(args[0]);
        double y = Double.parseDouble(args[1]);

        SumCalculator s = new SumCalculator();
        //if args.length is equal 2 Call the method to find the sum of two products
        if (args.length==2){
                s.Sum2(x,y);
        }
        //else if args.length is equal 3 Call the method to find the sum of three products
        if (args.length==3){
                double z = Double.parseDouble(args[2]);
                s.Sum3(x,y,z);
        }

    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Sum of two products: 11570.59
```

Test Case - 2

User Output

```
Sum of three products: 12145.065
```

Test Case - 3

User Output

```
Sum of three products: 679.0
```

Test Case - 4

User Output

```
Sum of two products: 859.0246000000001
```

S.No: 34	Exp. Name: ***Method Overloading***	**Date: 2023-11-10**

Aim:

Imagine you are working on a chat application for a tech startup. The chat application should allow users to send messages to each other, and you are responsible for implementing a feature that allows users to send both single messages and broadcast messages to multiple recipients.
Adapt the program to handle this scenario.

Sample Test Case:

Printing a message once:

```
Command line args Hello World!
Hello World!
```

Printing a message with a specified number of times:

```
Command line args Hello World! 3
Hello World!
Hello World!
Hello World!
```

Source Code:

```
q1330/MessagePrinter.java
```

```
package q1330;

public class MessagePrinter {
    // Method to print a message once(Method name: printMessage)
    public static void printMessage(String msg){
        System.out.println(msg);
    }




    // Method to print a message with a specified number of times(Method name: printMessage)
    public static void printMessage(String msg, int recipients){
        for (int i=0;i<recipients;i++){
                System.out.println(msg);
        }
    }



    //main method
    public static void main(String[] args) {

        String message = args[0];

        if (args.length == 1) {
            // Call the method to print the message once
            printMessage(message);
        } else if (args.length == 2) {
            int times = Integer.parseInt(args[1]);

            // Call the method to print the message multiple times
            printMessage(message, times);
        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Hello World!
```

Test Case - 2

User Output

```
Hello World!
Hello World!
Hello World!
```

Test Case - 3

User Output

```
Warning
Warning
Warning
Warning
Warning
```

Test Case - 4

User Output

```
Alert!
```

S.No: 35	Exp. Name: ***this keyword,Constructor Overloading***	**Date: 2023-11-10**

Aim:

Imagine you are working for a popular fast-food restaurant. The restaurant is known for its customizable orders, and you have been tasked with developing a command-line tool for creating and displaying orders.
Consider a scenario where a customer wants to order a "**Classic Cheeseburger**" with a price of **$6.99** and gave specific instructions like "**Extra ketchup and no pickles**"
Then the order should be displayed as:

```
Order:
Item: Classic Cheeseburger
Price: $6.99
Special Instructions: Extra ketchup and no pickles
```

Note: If price and instructions are not specified print default values(0.0 and null)

Source Code:

```
q1333/RestaurantOrderSystem.java
```

```
package q1333;

class Order {
    String name;
    double price;
    String specialInstructions;

    // Constructor for an order with an item(use this keyword)
    public Order(String name){
        this(name,0.0,null);
    }


    // Constructor for an order with an item and price(use this keyword)
   public Order(String name, double price){
        this(name, price, null);
   }


    // Constructor for an order with an item and special instructions(use this keyword)
    public Order(String name, String specialInstructions){
        this(name,0.0,specialInstructions);
    }

    // Constructor for an order with an item, price and specialInstructions(use this
keyword)
    public Order(String name, double price, String specialInstructions){
        this.name = name;
        this.price = price;
        this.specialInstructions = specialInstructions;
    }


    void displayOrder() {
        System.out.println("Order:");
        //print item,price and Special Instructions
        System.out.println("Item: "+name);
        System.out.println("Price: $"+price);
        System.out.println("Special Instructions: "+specialInstructions);


    }
}

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

        String itemName = args[0];
        Order order;

        switch (args.length) {
            //if args.length is 1 initialize order with new Order(itemName)
            case 1:
                order = new Order(itemName);
                break;
                double price = Double.parseDouble(args[1]);
                        order = new Order(itemName, price);
                        break;
            case 3:
                price = Double.parseDouble(args[1]);
                order = new Order(itemName, price, args[2]);
                break;

            default:
                System.out.println("Invalid number of arguments.");
                return;
        }
        order.displayOrder();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Order:
Item: Classic Cheeseburger
Price: $6.99
Special Instructions: Extra ketchup and no pickles
```

Test Case - 2

User Output

```
Order:
Item: Veggie Burger
Price: $5.49
Special Instructions: null
```

Test Case - 3

User Output

```
Order:
Item: Double Bacon Cheeseburger
Price: $9.99
Special Instructions: Add extra bacon
```

Test Case - 4

User Output

```
Order:
Item: Custom Burger
Price: $0.0
Special Instructions: null
```

S.No: 36	Exp. Name: ***this keyword***	**Date: 2023-11-10**

Aim:

Assume you work for the CodeTantra Company.New students sign up for the company every day to learn coding.You have the duty of extending a warm welcome to them.
Create a program which displays a greeting after receiving their names as input.

Sample Test Case:

```
Command line args Bob Johnson Alice Brown
Hello, Bob Johnson welcome to CodeTantra
Hello, Alice Brown welcome to CodeTantra
```

Source Code:

q1335/Greetings.java

```
package q1335;

public class Greetings {
            String name;
    public Greetings(String name) {
        //initialize name using this keyword inside the constructor
        this.name = name;
    }
        //create a method sayHello() to print the required output
        void sayHello(){
                System.out.println("Hello, "+name+" welcome to CodeTantra");
        }


        public static void main(String[] args){
        Greetings person1 = new Greetings(args[0]);
        Greetings person2 = new Greetings(args[1]);

        person1.sayHello();
        person2.sayHello();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Hello, Kevin Anderson welcome to CodeTantra
Hello, Olivia Taylor welcome to CodeTantra
```

Test Case - 2

User Output

```
Hello, Jessica Lee welcome to CodeTantra
Hello, William Harris welcome to CodeTantra
```

Test Case - 3

User Output

```
Hello, Samantha Clark welcome to CodeTantra
Hello, Thomas Evans welcome to CodeTantra
```

Test Case - 4

User Output

```
Hello, Jennifer Martinez welcome to CodeTantra
Hello, Richard Murphy welcome to CodeTantra
```

S.No: 37	Exp. Name: ***Write the code***	Date: 2023-11-10

Aim:

Vimesh works at a newspaper factory. He is responsible for creating the newspaper's headlines. Vimesh wants to take the very first sentence out of the story.
Using the information above as a basis, implement a program to assist Vimesh.

Note: If no period found print "**No period found. Unable to extract a summary.**"
Sample Test Case:
Command line args: I love to read books. Today is a beautiful day.

```
I love to read books.
```

Source Code:

q1337/ExtractFirstParagraph.java

```java
package q1337;

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

    // Get the input text from the command-line argument
        String inputText = args[0];

    // Extract the first paragraph (from the beginning to the first period)
        int periodIndex = inputText.indexOf('.');

        if (periodIndex!=-1){
                String firstParagraph = inputText.substring(0, periodIndex+1);
                System.out.println(firstParagraph);
        }
        else{
                System.out.println("No period found. Unable to extract a summary.");
        }







    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
I love to read books.
```

Test Case - 2

User Output

```
It was a perfect day at the beach.
```

Test Case - 3

User Output

```
The phone rang loudly.
```

Test Case - 4

User Output

```
No period found. Unable to extract a summary.
```

S.No: 38	Exp. Name: ***String Methods***	**Date: 2023-11-10**

Aim:

Raghav works in the field of cryptography.He receives encrypted messages from his informer every day.To understand the message, Raghav must break the sentence apart into distinct parts and substitute the nth word from the list of separated words with the key..This process takes most of Raghav's time.
Write a program which takes the encrypted message,nth word(**int**) from the list,key(**String**) and decrypts the message to aid Raghav.

Sample Test Case:

Command line args: Rock is down 0 Soilder

```
Original Message: Rock is down
Separeted Words:
Rock
is
down
Decrypted Message: Soilder is down
```

Source Code:

```
q1334/Decryption.java
```

```
package q1334;

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

    // 1.Print the Original message
        System.out.println("Original Message: "+args[0]);
    // 2. Split the message into words using space as the delimiter.
        String[] words = args[0].split(" ");
    // 3. Replace nth word(i.e.,args[1]) ,from the list of separated words, with args[2] in
the message.
        System.out.println("Separeted Words:");
        for (String word:words){
                System.out.println(word);
        }

        int n = Integer.parseInt(args[1]);
        if (n>=0 && n<words.length){
                words[n] = args[2];
        }
        else{
                System.out.println("Invalid index for the nth word");
                return;
        }

        StringBuilder decryptedMessage = new StringBuilder();
        for (String word:words){
                decryptedMessage.append(word).append(" ");
        }

        System.out.println("Decrypted Message: "+ decryptedMessage.toString().trim());

    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Original Message: Rock is down
Separeted Words:
Rock
is
down
Decrypted Message: Soilder is down
```

Test Case - 2

User Output

```
Original Message: There is a cute-dog
Separeted Words:
There
is
a
cute-dog
Decrypted Message: There is a incoming fighter jet
```

Test Case - 3

User Output

```
Original Message: Get ready for breakfast
Separeted Words:
Get
ready
for
breakfast
Decrypted Message: Get ready for war
```

Test Case - 4

User Output

```
Original Message: They carry good-will
Separeted Words:
They
carry
good-will
Decrypted Message: They carry heavy artillery
```

Aim:
Josh is a professional manga writer.He intends to add Easter eggs(Hidden surprises) in the story.To achieve that he needs to know the exact position of a certain word in a sentence.
Write a program to help Josh which takes the sentence and the word and displays the position.
(Use Scanner to take inputs)

Sample Test Case:

```
Enter the Sentence : Zror finds a sword called destroyer
Enter a word to find: sword
The word 'sword' is found at index 13 in the sentence.
```

Note:if the word is not found print:
The word 'sword' is not found in the sentence.

Source Code:

q1319/EasterEgg.java

```
package q1319;
import java.util.Scanner;

public class EasterEgg {
    public static void main(String[] args) {
     //write your code here
     Scanner a =new Scanner(System.in);
     System.out.print("Enter the Sentence : ");
     String s = a.nextLine();
     System.out.print("Enter a word to find: ");
     String t = a.nextLine();
     if (s.contains(t)){
        char[] charArray = t.toCharArray();
        System.out.println("The word '"+t+"' is found at index "+s.indexOf(t,0)+" in the
sentence.");
     }
     else{
        System.out.println("The word '"+t+"' is not found in the sentence.");
     }









    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the Sentence :
Zror finds a sword called destroyer
Enter a word to find:
sword
The word 'sword' is found at index 13 in the sentence.
```

Test Case - 2

User Output

```
Enter the Sentence :
Find the book ASAP
Enter a word to find:
the
The word 'the' is found at index 5 in the sentence.
```

Test Case - 3

User Output

```
Enter the Sentence :
No time to waste
Enter a word to find:
yes
The word 'yes' is not found in the sentence.
```

Test Case - 4

User Output

```
Enter the Sentence :
Eat the soup
Enter a word to find:
Eat
The word 'Eat' is found at index 0 in the sentence.
```

S.No: 40	Exp. Name: ***String Methods***	**Date: 2023-11-10**

Aim:
Uma wants to make her own dictionary.The words in a dictionary must be in lexical order.She also wants to know the length of the words.
Write a java program to help Uma which takes two words and displays the following details:
(Use Scanner to take inputs)
Note: If both words are same print "word1 and word2 are equal"
Sample Test Case:

```
Enter the first word: Code
Enter the second word: Tantra
Length of word1: 4
Length of word2: 6
Code comes before Tantra
```

Source Code:

q1244/Dictionary.java

```
package q1244;
import java.util.Scanner;

public class Dictionary {
    public static void main(String[] args) {
        //write your code here
        Scanner a = new Scanner(System.in);
        System.out.print("Enter the first word: ");
        String x = a.nextLine();
        System.out.print("Enter the second word: ");
        String y = a.nextLine();
        int len1 = x.length();
        int len2 = y.length();

        int lexOrder = x.compareTo(y);

        System.out.println("Length of word1: "+len1);
        System.out.println("Length of word2: "+len2);

        if (lexOrder == 0){
                System.out.println(x+" and "+y+" are equal");
        }
        else if (lexOrder < 0){
                System.out.println(x+" comes before "+y);
        }
        else{
                System.out.println(y+" comes before "+x);
        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output

```
Enter the first word:
Code
Enter the second word:
Tantra
Length of word1: 4
Length of word2: 6
Code comes before Tantra
```

Test Case - 2

User Output

```
Enter the first word:
India
Enter the second word:
USA
Length of word1: 5
Length of word2: 3
India comes before USA
```

Test Case - 3

User Output

```
Enter the first word:
Dog
Enter the second word:
Cat
Length of word1: 3
Length of word2: 3
Cat comes before Dog
```

Test Case - 4

User Output

```
Enter the first word:
Himalayas
Enter the second word:
ocean
Length of word1: 9
Length of word2: 5
Himalayas comes before ocean
```

S.No: 41	Exp. Name: ***Super Keyword***	**Date: 2023-11-13**

Aim:
An online shopping platform, ECart, allows customers to add items to their shopping cart. Each customer has a shopping cart where they can add multiple items. Once items are added, the system displays all the items in the customer's cart.
Write the *Customer* class which is the child class of *ECart*. Also create *addItems* method in the customer class which does the following

Note: addItems takes products array of String type as parameter
Sample Test Output

```
Command line args books veggies fruits
The items in your cart are:
books
veggies
fruits
```

Source Code:

q1372/Main.java

```
package q1372;
class ECart {
    public String[] items;
    public void itemsLen(int len){
        items = new String[len];
    }
    public void display(){
        System.out.println("The items in your cart are: ");
        for(int i = 0; i < items.length; i++){
            System.out.println(items[i]);
        }
    }
}
//Write code here....
class Customer extends ECart{
        public void addItems(String[] items){
                itemsLen(items.length);
                System.arraycopy(items,0,this.items,0,items.length);
                display();
        }
}

class Main{
    public static void main(String[] args) {
        Customer c1 = new Customer();
        String[] NumOfItems = new String[args.length];
        for(int i = 0; i < args.length; i++){
            NumOfItems[i] = args[i];
        }
        c1.addItems(NumOfItems);
    }
}
```

Execution Results - **All test cases have succeeded!**

Test Case - 1

User Output

```
The items in your cart are:
drinks
chocolates
```

Test Case - 2

User Output

```
The items in your cart are:
stationery
clothes
bags
shoes
jewellery
```

Test Case - 3

User Output

```
The items in your cart are:
lamp
```

Test Case - 4

User Output

```
The items in your cart are:
books
veggies
fruits
```

S.No: 42	Exp. Name: ***Using Super Keyword***	Date: 2023-11-13

Aim:
CodeTantra is an online learning platform. When a new student registers on the platform, they are greeted with a welcome message. The system has been programmed to automate this process.
Complete the given code by writing the *Student* class by extending the *CodeTantra* class which produces the following output
Sample Test Case:

```
Command line args Aditya
Aditya Your registration is complete
Hello Aditya, Welcome to CodeTantra!!
```

Note : Also create register method in student class which brings out the result

Source Code:

q1371/Main.java

```
package q1371;
class CodeTantra {
    public void greeting(String name){
        System.out.println("Hello "+name+", Welcome to CodeTantra!!");
    }
}
//Write code here...
class Student extends CodeTantra{
        public void register(String name){
                System.out.println(name+" Your registration is complete");
                greeting(name);
        }
}


public class Main {
    public static void main(String[] args) {
        Student s1 = new Student();
        String name = args[0];
        s1.register(name);
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Aditya Your registration is complete
Hello Aditya, Welcome to CodeTantra!!
```

Test Case - 2

User Output

```
Lahari Your registration is complete
```

```
Hello Lahari, Welcome to CodeTantra!!
```

Test Case - 3

User Output

```
John Your registration is complete
Hello John, Welcome to CodeTantra!!
```

Test Case - 4

User Output

```
CT12345 Your registration is complete
Hello CT12345, Welcome to CodeTantra!!
```

S.No: 43	Exp. Name: ***Using super keyword***	Date: 2023-11-13

Aim:
When a method in a subclass overrides a method in superclass, it is still possible to call the overridden method using super keyword. If you write **super.func()** to call the function func(), it will call the method that was defined in the superclass.

You are given a partially completed code in the editor. Modify the code so that the code prints the following text:

```
Hello I am a motorcycle, I am a cycle with an engine.
My ancestor is a cycle who is a vehicle with pedals.
```

Source Code:

q1356/UsingSuper.java

```java
package q1356;

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

class BiCycle{
String define_me(){
return "a vehicle with pedals.";
}
}

class MotorCycle extends BiCycle{
String define_me(){
return "a cycle with an engine.";
}
MotorCycle(){
System.out.println("Hello I am a motorcycle, I am "+ define_me());

String temp=  super.define_me(); //Fix this line

System.out.println("My ancestor is a cycle who is "+ temp );
}
}
class UsingSuper{
public static void main(String []args){
MotorCycle M=new MotorCycle();
}
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Hello I am a motorcycle, I am a cycle with an engine.
My ancestor is a cycle who is a vehicle with pedals.
```

S.No: 44	Exp. Name: ***Write the code***	**Date: 2023-11-13**

Aim:

You work for a university that is planning to implement a new student information system. This system will store and manage student data, including their personal information and student IDs.where you have a "**Person**" class as the base class and a "**Student**" class as a subclass.
Some of the code is supplied for you to simplify the task.Create a program that receives input from the students(**String**-args[0],**int**-args[1],**int**-args[2]) and outputs the following:

Sample Test Case:

```
Command line args John Smith 22 23456
Student Information:
Name: John Smith
Age: 22
Student ID: 23456
```

Source Code:

```
q1338/University.java
```

```
package q1338;

class Person {
    private String name;
    private int age;

    //write the code for constructor Person with 2 arguments
    public Person(String name, int age){
        this.name = name;
        this.age = age;
    }

    //write the code for displayDetails()
    public void displayDetails(){
        System.out.println("Name: "+name);
        System.out.println("Age: "+age);
    }

}

class Student extends Person{//inherit Super class properties
    private int studentId;

        //write the code for constructor Student with 3 arguments
        public Student(String name, int age, int studentId){
                super(name, age);
                this.studentId = studentId;
        }
        //call superclass constructor using super



        //write the code for displayStudentInfo() method
        public void displayStudentInfo(){
                displayDetails();
                System.out.println("Student ID: "+      studentId);
        }
        //call displayDetails() inside displayStudentInfo()




}
public class University {
    public static void main(String[] args) {
        //Receive 3 arguments from args[0],args[1],args[2] and assign to its respective
variables
        String name = args[0];
        int age = Integer.parseInt(args[1]);
        int studentId = Integer.parseInt(args[2]);


        Student student = new Student(name, age, studentId);

        System.out.println("Student Information:");
        student.displayStudentInfo();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Student Information:
Name: Alice Johnson
Age: 20
Student ID: 12345
```

Test Case - 2

User Output

```
Student Information:
Name: John Smith
Age: 22
Student ID: 23456
```

Test Case - 3

User Output

```
Student Information:
Name: Emily Davis
Age: 19
Student ID: 34567
```

Test Case - 4

User Output

```
Student Information:
Name: Michael Wilson
Age: 23
Student ID: 45678
```

S.No: 45	Exp. Name: ***Inheritance***	**Date: 2023-11-13**

Aim:

Write the following code in your editor below:

1. A class named *Arithmetic* with a method named *add* that takes 2 integers as parameters and returns an integer denoting their sum.
2. A class named *Adder* that inherits from a superclass named *Arithmetic*.

Your classes should **not** be be **public**.

Output Format

You are responsible for printing output. Print the result of 2 calls to Adder's `add(int,int)` method as 2 space-separated integers.

One call is already made for your reference.

Take two inputs from command-line arguments(i.e., args[0],args[1]).

Sample Output

The*main*method in the*Solution*class above should print the following:

```
Command line args 112 567
My superclass is: Arithmetic
42 679
```

Source Code:

Solution.java

```java
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

//Write your code here
class Arithmetic{
        int add(int a, int b){
                return a+b;
        }
}

class Adder extends Arithmetic{

}



public class Solution{
public static void main(String []args){
// Create a new Adder object
Adder a = new Adder();
// Print the name of the superclass on a new line
System.out.println("My superclass is: " + a.getClass().getSuperclass().getName());
// Print the result of 2 calls to Adder's `add(int,int)` method as 2 space-separated
integers:
//use args[0],args[1] for inputs

System.out.print(a.add(10,32)+"
"+a.add(Integer.parseInt(args[0]),Integer.parseInt(args[1]))+ "\n");


}
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
My superclass is: Arithmetic
42 679
```

Test Case - 2

User Output

```
My superclass is: Arithmetic
42 19133
```

Test Case - 3

User Output

```
My superclass is: Arithmetic
42 8894
```

Test Case - 4

User Output

```
My superclass is: Arithmetic
42 5
```

Aim:

Using inheritance, one class can acquire the properties of others. Consider the following Animal class:

```
class Animal{
    void walk(){
        System.out.println("I am walking");
    }
}
```

This class has only one method, walk. Next, we want to create a Bird class that also has a fly method. We do this using extends keyword:

```
class Bird extends Animal {
    void fly() {
        System.out.println("I am flying");
    }
}
```

Finally, we can create a Bird object that can both fly and walk.

```
public class Solution{
   public static void main(String[] args){
      Bird bird = new Bird();
      bird.walk();
      bird.fly();
   }
}
```

The above code will print:

```
I am walking
I am flying
```

This means that a Bird object has all the properties that an Animal object has, as well as some additional unique properties.

You must add a fly method along with a sing method to the Bird class,

and then modify the main method so that the code prints the following lines on user's command-line input(Convert the input string to lowercase):

```
I am walking
I am flying
I am singing
```

Source Code:

q1351/Solution.java

```
package q1351;
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

class Animal{
        void walk(){
                System.out.println("I am walking");
        }
}

class Bird extends Animal{
        //write your code here
        void fly(){
                System.out.println("I am flying");
        }
        void sing(){
                System.out.println("I am singing");
        }


}

public class Solution{
public static void main(String args[]){
                Bird bird = new Bird();
        //take input from args[0] and convert it to lowercase
        String action = args.length>0? args[0].toLowerCase(): "";
        switch (action) {
            case "walk":
                bird.walk();
                break;
            //Modify the code
            case "fly":
                bird.fly();
                break;
            case "sing":
                bird.sing();
                break;




            default:
                System.out.println("Invalid action. Use 'walk', 'fly', or 'sing'.");
        }
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
I am walking
```

Test Case - 2

User Output

```
I am flying
```

Test Case - 3

User Output

```
I am singing
```

Test Case - 4

User Output

```
Invalid action. Use 'walk', 'fly', or 'sing'.
```

S.No: 47	Exp. Name: ***Method Overriding and super***	**Date: 2023-11-13**

Aim:

Imagine that you are creating a toy robot to amuse kids that can shape clay into the desired shape.Write a code which enables robot to mould the shapeless clay into desired shape.

Sample Test Case:

```
Command line args: Traingle
Clay is shapeless
Clay is moulded into Traingle
```

Source Code:

q1374/Main.java

```
package q1374;

class Clay {
   String shape;
  Clay(String shape){
    this.shape=shape;
  }
    void mould() {
        System.out.println("Clay is "+shape);
    }
}

class Shape extends Clay {
  Shape(String shape){
   //Complete the code
        super(shape);
 }
        @Override
    void mould() {
        //Mould the clay into given shape
                System.out.println("Clay is moulded into "+shape);

    }
}

public class Main {
    public static void main(String[] args) {
        Clay myShape1 = new Clay("shapeless");
        Clay myShape2 = new Shape(args[0]);
        myShape1.mould();
        myShape2.mould();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Clay is shapeless
Clay is moulded into Buildings
```

Test Case - 2

User Output

```
Clay is shapeless
Clay is moulded into Birds
```

Test Case - 3

User Output

```
Clay is shapeless
Clay is moulded into a House
```

Test Case - 4

User Output

```
Clay is shapeless
Clay is moulded into an Island
```

S.No: 48	Exp. Name: ***Method Overriding***	**Date: 2023-11-13**

Aim:
Gaurav is a member of the automotive testing division. Gaurav is required to change the General statement to a Specific statement that describes the type of vehicle and its status after testing every vehicle.
To assist Gaurav, create a program that modifies the prior statement and prints both the General statement and the Specific statement, accordingly.

Sample Test Case:
```
Command line args Motor-Cycle
General Statement: Vehicle is working...
Specific Statement: Motor-Cycle is working...
```
Source Code:

q1373/Vehicle.java
```
package q1373;

class Statement {
    void run() {
        System.out.println("General Statement: Vehicle is working...");
    }
}

class Vehicle extends Statement {
  String vehicle;
  //update the variable
  Vehicle(String vehicle){
        this.vehicle = vehicle;
  }

  //Override run method
  void run(){
        super.run();
        System.out.println("Specific Statement: "+vehicle+" is working...");
  }



  public static void main(String args[]) {

        Vehicle obj = new Vehicle(args[0]);
        obj.run();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output
```
General Statement: Vehicle is working...
Specific Statement: Bike is working...
```

Test Case - 2

User Output

```
General Statement: Vehicle is working...
Specific Statement: Car is working...
```

Test Case - 3

User Output

```
General Statement: Vehicle is working...
Specific Statement: Truck is working...
```

Test Case - 4

User Output

```
General Statement: Vehicle is working...
Specific Statement: Van is working...
```

S.No: 49	Exp. Name: ***Method Overriding***	**Date: 2023-11-13**

Aim:
Complete the code in your editor by writing an overridden **getName**, **getNumberOfTeamMembers** methods that prints the same statement as the superclass **getName**, **getNumberOfTeamMembers** method, except that it should replace **11**with**n**(the number of players) and **Generic Sports** with given sport(Type of sport which is stored in String variable **sport**).
Override the methods allowing it to adapt to various sports with distinct characteristics and requirements.

Sample Test Case:
```
Command line args 15 Foot Ball
Generic Sports
Each team has 11 players in Generic Sports
Foot Ball
Each team has 15 players in Foot Ball
```

Source Code:

```
q1359/MethodOverriding.java
```

```
package q1359;
import java.util.*;
class Sports{

    String getName(){
        return "Generic Sports";
    }

    void getNumberOfTeamMembers(){
        System.out.println( "Each team has 11 players in " + getName() );
    }
}

class Soccer extends Sports{
  private static int players;
  private static String sport;
  Soccer(int players,String sport){
    this.players=players;
    this.sport=sport;
  }
    // Write your overridden getName() method here
  String getName(){
        return sport;
  }
  void getNumberOfTeamMembers(){
        System.out.println("Each team has "+players+" players in "+getName());
  }



        // Write your overridden getNumberOfTeamMembers method here



}

public class MethodOverriding{

    public static void main(String []args){
      int players=Integer.parseInt(args[0]);
      String sport=args[1];
        Sports c1 = new Sports();
        Soccer c2 = new Soccer(players,sport);
        System.out.println(c1.getName());
        c1.getNumberOfTeamMembers();
        System.out.println(c2.getName());
        c2.getNumberOfTeamMembers();
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output

```
Generic Sports
Each team has 11 players in Generic Sports
Foot Ball
Each team has 15 players in Foot Ball
```

Test Case - 2

User Output

```
Generic Sports
Each team has 11 players in Generic Sports
Volley-Ball
Each team has 11 players in Volley-Ball
```

Test Case - 3

User Output

```
Generic Sports
Each team has 11 players in Generic Sports
Throw Ball
Each team has 10 players in Throw Ball
```

Test Case - 4

User Output

```
Generic Sports
Each team has 11 players in Generic Sports
Cricket
Each team has 11 players in Cricket
```

S.No: 50	Exp. Name: ***Method overriding***	**Date: 2023-11-13**

Aim:

A circus is in town and they have a variety of performers ready to entertain the audience. The performers include a clown, a juggler, an acrobat, and a magician. Each performer has their own unique act. The circus has an automated system that announces the next performer based on the schedule.

Complete the below code where you need to add the Acrobat and Magician classes and also complete the switch case statements.

Sample Test Output 1:

```
Command line args clown
Clown is juggling balls...
```

Sample Test Output 2:

```
Command line args ringmaster
No such performer
```

Source Code:

```
q1370/Juggler.java
```

```
package q1370;
class Performer {
    void perform() {
        System.out.println("Performer is performing...");
    }
}

// Derived class Clown
class Clown extends Performer {
    @Override
    void perform() {
        System.out.println("Clown is juggling balls...");
    }
}
//Write code for Acrobat and Magician classes...
class Acrobat extends Performer{
        void perform(){
                System.out.println("Acrobats are doing the aerial acts...");
        }
}

class Magician extends Performer{
        void perform(){
                System.out.println("Magicians are captivating the audience with their 
tricks...");
        }
}



// Derived class Juggler
class Juggler extends Performer {
    @Override
    void perform() {
        System.out.println("Juggler is juggling pins...");
    }

    public static void main(String[] args) {
        String person = args[0];
        switch(person){
            case "clown":
                Performer performer1 = new Clown();
                performer1.perform();
                break;
            case "juggler":
                Performer performer2 = new Juggler();
                performer2.perform();
                break;
           //complete the cases here....
            case "acrobat":
                Performer performer3 = new Acrobat();
                performer3.perform();
                break;
            case "magician":
                Performer performer4 = new Magician();
                performer4.perform();
                System.out.println("No such performer");
        }

    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
No such performer
```

Test Case - 2

User Output

```
Clown is juggling balls...
```

Test Case - 3

User Output

```
Acrobats are doing the aerial acts...
```

Test Case - 4

User Output

```
Magicians are captivating the audience with their tricks...
```

Test Case - 5

User Output

```
Juggler is juggling pins...
```

S.No: 51	Exp. Name: ***Method Overriding***	**Date: 2023-11-13**

Aim:

Ravi wants to create a toy for kids that speaks in the voice of the animal when its image is clicked.Ravi's modifications made the animal's name appear(String),when the image is clicked.

Create a program which takes that name and displays the corresponding statement.

Sample Test Case:(Observe and match other test case statements)

```
Command line args dog
The dog barks.
```

Source Code:

```
q1339/AnimalSound.java
```

```
package q1339;

class Animal {
    void makeSound() {
        System.out.println("The animal makes a sound.");
    }
}

        //Create Dog class and extend Animal class
class Dog extends Animal{
        void makeSound(){
                System.out.println("The dog barks.");
        }
}
        //Override makeSound() method


        //Create Cat class and extend Animal class
class Cat extends Animal{
        void makeSound(){
                System.out.println("The cat meows.");
        }
}
        //Override makeSound() method



public class AnimalSound {
    public static void main(String[] args) {
         Animal myAnimal;

       //convert given word to lowercase
        //if args[0] is equal to dog initiliaze myAnimal with new Dog() and call the method
        //if args[0] is equal to cat initiliaze myAnimal with new Cat() and call the method
        //else print Unknown animal type: animalType

        String animalType = args[0].toLowerCase();
        if (animalType.equals("dog")){
                myAnimal = new Dog();
                myAnimal.makeSound();
        }
        else if (animalType.equals("cat")){
                myAnimal = new Cat();
                myAnimal.makeSound();
        }
        else{
                System.out.println("Unknown animal type: "+animalType);
        }



    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
The dog barks.
```

Test Case - 2

User Output

```
The cat meows.
```

Test Case - 3

User Output

```
Unknown animal type: fox
```

Test Case - 4

User Output

```
Unknown animal type: lion
```

S.No: 52	Exp. Name: ***Abstract***	**Date: 2023-11-29**

Aim:

Vehicle Information System : Consider you are working in a manufacturing company where you are required to create an abstract class Vehicle with an abstract method description. Also, define the method in Category class which returns you the information of the vehicle as below:

```
Command line args 2 bike
The number of wheels in a bike are 2
```

Note : The description method should accept two parameters where the first parameter is of type int and second of type String

Source Code:

q1358/Main.java

```
package q1358;
//Write your code here...
abstract class Vehicle{
        abstract void description(int numOfWheels, String typeOfVehicle);
}
class Category extends Vehicle{
        void description(int numOfWheels, String typeOfVehicle){
                System.out.println("The number of wheels in a " +typeOfVehicle+" are 
"+numOfWheels);
        }
}

public class Main {
    public static void main(String[] args){
        int numOfWheels = Integer.parseInt(args[0]);
        String typeOfVehicle = args[1];
        Vehicle b1 = new Category();
        b1.description(numOfWheels,typeOfVehicle);

    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
The number of wheels in a bike are 2
```

Test Case - 2

User Output

```
The number of wheels in a car are 4
```

Test Case - 3

User Output

```
The number of wheels in a bicycle are 2
```

Test Case - 4

User Output

```
The number of wheels in a aeroplane are 6
```

S.No: 53	Exp. Name: ***Final keyword***	**Date: 2023-11-29**

Aim:
Complete the program to show the following output

Note: *Final variable cannot be reassigned but can be used with other variables*
Sample Test Case:

```
Command line args 4.585
Final number is: 4.585
Final number multiplied with 2: 9.17
Final number added with 4: 8.585
```

Source Code:

q1363/FinalVariable.java

```
package q1363;

public class FinalVariable
{
public static void main(String[] args) {
                final double num;
                num = Double.parseDouble(args[0]);
                System.out.println("Final number is: "+num);
                //write your code here
                double multipliedByTwo = num*2;
                double addedWithFour = num+4;
                System.out.println("Final number multiplied with 2: "+multipliedByTwo);
                System.out.println("Final number added with 4: "+addedWithFour) ;
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Final number is: 4.585
Final number multiplied with 2: 9.17
Final number added with 4: 8.585
```

Test Case - 2

User Output

```
Final number is: 56.89675
Final number multiplied with 2: 113.7935
Final number added with 4: 60.89675
```

Test Case - 3

User Output

```
Final number is: 5.36987456
Final number multiplied with 2: 10.73974912
Final number added with 4: 9.36987456
```

Test Case - 4

User Output

```
Final number is: 0.2365
Final number multiplied with 2: 0.473
Final number added with 4: 4.2365
```

S.No: 54	Exp. Name: ***Final Keyword for Constants***	**Date: 2023-11-29**

Aim:

In many applications, there are certain values that don't change once they are assigned.
Observe the equation below and decide which needs to be specified as a final and print the result as shown in sample test cases.
Equation: x*y + 45 + 3.14*z . Do not use 45 and 3.14 directly in order to calculate the result. Store them in a final variable.

Note : Take x,y from user which are integers and z as double
Sample test case:

```
Command line args 2 3 5.2
The result is 67.328
```

Source Code:

q1421/Main.java

```
package q1421;
public class Main {
    public static void main(String[] args) {
       final int a = 45;
       final double pi = 3.14;
       double res = 0;
                int x = Integer.parseInt(args[0]);
                int y = Integer.parseInt(args[1]);
                double z = Double.parseDouble(args[2]);

                res = x*y+a+pi*z;
                System.out.println("The result is "+res);


    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
The result is 67.328
```

Test Case - 2

User Output

```
The result is 170.7
```

Test Case - 3

User Output

```
The result is 63.56
```

Test Case - 4

User Output

```
The result is 130.2648
```

S.No: 55	Exp. Name: ***Interface***	**Date: 2023-11-29**

Aim:

An employee abruptly quit while working on a calculator application.You were his replacement.It's your responsibility to decipher that code and enable the calculator's Subtraction and multiplication capability.

Sample Test Case 1:

```
Command line args multiply 2 4
Result: 8.0
```

Sample Test Case 2:

```
Command line args divide 2 4
Invalid operation: divide
Result: 0.0
```

Source Code:

```
q1369/CalculatorApp.java
```

```
package q1369;
interface Calculator {
    double calculate(double a, double b);
}

class Addition implements Calculator {
    @Override
    public double calculate(double a, double b) {
        return a + b;
    }
}
//write your code here
class Subtraction implements Calculator{
        public double calculate(double a,double b){
                return a-b;
        }
}
class Multiplication implements Calculator{
        public double calculate(double a, double b){
                return a*b;
        }
}
public class CalculatorApp {
    public static void main(String[] args) {
        String operation = args[0];
        double operand1 = Double.parseDouble(args[1]);
        double operand2 = Double.parseDouble(args[2]);
        double result=0;
        Calculator calculator;
        switch(operation){
            case "add":
                {
                    calculator = new Addition();
                    result = calculator.calculate(operand1, operand2);
                    break;
                }
        //Modify switch cases to get required output
                        case "subtract":
                                calculator = new Subtraction();
                                result = calculator.calculate(operand1,operand2);
                                break;
                        case "multiply":
                                calculator = new Multiplication();
                                result = calculator.calculate(operand1,operand2);
                                break;
                        default:
                                System.out.println("Invalid operation: "+operation);
        }

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

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Result: 8.0
```

Test Case - 2

User Output

```
Invalid operation: divide
Result: 0.0
```

Test Case - 3

User Output

```
Result: 20.0
```

Test Case - 4

User Output

```
Result: 13.0
```

S.No: 56	Exp. Name: ***Multiple inheritance with interfaces***	Date: 2023-11-29

Aim:

Gift Eligibility System:

Consider there is a promotional offer where customers who have accumulated more than 1000 points in Reliance Mart and are using a Reliance Jio Sim card are eligible for a special gift hamper.
Complete the code by taking two inputs from user, where the first input is points and the second is a boolean input which stores whether he/she is having a jio sim or not and pass the inputs to the object of the class.
Define the functions in the Customer class which return the points and isJioUser respectively
Also write the if else conditions so as to print as shown in sample test cases

Sample Test Case 1 :

```
Command line args 1500 true
Congratulations! You are eligible for a gift hamper.
```

Sample Test Case 2 :

```
Command line args 100 true
Sorry, you are not eligible for a gift hamper.
```

Source Code:

```
q1419/Customer.java
```

```
package q1419;
interface Reliancemart {
    int getPoints();
}

interface JioSim {
    boolean isJioUser();
}

public class Customer implements Reliancemart, JioSim{
    private int points;
    private boolean isJioUser;

    public Customer(int points, boolean isJioUser) {
        this.points = points;
        this.isJioUser = isJioUser;
    }

    //Define functions here...
        public int getPoints(){
                return points;
        }
        public boolean isJioUser(){
                return isJioUser;
        }
    public static void main(String[] args) {

        //Write code here...
                int points = Integer.parseInt(args[0]);
                Boolean isJioUser = Boolean.parseBoolean(args[1]);

                Customer customer = new Customer(points, isJioUser);
                if (customer.getPoints()>1000 && customer.isJioUser()){
                        System.out.println("Congratulations! You are eligible for a gift
hamper.");
                }
                else{
                        System.out.println("Sorry, you are not eligible for a gift
hamper.");
                }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Congratulations! You are eligible for a gift hamper.
```

Test Case - 2

User Output

```
Sorry, you are not eligible for a gift hamper.
```

Test Case - 3

User Output

```
Congratulations! You are eligible for a gift hamper.
```

Test Case - 4

User Output

```
Sorry, you are not eligible for a gift hamper.
```

S.No: 57	Exp. Name: ***Nested Interface***	**Date: 2023-11-29**

Aim:

Default Message Addition System:

Consider you need to create a system which adds a default greeting based on the time a customer logs into the system.
If the time is between 0 and 11, it returns “Good Morning”. If it’s between 12 and 17, it returns “Good Afternoon”. If it’s between 18 and 24, it returns “Good Evening”. Otherwise, it returns “Welcome!”.

Complete the given code which uses nested interfaces to achieve the mentioned.

Sample test case 1:

```
Command line args 4
Good Morning This is IDBI bank
```

Sample test case 2:

```
Command line args 22
Good Evening This is IDBI bank
```

Source Code:

Main.java

```
interface OuterInterface {
    String outerMethod(int time);

    // Nested interface
    interface InnerInterface {
        void innerMethod(String name);
    }
}

// Class that implements the outer and inner interfaces
class MyClass implements OuterInterface {
    public String outerMethod(int time) {
        String greeting = "";
                if (time>=0 && time<=11){
                        greeting="Good Morning";
                }
                else if (time>=12 && time<=17){
                        greeting="Good Afternoon";
                }
                else if (time>=18 && time<=24){
                        greeting="Good Evening";
                }
                else{
                        greeting="Welcome!";
                }

                return greeting+" This is IDBI bank";
    }

    public void innerMethod(String greeting) {
        System.out.println(greeting);
    }
}

public class Main {
    public static void main(String[] args) {
        int time = Integer.parseInt(args[0]);
                MyClass m = new MyClass();
                String greetingMessagge = m.outerMethod(time);
                m.innerMethod(greetingMessagge);
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Good Morning This is IDBI bank
```

Test Case - 2

User Output

```
Good Evening This is IDBI bank
```

Test Case - 3

User Output

```
Welcome! This is IDBI bank
```

Test Case - 4

User Output

```
Good Afternoon This is IDBI bank
```

S.No: 58	Exp. Name: ***Using inbuilt Scanner class***	**Date: 2023-12-09**

Aim:
Imagine you are a tour guide at a national park, and you are leading a group of tourists on a guided nature hike. You want to gather some basic information from each tourist to ensure their safety and provide them with an enjoyable experience.

As a tour guide at the national park, you need to collect essential information from each tourist before the hike. You decide to use the ***java.util package*** and the ***Scanner*** class to capture this data. Describe how you would use the Scanner class in this real-world scenario to gather the following information from each tourist:

Note : *Use for loop to take data of 2 tourists*

Sample Test Case:
Input :
```
Welcome to the National Park Guided Hike!
Tourist 1 - Please provide the following information:
Full Name: Ramesh
Age: 35
Emergency Contact: 1112223334
Any Medical Conditions or Allergies (if any): Fine
Tourist 2 - Please provide the following information:
Full Name: Raju
Age: 25
Emergency Contact: 7778889990
Any Medical Conditions or Allergies (if any): Fine
```
Output :
```
Tourist Information for the Guided Hike:
-----------------------------------------
Tourist 1 Details:
Name: Ramesh
Age: 35
Emergency Contact: 1112223334
Medical Conditions: Fine
-----------------------------------------
Tourist 2 Details:
Name: Raju
Age: 25
Emergency Contact: 7778889990
Medical Conditions: Fine
-----------------------------------------
```
Source Code:

```
q1366/TourGuideProgram.java
```

```
package q1366;
//import scanner here
import java.util.Scanner;
public class TourGuideProgram {
    public static void main(String[] args) {
        // Create a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);

        System.out.println("Welcome to the National Park Guided Hike!");

        // Initialize variables to store tourist information
        String[] touristNames = new String[5];
        int[] touristAges = new int[5];
        long[] emergencyContacts = new long[5];
        String[] medicalConditions = new String[5];

        // Gather information from each tourist
        for (int i=0;i<2;i++){
                        System.out.print("Tourist "+(i+1)+" - ");
                        System.out.print("Please provide the following information:");
                        System.out.println();
                        System.out.print("Full Name: ");
                        touristNames[i]=scanner.nextLine();
                        System.out.print("Age: ");
                        touristAges[i]=scanner.nextInt();
                        scanner.nextLine();
                        System.out.print("Emergency Contact: ");
                        emergencyContacts[i]=scanner.nextLong();
                        scanner.nextLine();
                        System.out.print("Any Medical Conditions or Allergies (if any): ");
                        medicalConditions[i]=scanner.nextLine();
                }







        // Display gathered information
        System.out.println("Tourist Information for the Guided Hike:");
        System.out.println("-----------------------------------------");
        for (int i = 0; i < 2; i++) {
            System.out.println("Tourist " + (i + 1) + " Details:");
            System.out.println("Name: " + touristNames[i]);
            System.out.println("Age: " + touristAges[i]);
            System.out.println("Emergency Contact: " + emergencyContacts[i]);
            System.out.println("Medical Conditions: " + medicalConditions[i]);
            System.out.println("-----------------------------------------");

        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Welcome to the National Park Guided Hike!
Tourist 1 - Please provide the following information:
Full Name:
Kiran
Age:
26
Emergency Contact:
2583691470
Any Medical Conditions or Allergies (if any):
Good
Tourist 2 - Please provide the following information:
Full Name:
Priya
Age:
20
Emergency Contact:
9876543210
Any Medical Conditions or Allergies (if any):
Fine
Tourist Information for the Guided Hike:
-----------------------------------------
Tourist 1 Details:
Name: Kiran
Age: 26
Emergency Contact: 2583691470
Medical Conditions: Good
-----------------------------------------
Tourist 2 Details:
Name: Priya
Age: 20
Emergency Contact: 9876543210
Medical Conditions: Fine
-----------------------------------------
```

Test Case - 2

User Output

```
Welcome to the National Park Guided Hike!
Tourist 1 - Please provide the following information:
Full Name:
Praveen
Age:
30
Emergency Contact:
```

```
9854612345
Any Medical Conditions or Allergies (if any):
Good
Tourist 2 - Please provide the following information:
Full Name:
Aniketh
Age:
19
Emergency Contact:
8555222321
Any Medical Conditions or Allergies (if any):
Good
Tourist Information for the Guided Hike:
----------------------------------------
Tourist 1 Details:
Name: Praveen
Age: 30
Emergency Contact: 9854612345
Medical Conditions: Good
----------------------------------------
Tourist 2 Details:
Name: Aniketh
Age: 19
Emergency Contact: 8555222321
Medical Conditions: Good
----------------------------------------
```

Test Case - 3

User Output

```
Welcome to the National Park Guided Hike!
Tourist 1 - Please provide the following information:
Full Name:
Vivek
Age:
24
Emergency Contact:
4442223335
Any Medical Conditions or Allergies (if any):
Fine
Tourist 2 - Please provide the following information:
Full Name:
chandu
Age:
45
Emergency Contact:
3336662221
Any Medical Conditions or Allergies (if any):
Good
```

```
Tourist Information for the Guided Hike:
-----------------------------------------
Tourist 1 Details:
Name: Vivek
Age: 24
Emergency Contact: 4442223335
Medical Conditions: Fine
-----------------------------------------
Tourist 2 Details:
Name: chandu
Age: 45
Emergency Contact: 3336662221
Medical Conditions: Good
-----------------------------------------
```

Test Case - 4

User Output

```
Welcome to the National Park Guided Hike!
Tourist 1 - Please provide the following information:
Full Name:
Ganga
Age:
35
Emergency Contact:
9996664447
Any Medical Conditions or Allergies (if any):
Fine
Tourist 2 - Please provide the following information:
Full Name:
Shiva
Age:
36
Emergency Contact:
7775553332
Any Medical Conditions or Allergies (if any):
Good
Tourist Information for the Guided Hike:
-----------------------------------------
Tourist 1 Details:
Name: Ganga
Age: 35
Emergency Contact: 9996664447
Medical Conditions: Fine
-----------------------------------------
Tourist 2 Details:
Name: Shiva
Age: 36
Emergency Contact: 7775553332
Medical Conditions: Good
```

```
-------------------------------------------
```

S.No: 59	Exp. Name: ***Util Package Currency Class***	**Date: 2023-12-09**

Aim:

Formatting Currencies: You are working on a financial application that displays amounts in different currencies. Use**java.util.Currency**to format amounts in USD, EUR, INR and JPY according to their usual display conventions in their respective countries.
Write a java program such that it takes the currency ISO code from the command Line arguments and it returns the symbol of the currency.

Sample input and output:

```
Command line args USD
USD $
```

Source Code:

q1534/Main.java

```
package q1534;
import java.util.Currency;
//Type Content here...

public class Main{
        public static void main(String[] args){
                String currencyCode = args[0];
                try{
                        Currency c = Currency.getInstance(currencyCode);
                        String currencySymbol = c.getSymbol();

                        System.out.println(currencyCode+" "+currencySymbol);
                }
                catch (IllegalArgumentException e){
                        System.out.println("Invalid Currency Code: "+currencyCode);
                }
        }



}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
USD $
```

Test Case - 2

User Output

```
PKR PKR
```

Test Case - 3

User Output

```
JMD JMD
```

Test Case - 4

User Output

```
AUD A$
```

S.No: 60	Exp. Name: ***Built in Time and Util Package***	Date: 2023-12-09

Aim:

Event Planning System: Imagine you are planning a special event and you want to ensure that it falls on a Saturday. You have a Java program that can tell you the day of the week for any given date. The program takes three command-line arguments: year, month, and date in that order.
The program uses the**java.time** package, which provides classes for working with dates, times, and durations. It includes the**LocalDate**class, which represents a date (year, month, day) in the ISO calendar system.
Your task is to create a **Main** class which returns the day of the given date.

Sample Input and output

```
Command line args 2023 12 4
Day of week in text : MONDAY
```

Source Code:

q1535/Main.java

```
package q1535;

//Type Content here...
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.Locale;

public class Main{
        public static void main(String[] args){
                int year = Integer.parseInt(args[0]);
                int month = Integer.parseInt(args[1]);
                int day = Integer.parseInt(args[2]);

                try{
                        LocalDate date = LocalDate.of(year, month, day);
                        String dayOfWeek =
date.getDayOfWeek().getDisplayName(java.time.format.TextStyle.FULL, Locale.ENGLISH);
                        System.out.println("Day of week in text :
"+dayOfWeek.toUpperCase());
                }
                catch (Exception e){
                        System.out.println("Invalid date");
                }
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Day of week in text : MONDAY
```

Test Case - 2

User Output

```
Day of week in text : WEDNESDAY
```

Test Case - 3

User Output

```
Day of week in text : MONDAY
```

Test Case - 4

User Output

```
Day of week in text : SATURDAY
```

S.No: 61	Exp. Name: ***User Defined Package***	Date: 2023-12-09

Aim:

Write a Java program that takes two command-line arguments, converts them into integers, and then uses a **Sample** class from the **p2** package to perform arithmetic operations on these numbers.

Note: use import statement in the Main.java file so that you are able to access the methods from **p2** package

Sample input and output

```
Command line args 45 5
50
40
225
9
0
```

Source Code:

q1536/Main.java

```
package q1536;
//import here
import p2.Sample;
public class Main  {
// write the code here
        public static void main(String[] args){
                int num1 = Integer.parseInt(args[0]);
                int num2 = Integer.parseInt(args[1]);

                Sample s = new Sample();

                int sumResult = s.add(num1, num2);
                int differenceResult = s.sub(num1, num2);
                int productResult = s.mul(num1, num2);
                int quotientResult = s.div(num1, num2);
                int modulusResult = s.mod(num1, num2);
                System.out.println(sumResult);
                System.out.println(differenceResult);
                System.out.println(productResult);
                System.out.println(quotientResult);
                System.out.println(modulusResult);
        }
}
```

p2/Sample.java

```
package p2;

public class Sample {
	public int add(int a, int b) {
		return a+b;
	}
	public int sub(int a, int b) {
		return a-b;
	}
	public int mul(int a, int b) {
		return a*b;
	}
	public int div(int a, int b) {
		return a/b;
	}
	public int mod(int a, int b) {
		return a%b;
	}
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
35
-11
276
0
12
```

Test Case - 2

User Output

```
28
22
75
8
1
```

Test Case - 3

User Output

```
28
20
96
6
0
```

Test Case - 4
User Output
19
13
48
5
1

S.No: 62	Exp. Name: ***User Defined Package***	**Date: 2023-12-09**

Aim:

Your task is to create a package **p2** in **Sample2.java** file which contains two static methods named **firstName** and **lastName**. Both methods have one String parameter and both of them return the string. Also, In the Main2.java file accept the names from the command line arguments and call the methods created in the p2 package.

Sample Test case

```
Command line args code tantra
code
tantra
```

Source Code:

p3/Main.java

```java
package p3;
// import package
import p2.Sample;
public class Main {
        // write the code here
        public static void main(String[] args){
                String name1 = args[0];
                String name2 = args[1];
                System.out.println(name1);
                System.out.println(name2);
        }
}
```

p2/Sample.java

```java
// create package
package p2;
public class Sample {
        public static String firstName(String name){
                return name;
        }
        public static String lastName(String name){
                return name;
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
code
tantra
```

Test Case - 2

User Output

```
ganga
yamuna
```

Test Case - 3

User Output

```
good
morning
```

Test Case - 4

User Output

```
rama
sita
```

S.No: 63	Exp. Name: ***User defined Package***	**Date: 2023-12-09**

Aim:

Hospital Management System : Imagine you are a software developer at a large hospital. The hospital uses a Java-based Hospital Management System (HMS) to manage patient's registration and makes appointments.
Your task is to create a class named**Appointment**in the**hms**package in the *Appointment.java* file. This class should have two methods:

3. registration(String s) : This method should take a patient's name as a parameter and print a message saying that the patient is registered.
4. timeSlot(int hour) : This method should take an integer representing the hour of the day (in 24-hour format) when the patient wants to book an appointment. The method should check if the hour is valid (i.e., less than or equal to 24). If the hour is not valid, it should print a message saying 'Not a valid time'. If the hour is during the time slots 12-14 or 20-22, it should print a message asking the patient to book another time. For all other valid hours, it should print a message saying 'Your appointment is booked'.

Also do the required in Patient.java.

Sample Input and Output:

```
Command line args ram 11
The patient ram is registered
Your appointment is booked
```

Source Code:

`q1541/Patient.java`

```
package q1541;
//import package here


import hms.Appointment;


public class Patient {
    public static void main(String[] args){
        Appointment a1 = new Appointment();
        String name = args[0];
        int hr = Integer.parseInt(args[1]);
        a1.registration(name);
        a1.timeSlot(hr);
    }
}
```

`hms/Appointment.java`

```
package hms;
public class Appointment{
	public void registration(String s){
		System.out.println("The patient "+s+" is registered");
	}
	public void timeSlot(int hour){
		if (hour<=24){
			if ((hour >=12 && hour <=14) || (hour>=20 && hour<=22)){
				System.out.println("Please book other time");
			}
			else{
				System.out.println("Your appointment is booked");
			}
		}
		else{
			System.out.println("Not a valid time");
		}
	}
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
The patient ram is registered
Your appointment is booked
```

Test Case - 2

User Output

```
The patient geeta is registered
Please book other time
```

Test Case - 3

User Output

```
The patient anita is registered
Your appointment is booked
```

Test Case - 4

User Output

```
The patient sai is registered
Not a valid time
```

S.No: 64	Exp. Name: ***User defined packages***	**Date: 2023-12-10**

Aim:

Library Management System:
Consider you are a software developer at a library. The library has a system where each book is represented by an instance of the**Books** class. The**Books**class has the following attributes:*bookName,authorName,quantity*, and an array**books**that stores the names of the books.
The**Books**constructor takes the book name, author name, and quantity as parameters. If the quantity is more than 10, it prints "Cannot add more than 10 books". Otherwise, it creates an array of size equal to the quantity and fills it with the book name.
Your task is to createdisplayBooksmethod that prints the names of the books in thebooksarray. Also in the Main.java file, take the inputs using command line arguments and create Books instance and call the displayBooks method

Sample Input and Output 1:

```
Command line args Pride and Prejudice Jane Austen 8
Pride and Prejudice
Pride and Prejudice
Pride and Prejudice
Pride and Prejudice
Pride and Prejudice
Pride and Prejudice
Pride and Prejudice
Pride and Prejudice
```

Sample Input and Output 2:

```
Command line args Hamlet Shakespeare 13
Cannot add more than 10 books
null
```

Source Code:

`library/Books.java`

```
package library;
public class Books {
    private String bookName;
    private String authorName;
    private int quantity;
    public String[] books = {null};
    public Books(String bookName, String authorName,int quantity) {
        this.bookName = bookName;
        this.authorName = authorName;
        this.quantity = quantity;
        if(quantity > 10) {
                System.out.println("Cannot add more than 10 books");
                        System.out.println("null");
        }else {
                books = new String[books.length+quantity-1];
                for(int i = 0;i < books.length; i++) {
                        books[i] = bookName;
                }
        }
    }
    //write code here...
        public void displayBooks(){
                for (String book:books){
                        System.out.println(book);
                }
        }


}
```

q1542/Main.java

```
package q1542;
//import statement here
import library.Books;
public class Main{
        public static void main(String[] args){
                String bookName = args[0];
                String authorName = args[1];
                int quantity = Integer.parseInt(args[2]);
                Books bookInstance = new Books(bookName, authorName, quantity);
                if (quantity<=10){
                        bookInstance.displayBooks();
                }
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output

```
Pride and Prejudice
Pride and Prejudice
Pride and Prejudice
Pride and Prejudice
Pride and Prejudice
Pride and Prejudice
Pride and Prejudice
Pride and Prejudice
```

Test Case - 2

User Output

```
Cannot add more than 10 books
null
```

Test Case - 3

User Output

```
Wings of Fire
Wings of Fire
```

Test Case - 4

User Output

```
To kill a mockingbird
To kill a mockingbird
To kill a mockingbird
```

S.No: 65	Exp. Name: ***Flight Booking System***	**Date: 2023-12-10**

Aim:

Flight Reservation System: Consider you are a software developer at a travel agency.
The agency has a system where each flight is represented by an instance of the Flight class.Your task is to create a system which shows the reserved, cancelled and remaining seats after every booking.
Tasks to be done in each file:
1. FlightBooking.java: Create a package **p1** and create a static variable availableSeats of int type. Also,create a method
occupency with one int parameter named reserveSeats.In the method write the code for calculating availableSeats
2. FlightCancel.java: Here, create another package **p2** and import the **p1** package.Write the code for finding and printing the
remainingSeats
3. Flight.java: Create a package **p3** and import the other packages. Now, take the inputs from the command line arguments
which are **reservedSeats** and **cancelledSeats** respectively

Sample Test Case

```
Command line args 66 4
Total no.of seats in flight: 200
Reserved seats in flight:66
available seats in flight :134
cancelled seats in flight :4
The remaining seats in flight: 138
```

Source Code:

`p1/FlightBooking.java`

```
//create package p1
package p1;

public class FlightBooking {
    public static int totalSeats=200;
        //write code here
        public static int reservedSeats;

        // System.out.println("Total no.of seats in flight: "+totalSeats);
    // System.out.println("Reserved seats in flight:"+reservedSeats);
    //write code here
        public static void occupancy(int reserveSeats){
                reservedSeats=reserveSeats;
                int availableSeats = totalSeats-reservedSeats;
                System.out.println("Total no.of seats in flight: "+totalSeats);
                System.out.println("Reserved seats in flight:"+reservedSeats);
                System.out.println("available seats in flight :"+availableSeats);

    // System.out.println("available seats in flight :"+availableSeats);
        }
}
```

`p2/FlightCancel.java`

```java
// create package p2
package p2;
// import p1 here
import p1.FlightBooking;
public class FlightCancel {
    public static void cancel(int cancelledSeats) {
        System.out.println("cancelled seats in flight :"+cancelledSeats);
        // write code here
                int remainingSeats = FlightBooking.totalSeats-
FlightBooking.reservedSeats+cancelledSeats;
                System.out.println("The remaining seats in flight: "+remainingSeats);
    }
}
```

p3/Flight.java

```java
// craete pacakge p3
package p3;
// import p1&p2 packages here
import p1.FlightBooking;
import p2.FlightCancel;
public class Flight {
        public static void main(String[] args) {
        // write code here
                int reservedSeats = Integer.parseInt(args[0]);
                int cancelledSeats = Integer.parseInt(args[1]);

                FlightBooking.occupancy(reservedSeats);
                FlightCancel.cancel(cancelledSeats);
        }

}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Total no.of seats in flight: 200
Reserved seats in flight:56
available seats in flight :144
cancelled seats in flight :12
The remaining seats in flight: 156
```

Test Case - 2

User Output

```
Total no.of seats in flight: 200
Reserved seats in flight:25
available seats in flight :175
cancelled seats in flight :8
```

```
The remaining seats in flight: 183
```

Test Case - 3

User Output

```
Total no.of seats in flight: 200
Reserved seats in flight:38
available seats in flight :162
cancelled seats in flight :8
The remaining seats in flight: 170
```

Test Case - 4

User Output

```
Total no.of seats in flight: 200
Reserved seats in flight:42
available seats in flight :158
cancelled seats in flight :8
The remaining seats in flight: 166
```

S.No: 66	Exp. Name: ***Shooting Game Simulation***	**Date: 2023-12-10**

Aim:

Shooting Game Simulation:

In the video game,each gun is loaded initially with 6 bullets. The shoot method in **GunShot** class accepts the number of
bullets to be fired and returns the remaining **bullets**. If the remaining bullets are **equal to zero** call the **reload** method
which prints the message "**Reload** the **gun**". Also if the user gives the input more than 6 you need to print the message
"**You can't load more than 6 bullets**".

Tasks to be done:

In the **Gun.java file** write the unimplemented methods **shoot**(int bullets) and **reload**()

In the **GunShot.java** file create the method shoot

In the **Main.java** file accept the input from the user and call the methods accordingly.

Note: Make sure to import and create **packages** accordingly in all the files

Sample Test Case

```
Command line args 4
The no.of bullets:4
The no.of remaining bullets:2
```

Source Code:

`p1/Gun.java`

```
//create package p1
package p1;
public interface Gun {
        //write code here
        void shoot(int bullets);
        void reload();

}
```

`p2/GunShot.java`

```
//create Package p2
package p2;
//import Package p1
import p1.Gun;

public class GunShot implements Gun{
        public static int rounds=6;
        public int remainBullets;
        //write code here
        public void shoot(int bullets){
                if (bullets>rounds){
                        System.out.println("You can't load more then 6 bullets");
                        return;
                }
                remainBullets = rounds-bullets;
                System.out.println("The no.of bullets:"+bullets);
                System.out.println("The no.of remaining bullets:"+remainBullets);

                if (remainBullets==0){
                        reload();
                }
        }



        public void reload() {
                System.out.println("Reload the gun");
    }
}
```

p3/Main.java

```
// create package p3
package p3;
//import package
import p2.GunShot;
public class Main {
        public static void main(String[] args) {
                GunShot g = new GunShot();
                int bullets = Integer.parseInt(args[0]);
                g.shoot(bullets);

        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
The no.of bullets:5
The no.of remaining bullets:1
```

Test Case - 2

User Output

```
The no.of bullets:0
The no.of remaining bullets:6
```

Test Case - 3

User Output

```
The no.of bullets:6
The no.of remaining bullets:0
Reload the gun
```

Test Case - 4

User Output

```
You can't load more then 6 bullets
```

S.No: 67	Exp. Name: ***Using Protected methods in one package and using into another package***	**Date: 2023-12-10**

Aim:

Know your Birth Flower and Its Meaning:
Your task is to complete the function **meaning** which takes your birth flower as input and returns the meaning of the flower.
You can refer to the below table in order to return the messages

Flower	**Meaning**
Carnation	Admiration
Violet	Modesty
Daffodil	Prosperity
Daisy	Purity
Lily of valley	Hope
Rose	Romance
Water Lily	Dignity
Gladiolus	Imagination
Morning Glory	Affection
Cosmos	Creativity
Chrysanthemum	Loyalty
Holly	Defense
Default	Not a birth flower

Note: Use command line arguments to get input for the month in integer format. Also observe the methods in flower package are defined as protected. Do the required in order to implement them in Type.java

Sample Input and Output:

```
Command line args 3
The birth flower of entered month is Daffodil and it stands for Prosperity
```

Source Code:

flower/Flower.java

```
package flower;

public abstract class Flower {

        protected abstract String flower(int month);
        protected abstract String meaning(String s);

}
```

q1545/Type.java

```
package q1545;
//import here
import flower.Flower;
public class Type   {
	public String flower(int month) {
		switch(month) {
			case 1:return "Carnation";
			case 2:return "Violet";
			case 3:return "Daffodil";
			case 4:return "Daisy";
			case 5:return "Lily of valley";
			case 6:return "Rose";
			case 7:return "Water Lily";
			case 8:return "Gladiolus";
			case 9:return "Morning Glory";
			case 10:return "Cosmos";
			case 11:return "Chrysanthemum";
			case 12:return "Holly";
			default:return "Enter a valid month";
		}
	}
	public String meaning(String s) {
		switch (s){
			case "Carntion":return"Admiration";
			case "Violet":return "Modesty";
			case "Daffodil":return "Prosperity";
			case "Daisy":return "Purity";
			case "Lily of valley":return "Hope";
			case "Rose":return "Romance";
			case "Water Lily":return "Dignity";
			case "Gladiolus":return "Imagination";
			case "Morning GLory":return "Affection";
			case "Cosmos":return "Creativity";
			case "Chrysanthemum":return "Loyalty";
			case "Holly":return "Defense";
			default:return "Not a birth flower";
		}
	}
	public static void main(String[] args) {
		int month = Integer.parseInt(args[0]);
		Type t = new Type();
		String flower = t.flower(month);
		String meaning = t.meaning(flower);
		System.out.println("The birth flower of entered month is "+flower+" and it
stands for "+meaning);
	}

}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output

```
The birth flower of entered month is Daffodil and it stands for Prosperity
```

Test Case - 2

User Output

```
The birth flower of entered month is Daisy and it stands for Purity
```

Test Case - 3

User Output

```
The birth flower of entered month is Holly and it stands for Defense
```

Test Case - 4

User Output

```
The birth flower of entered month is Violet and it stands for Modesty
```

S.No: 68	Exp. Name: ***User Defined Packages***	**Date: 2023-12-10**

Aim:

Imagine you are a student in a university. At the end of the semester, you receive your total marks for a course out of 100.
You want to calculate your percentage and determine your grade based on the university's grading system.
The Marks interface and the Percentage class in the **p1** package represent the university's grading system.
The percentage method in the Percentage class calculates your percentage based on your total marks.
The grade method determines your grade based on your percentage.
Your task is to create **percentage**(double marks) methods which returns your percentage based on the formula
percentage = (marks/total)*100
Also create a grade method which accepts your percentage and returns the grade accordingly. Refer to below information for
assigning the grade
90-100 = A+ grade
80-90= A grade
70-80 = B+ grade
60-70 = B grade
50-60 = C+ grade
40-50 = C grade
35-40 = D grade
<35 = better luck next time!
In the **Main.java** file accept the marks as input from command line arguments and call the percentage and grade functions
to get the below output

Sample Test Case

```
Command line args 65
65.0
B grade
```

Source Code:

p1/Marks.java

```
//create package p1
package p1;
public interface Marks {
          public double percentage(double marks);
}
```

p1/Percentage.java

```
//create package p1
package p1;
public class Percentage implements Marks{
        public int p;
        public static int total=100;
        public double percentage(double marks) {
                return (marks/total)*100;
        }
        public String grade(double percentage){
                if (percentage>=90){
                        return "A+ grade";
                }else if(percentage>=80){
                        return "A grade";
                }else if (percentage>=70){
                        return "B+ grade";
                }else if (percentage>=60){
                        return "B grade";
                }else if (percentage>=50){
                        return "C+ grade";
                }else if (percentage>=40){
                        return "C grade";
                }else if (percentage>=35){
                        return "D grade";
                }else{
                        return "better luck next time!";
                }
        }
}
```

p2/Main.java

```
//create package p2
package p2;
// import the p1 package
import p1.Percentage;

public class Main {
        public static void main(String[] args) {
                double marks = Double.parseDouble(args[0]);
                Percentage p =new Percentage();

                double per = p.percentage(marks);
                System.out.println(per);

                String gra = p.grade(per);
                System.out.println(gra);
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
67.0
B grade
```

Test Case - 2

User Output

```
92.0
A+ grade
```

Test Case - 3

User Output

```
30.0
better luck next time!
```

Test Case - 4

User Output

```
70.0
B+ grade
```

S.No: 69	Exp. Name: ***Using lang package Math class and util package Scanner class***	**Date: 2023-12-10**

Aim:
Imagine you are developing a software for a construction company. They are building a circular park and need to calculate the area and circumference of the park based on the radius provided. They also want to calculate the diagonal length of a square plot inside the park.
Write a Java program that takes the radius of the circular park as input and calculates the following:

5. The area of the park using the formula πr^2, where**r**is the radius of the park.
6. The circumference of the park using the formula**$2\pi r$**.
7. The diagonal length of the square plot using the formula $\sqrt{2}r$, where**r**is the radius of the park.

Use the**Math**class in Java for these calculations. The program should display the calculated area, circumference, and diagonal length.
Note: Use Double datatype
Sample Input and Output:

```
Enter the radius of the circular park: 40
The area of the park is: 5026.548245743669 square units.
The circumference of the park is: 251.32741228718345 units.
The diagonal length of the square plot is: 56.568542494923804 units.
```

Source Code:

q1547/ParkCalculator.java

```
package q1547;
import java.lang.Math;
import java.util.Scanner;


public class ParkCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
                System.out.print("Enter the radius of the circular park: ");
                double radius = scanner.nextDouble();
                double area = Math.PI*Math.pow(radius,2);
                double circumference = 2*Math.PI*radius;
                double diagonaLength = Math.sqrt(2)*radius;
                System.out.println("The area of the park is: "+area+" square units.");
                System.out.println("The circumference of the park is: "+circumference+"
units.");
                System.out.println("The diagonal length of the square plot is:
"+diagonaLength+" units.");
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the radius of the circular park:
40
The area of the park is: 5026.548245743669 square units.
The circumference of the park is: 251.32741228718345 units.
The diagonal length of the square plot is: 56.568542494923804 units.
```

Test Case - 2

User Output

```
Enter the radius of the circular park:
10
The area of the park is: 314.1592653589793 square units.
The circumference of the park is: 62.83185307179586 units.
The diagonal length of the square plot is: 14.142135623730951 units.
```

Test Case - 3

User Output

```
Enter the radius of the circular park:
6
The area of the park is: 113.09733552923255 square units.
The circumference of the park is: 37.69911184307752 units.
The diagonal length of the square plot is: 8.485281374238571 units.
```

Test Case - 4

User Output

```
Enter the radius of the circular park:
66
The area of the park is: 13684.77759903714 square units.
The circumference of the park is: 414.6902302738527 units.
The diagonal length of the square plot is: 93.33809511662427 units.
```

S.No: 70	Exp. Name: ***Built In Exception***	Date: 2023-12-10

Aim:
In the given program use exception handling to maintain the normal flow of program
Note: use try catch statements
Source Code:

q1361/Main.java

```
package q1361;
public class Main
{
	public static void main(String[] args) {

		try{
		    int[] a = {5,0,2,10,20};
		    int b = Integer.parseInt(args[0]);

		    for(int i = 0; i < a.length; i++){
		        int c = b / a[i];
		        System.out.println("Result : " +c);
		    }
		}catch (NumberFormatException e){
			System.out.println("Invalid input. Please provide a valid number as
a command line argument.");
		}
		catch (ArithmeticException e){
			System.out.println("Can't divide a number by 0");
		}

	}
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Result : 100
Can't divide a number by 0
```

Test Case - 2

User Output

```
Result : 200
Can't divide a number by 0
```

Test Case - 3

User Output

```
Result : 2010
```

```
Can't divide a number by 0
```

Test Case - 4

User Output

```
Result : 469
```

```
Can't divide a number by 0
```

S.No: 71	Exp. Name: ***Built In Exception***	Date: 2023-12-10

Aim:
Use Exception Handling in order to get the below output:

Sample Test Case 1:
```
Command line args 3
The element present at 3 is 40
```
Sample Test Case 2:
```
Command line args 6
The mentioned index is out of bounds
```

Source Code:

q1362/Main.java

```
package q1362;
public class Main {
    public static void main(String[] args) {
        int[] arr = {10,20,30,40,50,60};
        int index = Integer.parseInt(args[0]);
        try{
                        int element = arr[index];
                        System.out.println("The element present at "+index+" is "+element);
                }catch (ArrayIndexOutOfBoundsException e){
                        System.out.println("The mentioned index is out of bounds");
                }catch (NumberFormatException e){
                                        System.out.println("Invalid input. Please provide a
valid number as a command line argument");
                }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
The element present at 3 is 40
```

Test Case - 2

User Output

```
The mentioned index is out of bounds
```

Test Case - 3

User Output

```
The element present at 1 is 20
```

Test Case - 4

User Output

```
The element present at 0 is 10
```

S.No: 72	Exp. Name: ***User Defined Exception***	Date: 2023-12-13

Aim:

Employee Search System: Consider you need to create a employee search system such that it returns an Exception when they enter a number less than 0 or greater than 999.

Note : Use Custom defined Exceptions

Sample test output 1:

```
Command line args 1000
Exception caught
Invalid Employee ID
```

Sample test output 2:

```
Command line args 23
Valid Employee ID
```

Source Code:

q1364/SampleEmp.java

```
package q1364;
class EmployeeException extends Exception
{
        public EmployeeException(String s)
        {
                super(s);
        }
}
class SampleEmp
{
        public void checkId(int id) throws EmployeeException{
                if (id <=0 || id>=999){
                        throw new EmployeeException("Exception caught\nInvalid Employee 
ID");
                }
                System.out.println("Valid Employee ID");
        }
        public static void main(String args[])
        {
                SampleEmp emp = new SampleEmp();
                try{
                        int in1 = Integer.parseInt(args[0]);
                        emp.checkId(in1);
                }catch(Exception e){
                        System.out.println("Exception caught\nInvalid Employee ID");
                }

        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Exception caught
Invalid Employee ID
```

Test Case - 2

User Output

```
Valid Employee ID
```

Test Case - 3

User Output

```
Exception caught
Invalid Employee ID
```

Test Case - 4

User Output

```
Valid Employee ID
```

S.No: 73	Exp. Name: ***User Defined Exception(try-catch-finally)***	**Date: 2023-12-10**

Aim:
Gift Coupon System : Consider you need build a program which takes the user's coupon code and applies it on the bill.
Note : Use try-catch-finally blocks to get the following output
Sample Test case 1:

```
Command line args COUPON123
Coupon code applied successfully
Thank you for shopping with us!!
```

Sample Test case 2:

```
Command line args CODE55
Failed to apply coupon: Invalid Coupon Code
Thank you for shopping with us!!
```

Source Code:

q1367/GiftCoupon.java

```java
package q1367;
public class GiftCoupon {
    public static void main(String[] args) {
        String code = args[0];
        //Write code here...
        try{
                        checkCoupon(code);
                        System.out.println("Coupon code applied successfully");
                }catch (Exception e){
                        System.out.println("Failed to apply coupon: "+e.getMessage());
                }finally{
                        System.out.println("Thank you for shopping with us!!");
                }
    }
    public static void checkCoupon(String code) throws Exception {
        if (!"COUPON123".equals(code)) {
            throw new Exception("Invalid Coupon Code");
        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Coupon code applied successfully
Thank you for shopping with us!!
```

Test Case - 2

User Output

```
Failed to apply coupon: Invalid Coupon Code
```

```
Thank you for shopping with us!!
```

Test Case - 3

User Output

```
Failed to apply coupon: Invalid Coupon Code
Thank you for shopping with us!!
```

Test Case - 4

User Output

```
Failed to apply coupon: Invalid Coupon Code
Thank you for shopping with us!!
```

S.No: 74	Exp. Name: ***User Defined Exception***	**Date: 2023-12-10**

Aim:

Office Device Availability System: Consider you need to create a program which checks the availability of printers in the office and throws an exception if it is not available.
Create the method *use* in *Device* class where it prints *Available* for printer1 and printer3 or throws *DeviceFailureException* for other cases.

Sample Test Case 1:

```
Command line args printer1
Available
```

Sample Test Case 2:

```
Command line args printer2
Device failure: Not Available
```

Source Code:

q1368/Main.java

```
package q1368;
public class Main {
    public static void main(String[] args) {
        Device device = new Device();
        String printer = args[0];
        try {
            device.use(printer);
        } catch (DeviceFailureException e) {
            System.out.println("Device failure: " + e.getMessage());
        }
    }
}

class Device {
    //Write Code here...
    public void use(String printer) throws DeviceFailureException{
                if ("printer1".equals(printer) || "printer3".equals(printer)){
                        System.out.println("Available");
                }else{
                        throw new DeviceFailureException("Not Available");
                }
        }

}

class DeviceFailureException extends Exception {
    DeviceFailureException(String message) {
        super(message);
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Device failure: Not Available
```

Test Case - 2

User Output

```
Device failure: Not Available
```

Test Case - 3

User Output

```
Device failure: Not Available
```

Test Case - 4

User Output

```
Available
```

S.No: 75	Exp. Name: ***User Defined Exception***	Date: 2023-12-10

Aim:
Imagine you are developing an online shopping cart system, and you are responsible for implementing a feature that checks if the quantity of a product added to the cart exceeds the available stock. To handle this situation, you decide to create a custom exception called BulkStockException.

1. Exception Implementation: Write a simple Java class representing the BulkStockException that extends the Exception class. Include a constructor that accepts a product name as a parameter and generates an error message indicating that the product cannot be ordered in quantities of more than 10 units.
2. Shopping Cart Functionality: Create a snippet of code for a ShoppingCart class that includes a method called addToCart. This method should take parameters such as the product name and quantity. If the quantity requested exceeds 10 units, throw the BulkStockException with the appropriate error message else give the message to the user that the product has been added to the cart.
3. User Input: In the main class, take the product name and quantity from the command line arguments. Create an instance of ShoppingCart class and call the method addToCart with the help of the instance created

Sample output :1
```
Command line args pen 9
9 units of pen added to the cart.
```
Sample output :2
```
Command line args laptop 11
Error: The laptop cannot be ordered in quantities of more than 10 units.
```
Source Code:

```
q1531/UserDefinedException.java
```

```
package q1531;
class BulkStockException extends Exception{
        public BulkStockException(String productName){
                super("The "+productName+" cannot be ordered in quantities of more than 10
units.");
        }
}
class ShoppingCart {
    // write your logic here
    public void addToCart(String productName, int quantity) throws BulkStockException{
                if (quantity>10){
                        throw new BulkStockException(productName);
                }else{
                        System.out.println(quantity+" units of "+productName+" added to the
cart.");
                }
        }
}
public class UserDefinedException {
    public static void main(String[] args) {
        String product = args[0];
        int quantity = Integer.parseInt(args[1]);
        ShoppingCart shoppingCart = new ShoppingCart();
        try {
            shoppingCart.addToCart(product,quantity);
        } catch (BulkStockException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
9 units of pen added to the cart.
```

Test Case - 2

User Output

```
Error: The laptop cannot be ordered in quantities of more than 10 units.
```

Test Case - 3

User Output

```
2 units of laptop added to the cart.
```

Test Case - 4

User Output

```
7 units of book added to the cart.
```

S.No: 76	Exp. Name: ***Built in Exceptions (multiple catch)***	Date: 2023-12-10

Aim:

Imagine you are a software developer working on an interactive quiz application for students. As part of the application, you've implemented a feature that allows students to perform arithmetic operations. The students are prompted to enter two integer values, 'a' and 'b', and the application calculates and displays the result of the division operation (a/b). However, you've encountered a few challenges in handling exceptions, and you want the students to understand the importance of providing valid inputs.

Valid Input Scenario:
- Enter two integer values for 'a' and 'b' to successfully perform the division operation.

Division by Zero Scenario:
- Attempt to enter values where 'b' is zero and write how the application handles the ArithmeticException.

Invalid Input Scenario:
- Enter non-integer values (such as characters or decimals) for 'a' and 'b' and write how the application handles the InputMismatchException.

Sample Out Put : 1

```
Enter the value of 'a': 50
Enter the value of 'b': 20
Result of a/b: 2
```

Sample Out Put : 2

```
Enter the value of 'a': 50
Enter the value of 'b': 0
Arithmetic Exception: / by zero
```

Sample Out Put : 3

```
Enter the value of 'a': code
Input Mismatch Exception: Please enter valid integer values.
```

Source Code:

```
q1537/ExceptionHandlingExample.java
```

```
package q1537;
import java.util.InputMismatchException;
import java.util.Scanner;

public class ExceptionHandlingExample {
        public static void main(String[] args) {
                // Write required scanner class and variables a, and b
                Scanner s = new Scanner(System.in);
                int a=0, b=0;
        try {

                // write your code here
                System.out.print("Enter the value of 'a': ");
                        a = s.nextInt();

                        System.out.print("Enter the value of 'b': ");
                        b = s.nextInt();

                        int result=divide(a,b);
                        System.out.println("Result of a/b: "+result);
        } catch (InputMismatchException e) {
                // write your code here
                System.out.println("Input Mismatch Exception: Please enter valid integer
values.");
        } catch (ArithmeticException e) {
                // write your code here
                System.out.println("Arithmetic Exception: "+e.getMessage());
        }finally{
                        s.close();
                }

    }

    public static int divide(int a,int b) {
        // write your code here
                if (b==0){
                        throw new ArithmeticException("/ by zero");
                }
                return a/b;

    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the value of 'a':
0
Enter the value of 'b':
56
Result of a/b: 0

Test Case - 2

User Output

```
Enter the value of 'a':
36
Enter the value of 'b':
7
Result of a/b: 5
```

Test Case - 3

User Output

```
Enter the value of 'a':
code tantra
Input Mismatch Exception: Please enter valid integer values.
```

Test Case - 4

User Output

```
Enter the value of 'a':
100
Enter the value of 'b':
0
Arithmetic Exception: / by zero
```

S.No: 77	Exp. Name: ***Built In Exception***	**Date: 2023-12-10**

Aim:

Imagine you are developing a software application for a bank. The application needs to process transactions which are initially entered as strings. The*ParseIntExample*class represents a module in this application that attempts to parse these strings into integers for further processing.
Create a Java program that attempts to convert a string to an integer using Integer.parseInt() and handles a NumberFormatException if the string is not a valid integer.

Sample Output:1

```
Parsed integer: 123456789
```

Sample Output:2

```
Enter an integer as a string: code
Error: The input is not a valid integer.
```

Source Code:

q1539/ParseIntExample.java

```
package q1539;
// import scanner class
import java.util.Scanner;
public class ParseIntExample {
    public static void main(String[] args) {
            // Input a string that represents an integer
                Scanner s = new Scanner(System.in);
        System.out.print("Enter an integer as a string: ");
                String input = s.nextLine();

            // Attempt to parse the string to an integer
        try{
            // Display the parsed integer
                        int number = Integer.parseInt(input);
            System.out.println("Parsed integer: " + number);
                }catch (NumberFormatException e){
            // Handle the NumberFormatException
                System.out.println("Error: The input is not a valid integer.");
            // Print the message ("Error: The input is not a valid integer.");
                }finally{
                        s.close();
                }
    }
}
// Strings required to print desired output are given in above partial code
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter an integer as a string:
0123456789
Parsed integer: 123456789
```

Test Case - 2

User Output

```
Enter an integer as a string:
36.5
Error: The input is not a valid integer.
```

Test Case - 3

User Output

```
Enter an integer as a string:
code
Error: The input is not a valid integer.
```

Test Case - 4

User Output

```
Enter an integer as a string:
1254
Parsed integer: 1254
```

S.No: 78	Exp. Name: ***User Defined Package***	**Date: 2023-12-10**

Aim:
Imagine you are developing a software for a library. The software has a feature where it takes the ISBN number of a book as input from the user and fetches the details of the book from an online database. **The ISBN number is a string of 10 or 13 digits.**

Write a Java program that takes the **ISBN** number as input and checks if it's valid. The program should throw and handle the **IllegalArgumentException**: If the **ISBN is not 10 or 13 digits long**. If the ISBN is valid, the program should display a message saying that the ISBN is valid.

Sample Test Case-1
```
Enter the ISBN number: A1202345E43A1202345E4
The ISBN is valid.
```

Sample Test Case-2
```
Enter the ISBN number: A12023E345E32
The ISBN is valid.
```

Sample Test Case-3
```
Enter the ISBN number: A12023E23
Error: length of ISBN must be 10 or 13 digits long.
```

Source Code:

q1540/Main.java

```
package q1540;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
                System.out.print("Enter the ISBN number: ");
                String isbn = scanner.nextLine();

                try{
                        validateISBN(isbn);
                }catch (IllegalArgumentException e){
                        System.out.println("Error: "+e.getMessage());
                }finally{
                        scanner.close();
                }
        }
        public static void validateISBN(String isbn){
                if (isbn.length()==10 || isbn.length()==13){
                        System.out.println("The ISBN is valid.");
                }else{
                        throw new IllegalArgumentException("length of ISBN must be 10 or 13
digits long.");
                }
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the ISBN number:
A12023607E
The ISBN is valid.
```

Test Case - 2

User Output

```
Enter the ISBN number:
A120023876E12
The ISBN is valid.
```

Test Case - 3

User Output

```
Enter the ISBN number:
A202345A12
The ISBN is valid.
```

Test Case - 4

User Output

```
Enter the ISBN number:
A12023A2
Error: length of ISBN must be 10 or 13 digits long.
```

S.No: 79	Exp. Name: ***User Defined Exception***	**Date: 2023-12-10**

Aim:

User Registration System
Imagine you are developing a user registration system for a website.
The User class represents a user in this system.
When creating a new **user**, the system needs to validate the user's age to ensure it is within acceptable limits.
The **InvalidAgeException** class is a custom exception that is thrown when the user's age is not valid.
The User constructor throws this exception if the **age is less than or equal 0 or greater than 100**, which are considered invalid values.
The Main class represents the part of the system where new users are registered.
It takes the user's age as input, attempts to create a new **User** object with this **age**,
and catches the **InvalidAgeException** if it is thrown. The error message from the exception is then printed to the console.

Sample Test Case-1
```
Command line args 65
Valid age.
```
Sample Test Case-2
```
Command line args 0
Age cannot be negative.
```
Sample Test Case-3
```
Command line args -09
Age cannot be negative.
```

Source Code:

```
q1548/Main.java
```

```
package q1548;

class InvalidAgeException extends Exception {
        public InvalidAgeException(String message){
                super(message);
        }
}
class User {
        private int age;
        public User(int age) throws InvalidAgeException{
                if (age<=0){
                        throw new InvalidAgeException("Age cannot be negative.");
                }else if (age>100){
                        throw new InvalidAgeException("Age is too high.");
                }
                this.age = age;
        }
        public int getAge(){
                return age;
        }
                // }
    // }
}
class Main {
         public static void main(String[] args) {
                try{
                        int age = Integer.parseInt(args[0]);
                        registerUser(age);
                }catch (NumberFormatException e){
                        System.out.println("Invalid input: Please enter a valid integer for
age.");
                }catch (InvalidAgeException e){
                        System.out.println(e.getMessage());
                }
        }
        public static void registerUser(int age) throws InvalidAgeException{
                User u = new User(age);
                System.out.println("Valid age.");
        }
                // catch (InvalidAgeException ex) {
   //          System.out.println(ex.getMessage());
   //      }
    // }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Valid age.

Test Case - 2

User Output

```
Age cannot be negative.
```

Test Case - 3

User Output

```
Age cannot be negative.
```

Test Case - 4

User Output

```
Age is too high.
```

S.No: 80	Exp. Name: ***Built in Exceptions***	Date: 2023-12-10

Aim:

Student Score Recording System:

Imagine you are developing a software for a school. The software has a feature where it takes the name of a student and the student's score as input from the user. The score is a string that should be convertible to an integer between 0 and 100.

Write a Java program that takes the name and score as input and checks if they're valid. The program should throw and handle the following exceptions:

8. NullPointer Exception: If the name is null.
9. NumberFormat Exception: If the score is not a valid integer.
10. IllegalArgument Exception: If the score is not between 0 and 100.

The program should display an appropriate error message for each exception. If the name and score are valid, the program should display a message saying that the score has been successfully recorded.

Sample Input and Output 1:

```
Enter the student's name: Aditya
Enter the student's score: 23e
Error: The score is not a valid integer.
```

Sample Input and Output 2:

```
Enter the student's name: Rahul
Enter the student's score: 121
Error: Score must be between 0 and 100.
```

Sample Input and Output 3:

```
Enter the student's name: Riya
Enter the student's score: 66
The score has been successfully recorded.
```

Source Code:

```
q1549/StudentScoreRecorder.java
```

```
package q1549;
import java.util.Scanner;

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

        System.out.print("Enter the student's name: ");
        String name = scanner.nextLine();

        System.out.print("Enter the student's score: ");
        String scoreStr = scanner.nextLine();

        //write code here
                try{
                        recordScore(name, scoreStr);
                        System.out.println("The score has been successfully recorded.");
                }catch (NullPointerException e){
                        System.out.println("Error: The name cannot be null.");
                }catch (NumberFormatException e){
                        System.out.println("Error: The score is not a valid integer.");
                }catch (IllegalArgumentException e){
                        System.out.println("Error: "+e.getMessage());
                }finally{
                        scanner.close();
                }
    }
        public static void recordScore(String name, String scoreStr){
                if (name==null){
                        throw new NullPointerException("The name cannot be null.");
                }
                int score;
                try{
                        score=Integer.parseInt(scoreStr);
                }catch (NumberFormatException e){
                        throw new NumberFormatException("The score is not a valid
integer.");
                }
                if (score<0 || score>100){
                        throw new IllegalArgumentException("Score must be between 0 and
100.");
                }
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the student's name:
```

Aditya

```
Enter the student's score:
```

```
23e
Error: The score is not a valid integer.
```

Test Case - 2

User Output

```
Enter the student's name:
Rahul
Enter the student's score:
121
Error: Score must be between 0 and 100.
```

Test Case - 3

User Output

```
Enter the student's name:
Riya
Enter the student's score:
66
The score has been successfully recorded.
```

Test Case - 4

User Output

```
Enter the student's name:
Kiran
Enter the student's score:
99
The score has been successfully recorded.
```

S.No: 81	Exp. Name: ***ATM Transaction System:***	**Date: 2023-12-10**

Aim:

ATM Transaction System:
Consider a scenario where you are creating a software for an ATM machine.
The Main class represents the part of software that handles the transactions
When a customer needs to withdraw money, they need to give amount as input and the software needs to validate that the entered amount is within acceptable limits.
The **InvalidAmountException** class is a custom exception that is thrown when the entered amount is greater than or equal to the fixed balance of **10000** or if the amount is **less than or equal to 0.**

Tasks to done:
Create a user defined **InvalidAmountException** class which returns the message accordingly.
In the **Main** class use try catch blocks to catch the exceptions and print the remaining balance.

Sample Test Case-1

```
Enter the amount to be withdrawn: 6700
The transaction is done
Remaining balance is 3300
```

Sample Test Case-2

```
Enter the amount to be withdrawn: 10500
Error: Not Sufficient Balance
```

Sample Test Case-3

```
Enter the amount to be withdrawn: -4500
Error: Amount cannot be negative
```

Source Code:

```
q1550/Main.java
```

```
package q1550;
import java.util.Scanner;

//write the code here
        class InvalidAmountException extends Exception{
                public InvalidAmountException(String message){
                        super(message);
                }
        }



public class Main {
    public static void main(String[] args) {
        int fixedAmount = 10000;
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the amount to be withdrawn: ");
        int amount = scanner.nextInt();
        try{
                        validateAmount(amount, fixedAmount);
                        System.out.println("The transaction is done");
                        System.out.println("Remaining balance is "+(fixedAmount-amount));
                }catch (InvalidAmountException e){
                        System.out.println("Error: "+e.getMessage());
                }finally{
                        scanner.close();
                }
        }
        public static void validateAmount(int amount, int fixedAmount) throws
InvalidAmountException{
                if (amount<=0){
                        throw new InvalidAmountException("Amount cannot be negative");
                }else if (amount>=fixedAmount){
                        throw new InvalidAmountException("Not Sufficient Balance");
                }
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the amount to be withdrawn:
7600
The transaction is done
Remaining balance is 2400
```

Test Case - 2

User Output

```
Enter the amount to be withdrawn:
-459
Error: Amount cannot be negative
```

Test Case - 3

User Output

```
Enter the amount to be withdrawn:
1043
The transaction is done
Remaining balance is 8957
```

Test Case - 4

User Output

```
Enter the amount to be withdrawn:
10650
Error: Not Sufficient Balance
```

S.No: 82	Exp. Name: ***SingleThread***	**Date: 2023-12-15**

Aim:

This **UserThread** class extends **Thread** and overrides the **run()** method. Within the **run()** method, it creates a **Scanner** object to read input from the user. It prompts the user to enter something, reads the input, and then prints out what was entered.
When you run the program, it starts the thread, allowing the user to input something while the thread is running in the background.

Sample Test Case:

```
Thread Started Running...
Enter something: vinayaka
You entered: vinayaka
```

Source Code:

```
q1551/UserThread.java
```

```
package q1551;
import java.util.Scanner;

public class UserThread extends Thread {
    public void run() {
        System.out.println("Thread Started Running...");

        // Create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter something
        System.out.print("Enter something: ");

        // Read the input
        String userInput = scanner.nextLine();

        // Print out what was entered
        System.out.println("You entered: " + userInput);

        // Close the scanner to prevent resource leaks
        scanner.close();
    }

    public static void main(String[] args) {
        // Create an instance of UserThread
        UserThread userThread = new UserThread();

        // Start the thread
        userThread.start();

        // You can add other tasks to be performed in the main thread while UserThread is
running
        // For example, you might want to perform some background tasks here.

        // Wait for the UserThread to finish before exiting the program
        try {
            userThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Thread Started Running...
Enter something:
codetantra
```

```
You entered: codetantra
```

Test Case - 2

User Output

```
Thread Started Running...
Enter something:
hyderabad
You entered: hyderabad
```

Test Case - 3

User Output

```
Thread Started Running...
Enter something:
ganga
You entered: ganga
```

Test Case - 4

User Output

```
Thread Started Running...
Enter something:
yamuna
You entered: yamuna
```

S.No: 83	Exp. Name: ***Thread Creation using Thread Class***	Date: 2023-12-15

Aim:

You are a software engineer working on a project that requires the use of multithreading in Java. Your task is to create a simple program that creates and starts a new thread. The thread should print a message indicating that it is running.
The program should include a class named **SimpleThread** that extends the Thread class. It should also override the run method to print a message indicating that the thread is running.
The program should also include a main method in a class named MultipleThreads. The main method should create an instance of **SimpleThread** with a name passed as a command-line argument and start the thread.

Write the complete Java program based on the above requirements

Sample Test Case:

```
Command line args chennai
Running chennai
```

Source Code:

`q1552/MultipleThreads.java`

```
package q1552;
class SimpleThread extends Thread {
    private String threadName;
        //write code here..
    public SimpleThread(String threadName) {
        this.threadName = threadName;
    }

    // Override the run method to print a message indicating that the thread is running
    @Override
    public void run() {
        System.out.println("Running " + threadName);
        }
}
public class MultipleThreads {
    public static void main(String[] args) {

        //write code here..
        // Get the thread name from the command-line argument
        String threadName = args[0];

        // Create an instance of SimpleThread with the provided thread name
        SimpleThread simpleThread = new SimpleThread(threadName);

        // Start the thread
        simpleThread.start();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Running Ganga
```

Test Case - 2

User Output

```
Running Yamuna
```

Test Case - 3

User Output

```
Running Kaveri
```

Test Case - 4

User Output

```
Running Hyderabad
```

S.No: 84	Exp. Name: ***Thread Creation Using Runnable Interface***	**Date: 2023-12-15**

Aim:
Your task is to create a simple program that calculates the factorial of a number using a separate thread. The thread should print the factorial of the number.
The program should include a class named**ThreadImplement**that implements the**Runnable**interface.
TheThreadImplementclass should have a constructor that accepts an integer argument representing the number for which the factorial is to be calculated. It should also implement the**run**method to calculate the factorial of the number and print it.
The**main**method should create an instance ofThreadImplementwith a number passed as a command-line argument, wrap it in a**Thread**object, and start the thread.
Complete the java program based on above requirements.
Sample Input and Output:

```
Command line args 5
120
```

Source Code:

```
q1556/Main.java
```

```
package q1556;
class ThreadImplement implements Runnable {
    private int number;

    // Constructor to accept the number for which the factorial is to be calculated
    public ThreadImplement(int number) {
        this.number = number;
    }

    // Override the run method to calculate the factorial and print it
    @Override
    public void run() {
        long factorial = calculateFactorial(number);
        System.out.println(factorial);
    }

    // Helper method to calculate the factorial
    private long calculateFactorial(int n) {
        if (n == 0 || n == 1) {
            return 1;
        } else {
            return n * calculateFactorial(n - 1);
        }
        }
}

public class Main {
        public static void main(String[] args)
        {
                //write code here
                // Get the number from the command-line argument
        int number = Integer.parseInt(args[0]);

        // Create an instance of ThreadImplement with the provided number
        ThreadImplement threadImplement = new ThreadImplement(number);

        // Wrap it in a Thread object
        Thread thread = new Thread(threadImplement);

        // Start the thread
        thread.start();
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
120
```

Test Case - 2

User Output

```
720
```

Test Case - 3

User Output

```
24
```

Test Case - 4

User Output

```
479001600
```

S.No: 85	Exp. Name: ***Multiple Threads performing Multiple Operations***	**Date: 2023-12-15**

Aim:
You are a software engineer working on a project that requires the use of multithreading in Java. Your task is to create a program that calculates the Fibonacci series and prints a reverse series from 10 to 1. Each task should be performed by a separate thread.
The program should include two classes:**Fibonacci**and**Reverse**, each extending the**Thread**class.
The**Fibonacci**class should override the**run**method to calculate and print the Fibonacci series up to a limit entered by the user. The**Reverse**class should override the**run**method to print a reverse series from 10 to 1.
The**main**method should create an instance of each class, start each thread, and ensure that the**Reverse**thread does not start until the**Fibonacci**thread has finished its execution.
Complete the Java program based on the above requirements.
Sample Input and Output:

```
Enter the Limit for fibonacci: 8
Fibonacci series:
0 1 1 2 3 5 8 13
Reverse is:
10 9 8 7 6 5 4 3 2 1
```

Source Code:

```
q1559/Main.java
```

```
package q1559;
import java.util.*;
class Fibonacci extends Thread
{
     public void run()
     {
          try
          {
               int a=0, b=1, c=0;
               System.out.print("Enter the Limit for fibonacci: ");
               Scanner sc = new Scanner(System.in);
               int n = sc.nextInt();
               System.out.println("================================");
               System.out.println("Fibonacci series:");
               //write code here....
               for (int i = 0; i < n; i++) {
                System.out.print(a + " ");
                c = a + b;
                a = b;
                b = c;
            }
          }
          catch (Exception ex)
          {
               ex.printStackTrace();
          }
     }
}
class Reverse extends Thread
{
     public void run()
     {
          try
          {
               // System.out.println("\n================================");
               // System.out.println("\nReverse is: ");
               // System.out.println("================================");
              //write code here..
                         System.out.println("\n================================");
                         System.out.println("Reverse is: ");
                         System.out.println("================================");
              for (int i = 10; i >= 1; i--) {
                System.out.print(i + "  ");
            }


    System.out.println("\n================================");

          }
          catch (Exception ex)
          {
               ex.printStackTrace();
          }
     }
}
```

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

        try
        {
            Fibonacci fib = new Fibonacci();
            fib.start();
            fib.sleep(4000);
            Reverse rev = new Reverse();
            rev.start();
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the Limit for fibonacci:
8
Fibonacci series:
0 1 1 2 3 5 8 13
Reverse is:
10 9 8 7 6 5 4 3 2 1
```

Test Case - 2

User Output

```
Enter the Limit for fibonacci:
10
Fibonacci series:
0 1 1 2 3 5 8 13 21 34
Reverse is:
10 9 8 7 6 5 4 3 2 1
```

S.No: 86	Exp. Name: ***Task Manager***	**Date: 2023-12-15**

Aim:

This Java program simulates a task manager where employees work on different tasks concurrently. Here's an illustrative scenario for this program.
Each task name entered triggers the creation of a separate thread representing that task. The threads run concurrently, displaying the progress and completion of each task after the simulated work duration.
This program showcases a basic example of task management where threads represent individual tasks, allowing for concurrent execution of those tasks by simulating work with sleep intervals.

sample Test Case:

```
Enter number of tasks:
2
Enter task 1 name:
docfile
Enter task 2 name:
docfile
Employee is working on: docfile
Employee is working on: docfile
Employee finished: docfile
Employee finished: docfile
```

Source Code:

```
q1570/TaskManager.java
```

```java
package q1570;
import java.util.Scanner;
class Task implements Runnable {
    private String taskName;
    public Task(String taskName) {
        this.taskName = taskName;
        }
        public void run() {
        try {
            System.out.println("Employee is working on: " + taskName);
            // Simulate work by sleeping for a random duration (e.g., between 1 and 5
seconds)
            Thread.sleep((long) (Math.random() * 4000) + 1000);
            System.out.println("Employee finished: " + taskName);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

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

        System.out.println("Enter number of tasks:");
        int numOfTasks = scanner.nextInt();
        scanner.nextLine();
                for (int i = 1; i <= numOfTasks; i++) {
            System.out.println("Enter task " + i + " name:");
            String taskName = scanner.nextLine();

            // Create an instance of Task for each task name and start a new thread for each
task
            Task task = new Task(taskName);
            Thread thread = new Thread(task);
            thread.start();
        }

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

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter number of tasks:
1
Enter task 1 name:
docfile
Employee is working on: docfile
Employee finished: docfile
```

S.No: 87	Exp. Name: ***Bank Account (deposit, withdraw, checkBalance)***	**Date: 2023-12-15**

Aim:

This program simulates a simple banking system with multiple customers interacting with a bank account. Here's a breakdown of the scenario this program represents:
Bank Account The program starts by initializing a bank account with an initial balance of $500.

Customer Interactions:

- **Input Gathering:** It prompts the user to enter the number of customers interacting with the bank.Customer **Operations**: For each customer, it prompts for an operation **(deposit/withdraw/checkBalance)**.
- **Thread Creation:** It creates a thread for each customer operation.
- **Customer Actions**:**Deposit**: If the customer chooses to deposit, they deposit $100 into the **account.Withdraw**
- If the customer chooses to **withdraw**, they withdraw $50 from the account if sufficient funds are available; otherwise, it displays an **"Insufficient funds"**
- **message.Check Balance:** If the customer checks the balance, it displays the current balance of the account.
-

Thread Handling: Once all customers and their respective operations are entered, the program starts each thread and waits for all threads to finish using **join()**

Sample Test Case:

```
Enter the number of customers:
1
Enter operation (deposit/withdraw/checkBalance) for customer 1:
checkBalance
Thread-0 - Current Balance: 500.0
```

Source Code:

```
q1573/Bank.java
```

```java
package q1573;
import java.util.Scanner;

class BankAccount {
    private double balance;

    public BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }
    // write code here
    public synchronized void deposit(double amount) {
        balance += amount;
        System.out.println(Thread.currentThread().getName() + " deposited $" + amount);
    }

    public synchronized void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
            System.out.println(Thread.currentThread().getName() + " withdrew $" + amount);
        } else {
            System.out.println(Thread.currentThread().getName() + " - Insufficient funds");
        }
    }

    public synchronized void checkBalance() {
        System.out.println(Thread.currentThread().getName() + " - Current Balance: " +
balance);
    }
}
 class Customer implements Runnable {
    private BankAccount account;
    private String operation;

    public Customer(BankAccount account, String operation) {
        this.account = account;
        this.operation = operation;
        }
        public void run() {
        if (operation.equals("deposit")) {
            account.deposit(100);
        } else if (operation.equals("withdraw")) {
            account.withdraw(50);
        } else if (operation.equals("checkBalance")) {
            account.checkBalance();
        }
    }
 }
public class Bank {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
    // }
        System.out.println("Enter the number of customers:");
        int numOfCustomers = scanner.nextInt();
        scanner.nextLine();
                BankAccount bankAccount = new BankAccount(500.0);
            System.out.println("Enter operation (deposit/withdraw/checkBalance) for customer
" + i + ":");
            String operation = scanner.nextLine();

            Customer customer = new Customer(bankAccount, operation);
            Thread thread = new Thread(customer);
            thread.start();
            try {
                thread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        scanner.close();
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of customers:
1
Enter operation (deposit/withdraw/checkBalance) for customer 1:
deposit
Thread-0 deposited $100.0
```

Test Case - 2

User Output

```
Enter the number of customers:
1
Enter operation (deposit/withdraw/checkBalance) for customer 1:
withdraw
Thread-0 withdrew $50.0
```

S.No: 88	Exp. Name: ***Inter Thread Communication***	Date: 2023-12-15

Aim:

Banking System:
Consider a banking system where customers can deposit and withdraw money from their accounts.
The**Customer**class represents a customer’s account with an initial balance of 10,000.
Thewithdrawalanddepositmethods are synchronized to ensure that only one thread can access the account at a time.
In the**Main**class, two threads are created: one for withdrawal and one for deposit. The withdrawal thread attempts to withdraw from the account, and the deposit thread attempts to deposit into the account. The amount for withdrawal and deposit is given using command line arguments.

Sample Input and Output 1:

```
Command line args 5000 6000
going to withdraw...
withdraw completed...
going to deposit...
deposit completed...
```

Sample Input and Output 2:

```
Command line args 15000 6000
going to withdraw...
Less balance; waiting for deposit...
going to deposit...
deposit completed...
withdraw completed...
```

Source Code:

```
q1564/Main.java
```

```java
package q1564;
class Customer{
        int amount=10000;

        synchronized void withdraw(int amount){
                System.out.println("going to withdraw...");

                if(this.amount<amount){
                System.out.println("Less balance; waiting for deposit...");
                try{wait();}catch(Exception e){}
                }
                this.amount-=amount;
                System.out.println("withdraw completed...");
        }

        synchronized void deposit(int amount){
                //write code for deposit method here..
                System.out.println("going to deposit...");
        this.amount += amount;
        System.out.println("deposit completed... ");
                //use notify()
                notify();
        }
}
public class Main{
        public static void main(String args[]){

        //write code here..
        //Create a Customer instance
        Customer customer = new Customer();

        //create a 2 Thread instances and implement run method
        //which call the withdraw and deposit methods respectively
        Thread withdrawalThread = new Thread(() -> {
            // Withdraw an amount provided as a command line argument
            int withdrawAmount = Integer.parseInt(args[0]);
            customer.withdraw(withdrawAmount);
        });

        Thread depositThread = new Thread(() -> {
            // Deposit an amount provided as a command line argument
            int depositAmount = Integer.parseInt(args[1]);
            customer.deposit(depositAmount);
        });

        // Start both threads
        withdrawalThread.start();
        depositThread.start();

        }

}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
going to withdraw...
withdraw completed...
going to deposit...
deposit completed...
```

Test Case - 2

User Output

```
going to withdraw...
Less balance; waiting for deposit...
going to deposit...
deposit completed...
withdraw completed...
```

Test Case - 3

User Output

```
going to withdraw...
withdraw completed...
going to deposit...
deposit completed...
```

Test Case - 4

User Output

```
going to withdraw...
Less balance; waiting for deposit...
going to deposit...
deposit completed...
withdraw completed...
```

S.No: 89	Exp. Name: ***Inter Thread Communication***	**Date: 2023-12-15**

Aim:

Classroom Learning Scenario:

Imagine a classroom where students are learning about numbers. The teacher has devised a fun activity to help students understand the concept of odd and even numbers. The class is divided into two groups: one group is responsible for odd numbers and the other for even numbers.

The teacher has a**Printer**object that can print numbers. However, it can only print one number at a time, and it must alternate between printing odd and even numbers. The**Printer**has two methods:**printOdd**and**printEven**. Each method checks a condition (**isOddPrinted**) to determine whether it’s their turn to print. If it’s not their turn, they wait. Once a number is printed, the condition is updated, and the other group is notified that it’s their turn.

In the**Main**class, two threads are created: one for the odd numbers group and one for the even numbers group. Each thread runs a loop up to a specified count (passed as a command-line argument), printing either odd or even numbers. The threads coordinate with each other using the**Printer**object to ensure that the numbers are printed in the correct order.

Complete the given code in order to achieve the above scenario

Sample Input and Output:

```
Command line args 5
Odd:1
Even:2
Odd:3
Even:4
Odd:5
```

Source Code:

```
q1567/Main.java
```

```java
package q1567;
class Printer {
    private boolean isOddPrinted = false;

    synchronized void printEven(int number) {
        while (!isOddPrinted) {
            try {
                wait();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        System.out.println("Even:" + number);
        isOddPrinted = false;
        notify();
    }

    synchronized void printOdd(int number) {
        //Complete this method based on printEven()
        while (isOddPrinted) {
            try {
                wait();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        System.out.println("Odd:" + number);
        isOddPrinted = true;
        notify();
    }
}

public class Main {
    public static void main(String[] args) {
        Printer printer = new Printer();
        int count = Integer.parseInt(args[0]);
        Thread t1 = new Thread(new Runnable() {
            public void run() {
                for (int i = 1; i <= count; i += 2) {
                    printer.printOdd(i);
                }
            }
        });

        //create a thread t2 for printing even numbers
        Thread t2 = new Thread(() -> {
            for (int i = 2; i <= count; i += 2) {
                printer.printEven(i);
            }
        });

        t1.start();
        t2.start();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Odd:1
Even:2
Odd:3
Even:4
Odd:5
```

Test Case - 2

User Output

```
Odd:1
Even:2
Odd:3
```

Test Case - 3

User Output

```
Odd:1
Even:2
Odd:3
Even:4
Odd:5
Even:6
Odd:7
```

Test Case - 4

User Output

```
Odd:1
Even:2
Odd:3
Even:4
Odd:5
Even:6
Odd:7
Even:8
Odd:9
```

S.No: 90	Exp. Name: ***Inter Thread Communication***	**Date: 2023-12-15**

Aim:

Ping Pong Game Simulation:
Consider a game of ping pong being played between two players. The game is played in rounds, and in each round, one player hits the ball and the other player hits it back. The players must take turns hitting the ball, and they cannot hit the ball out of turn.
Implement the**hitBall**method in the**PingPong**class using**wait** and**notifyAll**to ensure proper synchronization between the two players.
Sample Input and Output:

```
Command line args 2
Player 1 hit the ball
Player 2 hit the ball
Player 1 hit the ball
Player 2 hit the ball
```

Source Code:

```
q1569/Main.java
```

```
package q1569;
class PingPong {
    private boolean player1Turn = true;

    public synchronized void hitBall(String player) {
        //write code here..
                while ((player.equals("Player 1") && !player1Turn) || (player.equals("Player
2") && player1Turn)) {
            try {
                wait();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }

        System.out.println(player + " hit the ball");
        player1Turn = !player1Turn;
        notifyAll();
    }
}

class Player implements Runnable {
    private PingPong pingPong;
    private String player;
    private int rounds;

    public Player(PingPong pingPong, String player, int rounds) {
        this.pingPong = pingPong;
        this.player = player;
        this.rounds = rounds;
    }

    @Override
    public void run() {
        for (int i = 0; i < rounds; i++) {
            pingPong.hitBall(player);
        }
    }
}

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

        int rounds = Integer.parseInt(args[0]);
        PingPong pingPong = new PingPong();
        Thread player1Thread = new Thread(new Player(pingPong, "Player 1", rounds));
        Thread player2Thread = new Thread(new Player(pingPong, "Player 2", rounds));

        player1Thread.start();
        player2Thread.start();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Player 1 hit the ball
Player 2 hit the ball
Player 1 hit the ball
Player 2 hit the ball
```

Test Case - 2

User Output

```
Player 1 hit the ball
Player 2 hit the ball
Player 1 hit the ball
Player 2 hit the ball
Player 1 hit the ball
Player 2 hit the ball
```

Test Case - 3

User Output

```
Player 1 hit the ball
Player 2 hit the ball
Player 1 hit the ball
Player 2 hit the ball
Player 1 hit the ball
Player 2 hit the ball
Player 1 hit the ball
Player 2 hit the ball
```

Test Case - 4

User Output

```
Player 1 hit the ball
Player 2 hit the ball
```

S.No: 91	Exp. Name: ***Inter Thread Communication***	**Date: 2023-12-15**

Aim:

Imagine a situation where you have three threads,**T1**,**T2**, and**T3**.**T1**and**T2**are having a conversation just like in the current code. However,**T3**is a special thread that needs to send a “status update” message after every two exchanges between**T1**and**T2**.

How would you adapt the existing code to accommodate this new requirement? What modifications would you make to ensure that afterT3sends its status update, the conversation between**T1**and**T2**continues smoothly?

Sample Test Case:

```
Command line args 2
Hi
Hi
How are you ?
I am good, what about you?
```

Source Code:

```
q1571/Main.java
```

```
package q1571;
class Chat {
   boolean flag = false;

   public synchronized void Question(String msg) {
      if (flag) {
         try {
            wait();
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
      }
      System.out.println(msg);
      flag = true;
      notify();
   }
        private int conversationCount = 0;
   public synchronized void Answer(String msg) {

        //write code here..
                while (!flag) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println(msg);
        flag = false;
        notifyAll();
   }
}
class T1 implements Runnable {
   Chat m;
   String[] s1 = { "Hi", "How are you ?", "I am also doing fine!" };
   int numConversations;

   public T1(Chat m1, int numConversations) {
      this.m = m1;
      this.numConversations = numConversations;
      new Thread(this, "Question").start();
   }

   public void run() {
      for (int i = 0; i < numConversations; i++) {
         m.Question(s1[i % s1.length]);
      }
   }
}

class T2 implements Runnable {
   Chat m;
   String[] s2 = { "Hi", "I am good, what about you?", "Great!" };
   int numConversations;
        //write code here...
        this.numConversations = numConversations;
        new Thread(this, "Answer").start();
    }

    public void run() {
        for (int i = 0; i < numConversations; i++) {
            m.Answer(s2[i % s2.length]);
        }
        }
}

public class Main {
   public static void main(String[] args) {
      Chat m = new Chat();
      int numConversations = Integer.parseInt(args[0]);
      new T1(m, numConversations);
      new T2(m, numConversations);
   }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Hi
Hi
How are you ?
I am good, what about you?
```

Test Case - 2

User Output

```
Hi
Hi
How are you ?
I am good, what about you?
I am also doing fine!
Great!
Hi
Hi
```

S.No: 92	Exp. Name: ***Reading and Writing in the Array Using Threads***	**Date: 2023-12-15**

Aim:
You are developing a multi-threaded application for a data processing system. The system needs to handle two operations: writing data to a shared array and reading data from the shared array.
The writing operation is performed by a writer thread, which takes input from the user for the data to be written to the shared array. The reading operation is performed by a reader thread, which reads the data from the shared array and prints it.
The system should ensure that the write and read operations are synchronized, i.e., the reader thread should not read data while the writer thread is writing data, and vice versa. The writer thread should notify the reader thread when it has finished writing data, and the reader thread should notify the writer thread when it has finished reading data.
Sample Test Case:

```
Enter the elements :
2
3
4
5
6
Writing done Successfully
The elements are :
2
3
4
5
6
Reading done successfully
```

Source Code:

```
q1586/Main.java
```

```
// package q1586;
// import java.io.*;
// import java.util.*;
// public class Main {
//      public static void main(String[] args)
//      {
//              // Array created for 5 elements
//              int a[] = new int[5];

//              // Thread created for write operation
//              Thread t1 = new Thread(new Runnable() {
//                      public void run()
//                      {
//                              // Here the array is being
//                              // synchronized
//                              synchronized (a)
//                              {
//                                      Scanner s = new Scanner(System.in);
//                                      System.out.println(
//                                              "Enter the elements : ");

//                                      for (int i = 0; i < 5; i++) {
//                                              a[i] = s.nextInt();
//                                      }

//                                      System.out.println(
//                                              "Writing done Successfully");
//              //              }
//              //      }
//              // });
//                                      a.notify();
//                              }
//                      }
//              });

//                                      // create Thread for read operation
//              Thread t2 = new Thread(new Runnable() {
//             public void run() {
//                 // Here the array is being
//                 // synchronized
//                 synchronized (a) {
//                     try {
//                         // Wait for the writer thread to finish writing
//                         a.wait();
//                     } catch (InterruptedException e) {
//                         e.printStackTrace();
//                     }

//                     System.out.println("The elements are : ");
//                     for (int i = 0; i < 5; i++) {
//                         System.out.println(a[i]);
//                     }

//                     System.out.println("Reading done successfully");
//                 }
//              // Write thread is started
//              t1.start();

//              // Read thread is started
//              t2.start();
//      }
// }

package q1586;
import java.io.*;
import java.util.*;

public class Main {
    public static void main(String[] args) {
        // Array created for 5 elements
        int a[] = new int[5];

        // Object used for synchronization
        Object lock = new Object();

        // Thread created for write operation
        Thread t1 = new Thread(new Runnable() {
            public void run() {
                // Here the array is being
                // synchronized
                synchronized (lock) {
                    Scanner s = new Scanner(System.in);
                    System.out.println("Enter the elements : ");

                    for (int i = 0; i < 5; i++) {
                        a[i] = s.nextInt();
                    }

                    System.out.println("Writing done Successfully");

                    // Notify the waiting threads (reader thread in this case)
                    lock.notify();
                }
            }
        });

        // Thread created for read operation
        Thread t2 = new Thread(new Runnable() {
            public void run() {
                // Here the array is being
                // synchronized
                synchronized (lock) {
                    try {
                        // Wait for the writer thread to finish writing
                        while (a[0] == 0) { // Check the condition using a loop
                            lock.wait();
                        }
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    System.out.println("The elements are : ");
                    for (int i = 0; i < 5; i++) {
                        System.out.println(a[i]);
                    }

                    System.out.println("Reading done successfully");
                }
            }
        });

        // Write thread is started
        t1.start();

        // Read thread is started
        t2.start();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the elements :
2
3
4
5
6
Writing done Successfully
The elements are :
2
3
4
5
6
Reading done successfully
```

S.No: 93	Exp. Name: ***Producer Consumer Problem***	**Date: 2023-12-15**

Aim:

Imagine a factory assembly line where there is a single production machine and a single quality control machine. The production machine produces items and places them on a conveyor belt (the**Buffer**). However, it can only place one item on the conveyor belt at a time. If there's already an item on the belt, the production machine has to wait until the belt is clear before it can place another item.
On the other side of the conveyor belt, the quality control machine (the**Consumer**) checks each item. It can only check one item at a time. If the conveyor belt is empty, the quality control machine has to wait until a new item is placed on the belt.
The**Producer**class represents the production machine. It produces a certain number of items (**maxItems**) and places each one on the conveyor belt .
The**Consumer**class represents the quality control machine. It checks a certain number of items (**maxItems**) by taking each one from the conveyor belt.
The**Main**class sets up the production and quality control machines and starts them running. The number of items to be produced and checked is passed in as a command-line argument.
Sample Test Case:

```
Command line args 3
Produced: 0
Consumed: 0
Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
```

Source Code:

```
q1587/Main.java
```

```
package q1587;
class Buffer {
    private int item = -1;

    public synchronized void produce(int item) throws InterruptedException {
       while (this.item != -1) {
            wait();
           }
                // Place the produced item on the buffer
        this.item = item;
        System.out.println("Produced: " + item);

        // Notify the consumer that there is an item available
        notify();
    }

    public synchronized int consume() throws InterruptedException {
      while (this.item == -1) {
            wait();
        }

        // Consume the item from the buffer
        int consumedItem = this.item;
        this.item = -1;
        System.out.println("Consumed: " + consumedItem);

        // Notify the producer that the buffer is empty
        notify();

        return consumedItem;
    }
}

class Producer implements Runnable {
    private Buffer buffer;
    private int maxItems;

    public Producer(Buffer buffer, int maxItems) {
        this.buffer = buffer;
        this.maxItems = maxItems;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < maxItems; i++) {
                buffer.produce(i);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

class Consumer implements Runnable {
    public Consumer(Buffer buffer, int maxItems) {
        this.buffer = buffer;
        this.maxItems = maxItems;
    }
        //write code here...
    public void run() {
        try {
            for (int i = 0; i < maxItems; i++) {
                buffer.consume();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        }
}

public class Main {
    public static void main(String[] args) {
        Buffer buffer = new Buffer();

        int maxItems = Integer.parseInt(args[0]);  // Number of items to be produced and
consumed

        Thread producerThread = new Thread(new Producer(buffer, maxItems));
        Thread consumerThread = new Thread(new Consumer(buffer, maxItems));

        producerThread.start();
        consumerThread.start();
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Produced: 0
Consumed: 0
Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Consumed: 3
Produced: 4
Consumed: 4
```

Test Case - 2

User Output

```
Produced: 0
Consumed: 0
Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
```

S.No: 94	Exp. Name: ***Sorting***	**Date: 2023-12-15**

Aim:
You are participating in a coding competition where you are given an array of integers and asked to sort it in ascending order.
Complete the below code to achieve the given scenario

Sample Test Case

```
Enter the number of elements: 5
Enter the elements:
34
22
34
12
67
Sorted Array: [12, 22, 34, 34, 67]
```

Source Code:

ArraySorting.java

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

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

        System.out.print("Enter the number of elements: ");
        int size = scanner.nextInt();

        int[] numbers = new int[size];

        // write the code here
                System.out.println("Enter the elements:");
        for (int i = 0; i < size; i++) {
            numbers[i] = scanner.nextInt();
        }

        // Sorting the array in ascending order
        Arrays.sort(numbers);

        System.out.println("Sorted Array: " + Arrays.toString(numbers));
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of elements:
6
Enter the elements:
```

```
23
34
45
12
56
67
Sorted Array: [12, 23, 34, 45, 56, 67]
```

Test Case - 2

User Output

```
Enter the number of elements:
3
Enter the elements:
34
12
23
Sorted Array: [12, 23, 34]
```

Test Case - 3

User Output

```
Enter the number of elements:
5
Enter the elements:
45
24
26
13
12
Sorted Array: [12, 13, 24, 26, 45]
```

Test Case - 4

User Output

```
Enter the number of elements:
4
Enter the elements:
23
78
89
98
Sorted Array: [23, 78, 89, 98]
```

S.No: 95	Exp. Name: ***Binary Search***	**Date: 2023-12-15**

Aim:

Imagine you are developing a Java application where you need to sort an array of integers and then search for a specific element in the sorted array. You decide to use the provided Java program to accomplish this task.

Note: Use Arrays.binarySearch() method in order to find the index of given element

Sample Test Case-1

```
Enter the number of elements: 4
Enter the elements:
23
22
21
43
Enter the element to search: 44
Sorted array: [21, 22, 23, 43]
Element 44 not found
```

Sample Test Case-2

```
Enter the number of elements: 4
Enter the elements:
23
12
45
23
Enter the element to search: 23
Sorted array: [12, 23, 23, 45]
Element 23 found at index 1
```

Source Code:

```
q1554/ArraySearch.java
```

```
package q1554;
import java.util.Arrays;
import java.util.Scanner;

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

        System.out.print("Enter the number of elements: ");
        int size = scanner.nextInt();
        int[] numbers = new int[size];
                System.out.println("Enter the elements:");
        for (int i = 0; i < size; i++) {
            numbers[i] = scanner.nextInt();
        }

        // Sorting the array
        Arrays.sort(numbers);

        // Searching for an element
        System.out.print("Enter the element to search: ");
        int searchElement = scanner.nextInt();

                // Displaying the sorted array
        System.out.print("Sorted array: ");
        System.out.println(Arrays.toString(numbers));

        int index = Arrays.binarySearch(numbers, searchElement);

        if (index >= 0) {
            System.out.println("Element " + searchElement + " found at index " + index);
        } else {
            System.out.println("Element " + searchElement + " not found");
        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of elements:
5
Enter the elements:
23
12
24
35
13
Enter the element to search:
24

```
Sorted array: [12, 13, 23, 24, 35]
Element 24 found at index 3
```

Test Case - 2

User Output

```
Enter the number of elements:
3
Enter the elements:
23
12
45
Enter the element to search:
23
Sorted array: [12, 23, 45]
Element 23 found at index 1
```

Test Case - 3

User Output

```
Enter the number of elements:
4
Enter the elements:
12
11
22
33
Enter the element to search:
22
Sorted array: [11, 12, 22, 33]
Element 22 found at index 2
```

Test Case - 4

User Output

```
Enter the number of elements:
5
Enter the elements:
66
77
55
44
56
Enter the element to search:
56
Sorted array: [44, 55, 56, 66, 77]
Element 56 found at index 2
```

Test Case - 5

User Output

```
Enter the number of elements:
2
Enter the elements:
23
45
Enter the element to search:
45
Sorted array: [23, 45]
Element 45 found at index 1
```

S.No: 96	Exp. Name: ***Compare Both Arrays are Equal or Not***	**Date: 2023-12-15**

Aim:

Sarah is working on a project where she needs to compare two arrays to determine if they are equal or not. She's building a program in Java to accomplish this task.
She starts by writing a Java class named **ArrayEquality** This class contains a **main** method where she initializes a **Scanner** object to take input from the user.
When the program runs, it prompts the user to enter the number of elements for the first array.
Next, the program asks for the number of elements for the second array.

The program then uses the **Arrays.equals** method to compare the two arrays, **arr1** and **arr2**. Since both arrays contain the same elements in the same order, it returns **true** and returns **"**Arrays are equal" to the console. otherwise, it returns "**Arrays are not equal.**"

Sample Test Case-1

```
Enter the number of elements for the first array: 3
Enter the elements for the first array:
23
66
77
Enter the number of elements for the second array: 2
Enter the elements for the second array:
45
66
Arrays are not equal
```

Sample Test Case-2

```
Enter the number of elements for the first array: 4
Enter the elements for the first array:
23
24
25
26
Enter the number of elements for the second array: 4
Enter the elements for the second array:
23
24
25
26
Arrays are equal
```

Sample Test Case-3

```
Enter the number of elements for the first array: 4
Enter the elements for the first array:
23
12
12
23
Enter the number of elements for the second array: 4
Enter the elements for the second array:
23
45
34
65
Arrays are not equal
```

Source Code:

```
q1555/ArrayEquality.java
```

```
package q1555;
import java.util.Arrays;
import java.util.Scanner;

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

        //write the code here
                System.out.print("Enter the number of elements for the first array: ");
        int size1 = scanner.nextInt();
        int[] arr1 = new int[size1];

        System.out.println("Enter the elements for the first array:");
        for (int i = 0; i < size1; i++) {
            arr1[i] = scanner.nextInt();
        }

        System.out.print("Enter the number of elements for the second array: ");
        int size2 = scanner.nextInt();
        int[] arr2 = new int[size2];

        System.out.println("Enter the elements for the second array:");
        for (int i = 0; i < size2; i++) {
            arr2[i] = scanner.nextInt();
        }

        // Using Arrays.equals method to compare arrays
        boolean areEqual = Arrays.equals(arr1, arr2);

        if (areEqual) {
            System.out.println("Arrays are equal");
        } else {
            System.out.println("Arrays are not equal");
        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of elements for the first array:
5
Enter the elements for the first array:
12
34
45
61
62
Enter the number of elements for the second array:
```

```
5
Enter the elements for the second array:
12
34
45
61
62
Arrays are equal
```

Test Case - 2

User Output

```
Enter the number of elements for the first array:
3
Enter the elements for the first array:
34
45
65
Enter the number of elements for the second array:
2
Enter the elements for the second array:
45
76
Arrays are not equal
```

Test Case - 3

User Output

```
Enter the number of elements for the first array:
3
Enter the elements for the first array:
13
24
35
Enter the number of elements for the second array:
3
Enter the elements for the second array:
23
24
13
Arrays are not equal
```

Test Case - 4

User Output

```
Enter the number of elements for the first array:
3
```

```
Enter the elements for the first array:
12
13
16
Enter the number of elements for the second array:
3
Enter the elements for the second array:
12
13
16
Arrays are equal
```

S.No: 97	Exp. Name: ***Insert the Elements using ArrayList***	**Date: 2023-12-15**

Aim:

Ram is developing a program in Java to manage a list of tasks for his daily activities. He decides to create a Java class named **ListOperations** to handle the operations on his list using **ArrayList**.

When Ram runs the program, it asks him to input the number of elements he wants to add to the list.

Sample Test Case-1

```
Enter the number of elements you want to add:
3
Enter elements:
23
43
45
Elements in the list:
23
43
45
```

Sample Test Case-2

```
Enter the number of elements you want to add:
5
Enter elements:
76
65
I
56.9
codetantra
Elements in the list:
76
65
I
56.9
codetantra
```

Source Code:

```
q1557/ListOperations.java
```

```
package q1557;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class ListOperations {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
                System.out.println("Enter the number of elements you want to add:");
        int numElements = scanner.nextInt();

        List<Object> taskList = new ArrayList<>();

        System.out.println("Enter elements:");
        for (int i = 0; i < numElements; i++) {
            if (scanner.hasNextInt()) {
                int element = scanner.nextInt();
                taskList.add(element);
            } else if (scanner.hasNextDouble()) {
                double element = scanner.nextDouble();
                taskList.add(element);
            } else {
                String element = scanner.next();
                taskList.add(element);
            }
        }

        System.out.println("Elements in the list:");
        for (Object element : taskList) {
            System.out.println(element);
        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of elements you want to add:
4
Enter elements:
code
tantra
hyd
200
Elements in the list:
code
tantra
hyd
200
```

Test Case - 2

User Output

```
Enter the number of elements you want to add:
3
Enter elements:
23
43
45
Elements in the list:
23
43
45
```

Test Case - 3

User Output

```
Enter the number of elements you want to add:
5
Enter elements:
23.8
34
code
hyd
tantra
Elements in the list:
23.8
34
code
hyd
tantra
```

Test Case - 4

User Output

```
Enter the number of elements you want to add:
2
Enter elements:
india
country
Elements in the list:
india
country
```

S.No: 99	Exp. Name: ***Enter New Joined Employee Details using List***	**Date: 2023-12-15**

Aim:

Hari, a human resources manager, needs to input and store new joined employee details for a small project team using the provided Java program named **Main**.
When the program starts, it prompts Hari to enter employee details.

- The program asks for the employee's name.
- Then, the program requests the employee's ID.

After entering details of employees, the program displays the employee information.
note: use constructor and get() method.

Sample Test Case:

```
Enter employee name:
kalyani
Enter employee ID:
116
Enter employee name:
ravi
Enter employee ID:
123
Employee Name: kalyani
Employee ID: 116
------------
Employee Name: ravi
Employee ID: 123
------------
```

Source Code:

```
q1560/Main.java
```

```
package q1560;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

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

        for (int i = 0; i < 2; i++) {
            System.out.println("Enter employee name:");
            String name = scanner.nextLine();

            System.out.println("Enter employee ID:");
            int employeeId = scanner.nextInt();
            scanner.nextLine(); // consume the newline character

            Employee employee = new Employee(name, employeeId);
            employees.add(employee);
        }

        for (Employee employee : employees) {
            System.out.println("Employee Name: " + employee.getName());
            System.out.println("Employee ID: " + employee.getEmployeeId());
            System.out.println("------------");
        }
        }

static class Employee {
    private String name;
    private int employeeId;
    public Employee(String name, int employeeId) {
        this.name = name;
        this.employeeId = employeeId;
    }
    public String getName() {
        return name;
    }
    public int getEmployeeId() {
    return employeeId;
    }
  }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter employee name:
ram
Enter employee ID:
```

```
101
Enter employee name:
ravi
Enter employee ID:
102
Employee Name: ram
Employee ID: 101
------------
Employee Name: ravi
Employee ID: 102
------------
```

Test Case - 2

User Output

```
Enter employee name:
raghu
Enter employee ID:
103
Enter employee name:
raj
Enter employee ID:
104
Employee Name: raghu
Employee ID: 103
------------
Employee Name: raj
Employee ID: 104
------------
```

Test Case - 3

User Output

```
Enter employee name:
sai
Enter employee ID:
106
Enter employee name:
sunny
Enter employee ID:
108
Employee Name: sai
Employee ID: 106
------------
Employee Name: sunny
Employee ID: 108
------------
```

Test Case - 4

User Output

```
Enter employee name:
pradeep
Enter employee ID:
123
Enter employee name:
prashanth
Enter employee ID:
345
Employee Name: pradeep
Employee ID: 123
------------
Employee Name: prashanth
Employee ID: 345
------------
```

S.No: 100	Exp. Name: ***Map interface Using All Methods***	**Date: 2023-12-15**

Aim:

Suppose you are developing a simple program to manage student records for a small school or tutoring center. The program is meant to take in three student IDs and names, store them in a map, display the details of all entered students, and provide an option to delete a student's record based on their ID.

11. The program starts by prompting the user to enter details.
12. The user inputs the student ID and name for each student as requested.
13. Once the details for students are entered, the program displays the entered student details.
14. Next, the user is prompted to enter a student ID they want to delete from the records.
    - Based on the input, the program checks if the student ID exists in the records:If found, it deletes the corresponding record and confirms the deletion.If not found, it displays a message indicating that the student ID was not found in the records.
15. Finally, the program displays the updated student details after deletion (if any).

Sample Test Case-1

```
Enter student ID:
161
Enter student name:
bhanu
Enter student ID:
162
Enter student name:
pruthvi
Enter student ID:
163
Enter student name:
priya
Student Details:
ID: 161, Name: bhanu
ID: 162, Name: pruthvi
ID: 163, Name: priya
Enter the student ID to delete:
162
Student with ID 162 deleted successfully!
Updated Student Details:
ID: 161, Name: bhanu
ID: 163, Name: priya
```

Sample Test Case-2

```
Enter student ID:
174
Enter student name:
bhavani
Enter student ID:
175
Enter student name:
durga
Enter student ID:
176
Enter student name:
malathi
Student Details:
ID: 176, Name: malathi
ID: 174, Name: bhavani
ID: 175, Name: durga
Enter the student ID to delete:
178
Student ID not found!
Updated Student Details:
ID: 176, Name: malathi
ID: 174, Name: bhavani
ID: 175, Name: durga
```

Source Code:

```
q1561/MapExample.java
```

```
package q1561;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class MapExample {
    public static void main(String[] args) {
        Map<Integer, String> studentNames = new HashMap<>();
        Scanner sc = new Scanner(System.in);
                for (int i = 0; i < 3; i++) {
            System.out.println("Enter student ID:");
            int studentID = sc.nextInt();
            sc.nextLine(); // Consume the newline character

            System.out.println("Enter student name:");
            String studentName = sc.nextLine();

            studentNames.put(studentID, studentName);
        }

        System.out.println("Student Details:");
        for (Map.Entry<Integer, String> entry : studentNames.entrySet()) {
            System.out.println("ID: " + entry.getKey() + ", Name: " + entry.getValue());
        }

        System.out.println("Enter the student ID to delete:");
        int idToDelete = sc.nextInt();

        if (studentNames.containsKey(idToDelete)) {
            studentNames.remove(idToDelete);
            System.out.println("Student with ID " + idToDelete + " deleted successfully!");
        } else {
            System.out.println("Student ID not found!");
        }

        System.out.println("Updated Student Details:");
        for (Map.Entry<Integer, String> entry : studentNames.entrySet()) {
            System.out.println("ID: " + entry.getKey() + ", Name: " + entry.getValue());
        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter student ID:
145
Enter student name:
sadaf
Enter student ID:
146
```

```
Enter student name:
ganga
Enter student ID:
147
Enter student name:
gayatri
Student Details:
ID: 145, Name: sadaf
ID: 146, Name: ganga
ID: 147, Name: gayatri
Enter the student ID to delete:
146
Student with ID 146 deleted successfully!
Updated Student Details:
ID: 145, Name: sadaf
ID: 147, Name: gayatri
```

Test Case - 2

User Output

```
Enter student ID:
148
Enter student name:
kaveri
Enter student ID:
149
Enter student name:
kavya
Enter student ID:
140
Enter student name:
keerthana
Student Details:
ID: 148, Name: kaveri
ID: 149, Name: kavya
ID: 140, Name: keerthana
Enter the student ID to delete:
140
Student with ID 140 deleted successfully!
Updated Student Details:
ID: 148, Name: kaveri
ID: 149, Name: kavya
```

Test Case - 3

User Output

```
Enter student ID:
150
```

```
Enter student name:
niha
Enter student ID:
151
Enter student name:
niharika
Enter student ID:
152
Enter student name:
nuthan
Student Details:
ID: 150, Name: niha
ID: 151, Name: niharika
ID: 152, Name: nuthan
Enter the student ID to delete:
153
Student ID not found!
Updated Student Details:
ID: 150, Name: niha
ID: 151, Name: niharika
ID: 152, Name: nuthan
```

Test Case - 4

User Output

```
Enter student ID:
154
Enter student name:
rakesh
Enter student ID:
155
Enter student name:
mukesh
Enter student ID:
166
Enter student name:
rithesh
Student Details:
ID: 166, Name: rithesh
ID: 154, Name: rakesh
ID: 155, Name: mukesh
Enter the student ID to delete:
155
Student with ID 155 deleted successfully!
Updated Student Details:
ID: 166, Name: rithesh
ID: 154, Name: rakesh
```

S.No: 101	Exp. Name: ***Example for Iterator(Geography Quiz Administration)***	**Date: 2023-12-16**

Aim:

Geography Quiz Administration

Imagine you're creating a basic quiz administration system for a geography class. The program is designed to gather information about specific countries and their capitals for quiz preparation purposes.

User Interaction:

16. The program begins by asking the user to input the country and its corresponding capital for different countries.
17. The user enters the name of the country and its capital consecutively for each iteration.
18. Once the information for countries is entered, the program displays a list of the entered country-capital pairs.
19. The system then generates a quiz using this information for the students.

Note: Use Iterator

Map interface (key, value).Consider country name as key Capital as value.

Source Code:

q1562/IteratorEx.java

```java
package q1562;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Scanner;

public class IteratorEx {
    public static void main(String[] args) {
        Map<String, String> countryCapitals = new HashMap<>();
        Scanner scanner = new Scanner(System.in);
          for (int i = 0; i < 3; i++) {
                          System.out.println("Enter country:");
            String country = scanner.nextLine();

            System.out.println("Enter capital:");
            String capital = scanner.nextLine();

            countryCapitals.put(country, capital);
        }

        System.out.println("Country Capitals:");

        // Using Iterator to iterate over the entries
        Iterator<Map.Entry<String, String>> iterator =
countryCapitals.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry<String, String> entry = iterator.next();
            System.out.println("Country: " + entry.getKey() + ", Capital: " +
entry.getValue());
        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter country:
Angila
Enter capital:
Launda
Enter country:
Australia
Enter capital:
Conberra
Enter country:
Bangladesh
Enter capital:
Dhaka
Country Capitals:
Country: Bangladesh, Capital: Dhaka
Country: Australia, Capital: Conberra
Country: Angila, Capital: Launda
```

Test Case - 2

User Output

```
Enter country:
Bhutan
Enter capital:
Timphu
Enter country:
Canada
Enter capital:
Ottawa
Enter country:
China
Enter capital:
Beijing
Country Capitals:
Country: Canada, Capital: Ottawa
Country: Bhutan, Capital: Timphu
Country: China, Capital: Beijing
```

Test Case - 3

User Output

```
Enter country:
India
Enter capital:
```

```
Delhi
Enter country:
Afghanistan
Enter capital:
Kabul
Enter country:
Albania
Enter capital:
Tirana
Country Capitals:
Country: Afghanistan, Capital: Kabul
Country: India, Capital: Delhi
Country: Albania, Capital: Tirana
```

Test Case - 4

User Output

```
Enter country:
China
Enter capital:
Beijing
Enter country:
Colombia
Enter capital:
Bagota
Enter country:
Cuba
Enter capital:
Havana
Country Capitals:
Country: Colombia, Capital: Bagota
Country: Cuba, Capital: Havana
Country: China, Capital: Beijing
```

Aim:

Creating a Color Palette

Suppose you're developing an application to create a custom color palette where users can input their preferred colors. The program allows users to enter colors in a forward sequence, then displays these colors in both forward and reverse order for a better understanding of the palette.

User Interaction:

20. The program prompts the user to input different colors.
21. The user enters each color one by one.
22. After entering the colors, the program displays the colors in the order they were inputted (forward traversal).
23. Following that, the program showcases the same colors but in reverse order (backward traversal).
24. Users can take note of how the colors look in both sequences, helping them visualize the color palette.

Source Code:

q1563/ListIteratorEx.java

```
package q1563;
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;
import java.util.Scanner;

public class ListIteratorEx {
    public static void main(String[] args) {
        List<String> colors = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);
        for (int i = 0; i < 3; i++) {
            System.out.println("Enter a color:");
            String color = scanner.nextLine();
            colors.add(color);
        }
                System.out.println("Forward Traversal:");
        ListIterator<String> forwardIterator = colors.listIterator();
        while (forwardIterator.hasNext()) {
            System.out.println(forwardIterator.next());
        }

        // Backward Traversal
        System.out.println("Backward Traversal:");
        ListIterator<String> backwardIterator = colors.listIterator(colors.size());
        while (backwardIterator.hasPrevious()) {
            System.out.println(backwardIterator.previous());
        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter a color:
blue
Enter a color:
pink
Enter a color:
yellow
Forward Traversal:
blue
pink
yellow
Backward Traversal:
yellow
pink
blue
```

Test Case - 2

User Output

```
Enter a color:
black
Enter a color:
red
Enter a color:
white
Forward Traversal:
black
red
white
Backward Traversal:
white
red
black
```

Test Case - 3

User Output

```
Enter a color:
green
Enter a color:
orange
Enter a color:
maroon
Forward Traversal:
green
orange
maroon
Backward Traversal:
```

```
maroon
orange
green
```

Test Case - 4

User Output

```
Enter a color:
lime
Enter a color:
golden
Enter a color:
salmon
Forward Traversal:
lime
golden
salmon
Backward Traversal:
salmon
golden
lime
```

S.No: 103	Exp. Name: ***Enumarator Example using Vector***	Date: 2023-12-15

Aim:
The store manager to input new fruits into the system. The program then displays the list of fruits entered, providing a simple way to keep track of the available fruits in the store's inventory. This basic system could be expanded to include more features like quantity tracking of fruits, etc.

Note: Use Vector Class in Java Collections framework

Source Code:

q1565/EnumeratorEx.java

```
package q1565;
import java.util.Enumeration;
import java.util.Vector;
import java.util.Scanner;

public class EnumeratorEx {
    public static void main(String[] args) {
        Vector<String> fruits = new Vector<>();
        Scanner scanner = new Scanner(System.in);
        for (int i = 0; i < 3; i++) {
                        System.out.println("Enter a fruit:");
            String fruit = scanner.nextLine();
            fruits.add(fruit);
        }

        // Displaying Fruits
        System.out.println("Fruits:");
        Enumeration<String> fruitEnumeration = fruits.elements();
        while (fruitEnumeration.hasMoreElements()) {
            System.out.println(fruitEnumeration.nextElement());
        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter a fruit:
Grapes
Enter a fruit:
Mango
Enter a fruit:
Strawberry
Fruits:
Grapes
Mango
Strawberry
```

Test Case - 2

User Output

```
Enter a fruit:
Guava
Enter a fruit:
Lime
Enter a fruit:
Banana
Fruits:
Guava
Lime
Banana
```

Test Case - 3

User Output

```
Enter a fruit:
Banana
Enter a fruit:
Grapes
Enter a fruit:
Watermelon
Fruits:
Banana
Grapes
Watermelon
```

Test Case - 4

User Output

```
Enter a fruit:
Cucumber
Enter a fruit:
Mango
Enter a fruit:
Blueberry
Fruits:
Cucumber
Mango
Blueberry
```

S.No: 104	Exp. Name: ***Birthday Celebration Event***	Date: 2023-12-15

Aim:

Birthday Celebration Event

Imagine you're organizing a community event to celebrate the birthdays of individuals in a retirement community or a neighborhood. You've created a program to gather the names and ages of the attendees and simulate a birthday celebration for each person present.

User Interaction:

25. The program starts by prompting the organizer to input the number of people attending the event.
26. For each attendee, the organizer enters their name and age.
27. After all the details are entered, the program simulates a birthday celebration for each person by incrementing their age and displaying a message about the celebration.
28. The celebration event proceeds with other planned activities after everyone's birthdays are acknowledged.

The program allows the organizer to collect the names and ages of attendees for a birthday celebration event. It simulates the celebration by incrementing each person's age and acknowledging their birthday within the program.

Sample Test Case:

```
Enter the number of people:
2
Enter name for person 1:
rekha
Enter age for person 1:
45
Enter name for person 2:
lekha
Enter age for person 2:
55
Celebrating Birthdays:
rekha is celebrating their birthday! They are now 46 years old.
lekha is celebrating their birthday! They are now 56 years old.
```

Source Code:

```
q1566/ForEachEx.java
```

```
package q1566;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
class Person {
    private String name;
    private int age;

   public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
        public void celebrateBirthday() {
        System.out.println(name + " is celebrating their birthday! They are now " + (age +
1) + " years old.");
    }
}

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

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

        for (int i = 1; i <= numberOfPeople; i++) {
            System.out.println("Enter name for person " + i + ":");
            String name = scanner.nextLine();

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

            attendees.add(new Person(name, age));
        }

        System.out.println("Celebrating Birthdays:");
        for (Person person : attendees) {
            person.celebrateBirthday();
        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
`Enter the number of people:`
3
`Enter name for person 1:`

```
ram
Enter age for person 1:
13
Enter name for person 2:
rani
Enter age for person 2:
23
Enter name for person 3:
ravi
Enter age for person 3:
34
Celebrating Birthdays:
ram is celebrating their birthday! They are now 14 years old.
rani is celebrating their birthday! They are now 24 years old.
ravi is celebrating their birthday! They are now 35 years old.
```

Test Case - 2

User Output

```
Enter the number of people:
3
Enter name for person 1:
nithin
Enter age for person 1:
28
Enter name for person 2:
nikhil
Enter age for person 2:
25
Enter name for person 3:
niharika
Enter age for person 3:
67
Celebrating Birthdays:
nithin is celebrating their birthday! They are now 29 years old.
nikhil is celebrating their birthday! They are now 26 years old.
niharika is celebrating their birthday! They are now 68 years old.
```

Test Case - 3

User Output

```
Enter the number of people:
2
Enter name for person 1:
rekha
Enter age for person 1:
45
Enter name for person 2:
```

```
lekha
Enter age for person 2:
55
Celebrating Birthdays:
rekha is celebrating their birthday! They are now 46 years old.
lekha is celebrating their birthday! They are now 56 years old.
```

Test Case - 4

User Output

```
Enter the number of people:
2
Enter name for person 1:
harsha
Enter age for person 1:
45
Enter name for person 2:
kapil
Enter age for person 2:
65
Celebrating Birthdays:
harsha is celebrating their birthday! They are now 46 years old.
kapil is celebrating their birthday! They are now 66 years old.
```

S.No: 105	Exp. Name: ***Example for ForEach()***	Date: 2023-12-15

Aim:

Salary Calculation in a Small Business

Imagine a small business owner who needs to calculate the total salary expenditure for their employees in a given month. The owner uses this program to input the names and respective salaries of the employees and obtain the total salary payout.

User Interaction:

29. The program starts by asking the business owner to input the number of employees in their company.
30. For each employee, the owner enters the name and salary.
31. Once all the employee details are entered, the program calculates the total salary expenditure for all employees combined.
32. The owner can use this total salary amount for financial planning and budgeting purposes.

The program allows the business owner to input the names and salaries of their employees. It then calculates and displays the total salary expenditure for all the employees combined. This could aid the owner in managing their financial records and making informed decisions regarding budget allocation and future planning.

Sample Test Case:

```
Enter the number of employees:
2
Enter name for employee 1:
tarun
Enter salary for employee 1:
44000
Enter name for employee 2:
varun
Enter salary for employee 2:
33000
Total salary of all employees: 77000.0
```

Source Code:

```
q1568/Main.java
```

```
package q1568;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class Employee {
    private String name;
    private double salary;

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

    public double getSalary() {
        return salary;
    }
}
public class Main {
    public static void main(String[] args) {
                Scanner scanner = new Scanner(System.in);
        List<Employee> employees = new ArrayList<>();

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

        for (int i = 1; i <= numberOfEmployees; i++) {
            System.out.println("Enter name for employee " + i + ":");
            String name = scanner.nextLine();

            System.out.println("Enter salary for employee " + i + ":");
            double salary = scanner.nextDouble();
            scanner.nextLine(); // Consume the newline character

            employees.add(new Employee(name, salary));
        }

        double totalSalary = calculateTotalSalary(employees);
        System.out.println("Total salary of all employees: " + totalSalary);
    }

    private static double calculateTotalSalary(List<Employee> employees) {
        double totalSalary = 0.0;
        for (Employee employee : employees) {
            totalSalary += employee.getSalary();
        }
        return totalSalary;
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of employees:
3
Enter name for employee 1:
ram
Enter salary for employee 1:
23000
Enter name for employee 2:
sita
Enter salary for employee 2:
34000
Enter name for employee 3:
eshwar
Enter salary for employee 3:
12000
Total salary of all employees: 69000.0
```

Test Case - 2

User Output

```
Enter the number of employees:
4
Enter name for employee 1:
rakesh
Enter salary for employee 1:
34000
Enter name for employee 2:
rani
Enter salary for employee 2:
23000
Enter name for employee 3:
ram
Enter salary for employee 3:
56000
Enter name for employee 4:
renuka
Enter salary for employee 4:
33000
Total salary of all employees: 146000.0
```

Test Case - 3

User Output

```
Enter the number of employees:
2
Enter name for employee 1:
sachin
```

```
Enter salary for employee 1:
35000
Enter name for employee 2:
sravya
Enter salary for employee 2:
67000
Total salary of all employees: 102000.0
```

Test Case - 4

User Output

```
Enter the number of employees:
3
Enter name for employee 1:
sarika
Enter salary for employee 1:
23450
Enter name for employee 2:
revathi
Enter salary for employee 2:
24568
Enter name for employee 3:
theertha
Enter salary for employee 3:
45800
Total salary of all employees: 93818.0
```

S.No: 106	Exp. Name: ***Reading Single file***	**Date: 2023-12-16**

Aim:

This Java code demonstrates how to read data from a file named **"sample.txt"** using a Scanner.
This code initializes a File object representing "**sample.txt"** and uses a Scanner to read its content line by line.
The **hasNextLine()** method checks if there's another line to read, and **nextLine()** reads the next line of text from the file. Finally, it prints each line to the console until there are no more lines to read.

Note: file name: **"q1574/sample.txt"** use with package

Source Code:

q1574/SingleFileEx.java

```
package q1574;
import java.io.File;
import java.util.Scanner;
public class SingleFileEx {
    public static void main(String[] args) throws Exception
        {
                // Specify the file name along with the package
        String fileName = "q1574/sample.txt";

        // Create a File object representing the input file
        File file = new File(fileName);

        // Check if the file exists
        if (!file.exists()) {
            System.out.println("File not found: " + fileName);
            return;
        }

        // Use a try-with-resources statement to automatically close the Scanner
        try (Scanner scanner = new Scanner(file)) {
            // Read each line from the file using the Scanner
            while (scanner.hasNextLine()) {
                // Read the next line and print it to the console
                String line = scanner.nextLine();
                System.out.println(line);
            }
        } catch (Exception e) {
            // Handle exceptions, if any
            e.printStackTrace();
                }
        }
}
```

q1574/sample.txt

```
CodeTantra, Interactive Platform for Virtual School, Virtual University,
Online Classes, Assessments,
Low Bandwidth, Teach Anywhere, Learn Coding, ...
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
CodeTantra, Interactive Platform for Virtual School, Virtual University,
Online Classes, Assessments,
Low Bandwidth, Teach Anywhere, Learn Coding, ...
```

S.No: 107	Exp. Name: ***Reading Multiple Files***	Date: 2023-12-16

Aim:

Write the program to read multiple files, you can create an array of file names and loop through each file to read its content.

- You have a directory containing log files from various system components.
- The log files are named **Sample1.txt, Sample2.txt, and Sample3.txt**, which correspond to different software modules or components **(Sample1,Sample2,Sample3).**
- The **MultipleFilesEx** program is utilized to sequentially read the contents of each log file specified in the **filenames** array.
- During execution, the program iterates through each file name in the array.
- For each file, it opens the file, reads its content line by line, and prints the content to the console.
- If any error occurs while reading a file (for instance, if a file is missing or inaccessible), the program catches the exception and prints an error message indicating the issue encountered for that particular file.
- This allows the program to continue processing other files even if an error occurs with one of them.

Source Code:

```
q1575/MultipleFilesEx.java
```

```
package q1575;
import java.io.File;
import java.util.Scanner;

public class MultipleFilesEx {
    public static void main(String[] args) {
                // Array of file names
        String[] filenames = {"Sample1.txt", "Sample2.txt", "Sample3.txt"};

        // Iterate through each file name in the array
        for (String filename : filenames) {
            // Print a message indicating the file being read
            System.out.println("Reading contents of " + filename + ":");

            // Create a File object representing the input file
            File file = new File(filename);

            // Check if the file exists
            if (!file.exists()) {
                System.out.println("File not found: " + filename);
                continue; // Skip to the next iteration if the file is not found
            }

            // Use a try-with-resources statement to automatically close the Scanner
            try (Scanner scanner = new Scanner(file)) {
                // Read each line from the file using the Scanner
                while (scanner.hasNextLine()) {
                    // Read the next line and print it to the console
                    String line = scanner.nextLine();
                    System.out.println(line);
                }
            } catch (Exception e) {
                // Handle exceptions, if any
                System.out.println("Error reading file " + filename + ": " +
e.getMessage());
            }
        }
    }
}
```

Sample1.txt

```
A computer is a machine that can store and process information.
Most computers rely on a binary system, which uses two variables,
0 and 1, to complete tasks such as storing data,
calculating algorithms, and displaying information.
```

Sample2.txt

```
The first computers were used primarily for numerical calculations.
However, as any information can be numerically encoded,
people soon realized that computers are capable of general-purpose information processing.
```

Sample3.txt

```
A microcomputer is a small computer built around a microprocessor integrated circuit, or
chip.
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Reading contents of Sample1.txt:
A computer is a machine that can store and process information.
Most computers rely on a binary system, which uses two variables,
0 and 1, to complete tasks such as storing data,
calculating algorithms, and displaying information.
Reading contents of Sample2.txt:
The first computers were used primarily for numerical calculations.
However, as any information can be numerically encoded,
people soon realized that computers are capable of general-purpose information
processing.
Reading contents of Sample3.txt:
A microcomputer is a small computer built around a microprocessor integrated circuit, or
chip.
```

S.No: 108	Exp. Name: ***Choose One File from Multiple Files***	**Date: 2023-12-16**

Aim:
Here you have three text files named **Simple1.txt**, **Simple2.txt**, and **Simple3.txt**. Your Java program, **SelectedFileEx**, allows a user to input the name of the file they want to read.

User Input:
The user then enters the name of the file they want to read. Let's say they enter **Simple1.txt** and press Enter.

Source Code:

```
q1576/SelectedFileEx.java
```

```
package q1576;
import java.io.File;
import java.util.Scanner;

public class SelectedFileEx {
    public static void main(String[] args) {
        // Scanner to read user input
        Scanner userInput = new Scanner(System.in);

        // Prompt the user to enter the file name
        System.out.println("Enter the file name to read:");

        // Read the file name entered by the user
        String fileName = userInput.nextLine();

        // Create a File object representing the input file
        File file = new File(fileName);

        // Check if the file exists
        boolean fileFound = file.exists();
        if (!fileFound) {
            System.out.println("File not found!");
        }

//         userInput.close();
//     }
// }
                else {
            // Print a message indicating the file being read
            System.out.println("Reading contents of " + fileName + ":");

            // Use a try-with-resources statement to automatically close the Scanner
            try (Scanner fileScanner = new Scanner(file)) {
                // Read each line from the file using the Scanner
                while (fileScanner.hasNextLine()) {
                    // Read the next line and print it to the console
                    String line = fileScanner.nextLine();
                    System.out.println(line);
                }
            } catch (Exception e) {
                // Handle exceptions, if any
                System.out.println("Error reading file " + fileName + ": " +
e.getMessage());
            }
                }
        userInput.close();
    }
}
```

Simple1.txt

```
Java is the name of a programming language created by Sun Microsystems.
This company was bought out by Oracle Corporation, which continues to keep it up to date.
It is designed to be platform-independent, which means that code written in Java can run on
any device or operating system that has a Java Virtual Machine (JVM) installed.
```

Simple2.txt

```
Java was developed to achieve five main goals. These are:
Java should be easy to learn, object-oriented, and distributed.
Java should be safe and strong.
Java should not depend on any computer architecture or platform.
Java should function well.
Java should be able to have an interpreter written for it (for better portability), support
parallelism, and use dynamic typing.
```

Simple3.txt

```
Java, which was called Oak when it was still being developed,
is object oriented, meaning it is based on objects that work together to make programs do
their jobs.
Java code looks similar to C, C++, or C#, but code written in those languages will not work
in Java in most cases without being changed.
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the file name to read:
Simple2.txt
Reading contents of Simple2.txt:
Java was developed to achieve five main goals. These are:
Java should be easy to learn, object-oriented, and distributed.
Java should be safe and strong.
Java should not depend on any computer architecture or platform.
Java should function well.
Java should be able to have an interpreter written for it (for better portability),
support parallelism, and use dynamic typing.
```

S.No: 109	Exp. Name: ***Select more than one File from the Document***	**Date: 2023-12-16**

Aim:
The program, **SelectedMultipleFilesEx**, which prompts the user to input filenames separated by spaces and then reads the contents of those specified files.

- The program checks each filename provided by the user against the predefined **fileNames** array (**Sample1.txt,Sample2.txt,Sample3.txt and Sample4.txt**).
- For example Among the user-input filenames, it finds matches for **Sample1.txt** and **Sample3.txt** in the predefined filenames list.
- Since **Sample5.txt** is not in the predefined list, it's ignored.
- For each matching filename (**Sample1.txt)**and **Sample3.txt)**, the program attempts to read and display the contents of the files.
- It successfully reads and prints the contents of **Sample1.txt** and **Sample3.txt** line by line.
- Since **Sample5.txt** was not found in the predefined filenames list, the program prints. **"No valid files found!"**

Source Code:

```
q1577/SelectedMultipleFilesEx.java
```

```
package q1577;
import java.io.File;
import java.util.ArrayList;
import java.util.Scanner;

public class SelectedMultipleFilesEx {
    public static void main(String[] args) {
        // Predefined filenames
        String[] predefinedFileNames = {"Sample1.txt", "Sample2.txt", "Sample3.txt", 
"Sample4.txt"};

        // Scanner to read user input
        Scanner userInput = new Scanner(System.in);

        // Prompt the user to enter the file names separated by spaces
        System.out.println("Enter the file names to read (separated by spaces):");

        // Read the file names entered by the user and split them into an array
        String input = userInput.nextLine();
        String[] userFileNames = input.split("\\s+");

        // Create a list to store valid files to read
        ArrayList<String> filesToRead = new ArrayList<>();

        // Check each user-provided filename against the predefined list
        for (String fileName : userFileNames) {
            for (String predefinedFileName : predefinedFileNames) {
                if (fileName.equals(predefinedFileName)) {
                    filesToRead.add(fileName);
                    break; // Move to the next user-provided filename
                }
            }
                }
        if (filesToRead.isEmpty()) {
            System.out.println("No valid files found!");
        } else {
            for (String fileName : filesToRead) {
                try {
                        // Create a File object representing the input file
                    File file = new File(fileName);

                    // Print a message indicating the file being read
                    System.out.println("Reading contents of " + fileName + ":");

                    // Use a try-with-resources statement to automatically close the Scanner
                    try (Scanner fileScanner = new Scanner(file)) {
                        // Read each line from the file using the Scanner
                        while (fileScanner.hasNextLine()) {
                            // Read the next line and print it to the console
                            String line = fileScanner.nextLine();
                            System.out.println(line);
                        }
                    }
                            }
            catch (Exception e) {
                }
            }
        }

        userInput.close();
    }
}
```

Sample1.txt

```
A class is a user-defined blueprint or prototype from which objects are created.
It represents the set of properties or methods that are common to all objects of one type.
Using classes, you can create multiple objects with the same behavior instead of writing
their code multiple times.
```

Sample2.txt

```
Data Abstraction may also be defined as the process of identifying only the required
characteristics of an object,
ignoring the irrelevant details. The properties and behaviors of an object differentiate it
from other objects of similar type and also help in classifying/grouping the object.
```

Sample3.txt

```
It is defined as the wrapping up of data under a single unit.
It is the mechanism that binds together the code and the data it manipulates.
Another way to think about encapsulation is that it is a protective shield
that prevents the data from being accessed by the code outside this shield.
```

Sample4.txt

```
Inheritance is an important pillar of OOP (Object Oriented Programming).
It is the mechanism in Java by which one class is allowed to inherit the features (fields
and methods) of another class.
We are achieving inheritance by using extends keyword. Inheritance is also known as “is-a”
relationship.
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the file names to read (separated by spaces):
Sample1.txt Sample3.txt
Reading contents of Sample1.txt:
A class is a user-defined blueprint or prototype from which objects are created.
It represents the set of properties or methods that are common to all objects of one type.

```
Reading contents of Sample3.txt:
It is defined as the wrapping up of data under a single unit.
It is the mechanism that binds together the code and the data it manipulates.
Another way to think about encapsulation is that it is a protective shield
that prevents the data from being accessed by the code outside this shield.
```

S.No: 110	Exp. Name: ***Print the Number of Lines from the Selected File***	**Date: 2023-12-16**

Aim:

This program illustrates how the program interacts with the user, allowing them to choose a file by number and subsequently determining and displaying the line count within that selected file.
The program asks the user to enter the number of the file they want to read.
The user enters a file number, for example, **2**

- The program identifies that the user choose **Sample2.txt**.
- It opens **Sample2.txt** and proceeds to count the number of lines within the file.
- The program finishes counting lines and displays the count.

.Sample Test Case:

```
Files available to read:
1. Sample1.txt
2. Sample2.txt
3. Sample3.txt
Enter the file number to read:
2
Number of lines in Sample2.txt: 4
```

Source Code:

```
q1578/FileLineCounter.java
```

```
// package q1578;
// import java.io.File;
// import java.io.FileNotFoundException;
// import java.util.Scanner;

// public class FileLineCounter {
//     public static void main(String[] args) {
//         String[] fileNames = {"Sample1.txt", "Sample2.txt", "Sample3.txt"};
```

```
//         fileScanner.close();

//                     System.out.println("Number of lines in " + selectedFile + ": " +
lineCount);
//                 } else {
//                     System.out.println("File not found!");
//                 }
//             } catch (FileNotFoundException e) {
//                 System.out.println("File not found or could not be read: " +
e.getMessage());
//             } catch (Exception e) {
//                 System.out.println("An error occurred: " + e.getMessage());
//             }
//         } else {
//             System.out.println("Invalid file number!");
//         }

//         userInput.close();
//     }
// }
```

```
package q1578;

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

public class FileLineCounter {
    public static void main(String[] args) {
        String[] fileNames = {"Sample1.txt", "Sample2.txt", "Sample3.txt"};

        // Display available files to the user
        System.out.println("Files available to read:");
        for (int i = 0; i < fileNames.length; i++) {
            System.out.println((i + 1) + ". " + fileNames[i]);
        }

        // Scanner to read user input
        Scanner userInput = new Scanner(System.in);

        // Prompt the user to enter the file number
        System.out.println("Enter the file number to read:");

        // Read the file number entered by the user
        if (userInput.hasNextInt()) {
            int fileNumber = userInput.nextInt();

            // Check if the file number is valid
            if (fileNumber >= 1 && fileNumber <= fileNames.length) {
                // Subtract 1 to match the array index
                String selectedFile = fileNames[fileNumber - 1];

                try {
                    // Create a File object representing the selected file
                    File file = new File(selectedFile);

                    // Check if the file exists
                    if (file.exists()) {
                        // Use a try-with-resources statement to automatically close the
Scanner
                        try (Scanner fileScanner = new Scanner(file)) {
                            // Count the number of lines in the file
                            int lineCount = 0;
                            while (fileScanner.hasNextLine()) {
                                fileScanner.nextLine();
                                lineCount++;
                            }

                            // Display the number of lines in the selected file
                            System.out.println("Number of lines in " + selectedFile + ": " +
lineCount);
                        }
                    } else {
                        System.out.println("File not found!");
                    }
e.getMessage());
                } catch (Exception e) {
                    System.out.println("An error occurred: " + e.getMessage());
                }
            } else {
                System.out.println("Invalid file number!");
            }
        } else {
            System.out.println("Invalid input! Please enter a valid file number.");
        }

        // Close the Scanner
        userInput.close();
    }
}
```

Sample1.txt

```
Instance methods are methods that require an object of its class to be created before it can
be called.
To invoke an instance method, we have to create an Object of the class in which the method
is defined.
```

Sample2.txt

```
Instance method can access the instance methods and instance variables directly.
Instance method can access static variables and static methods directly.
Static methods can access the static variables and static methods directly.
Static methods can’t access instance methods and instance variables directly.
```

Sample3.txt

```
OOPS concepts are as follows:
Class
Object
Method and method passing
Pillars of OOPs
Abstraction
Encapsulation
Inheritance
Polymorphism
Compile-time polymorphism
Runtime polymorphism
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Files available to read:
1. Sample1.txt
```

```
2. Sample2.txt
3. Sample3.txt
Enter the file number to read:
1
Number of lines in Sample1.txt: 2
```

S.No: 111	Exp. Name: ***Count number of Words from Selected File***	Date: 2023-12-16

Aim:

Write the program for How the program allows users to choose a file by its number and then calculates and displays the count of words within that selected file.

- For example The program validates the input and identifies that the user selected **Simple2.txt**.
- It opens and reads **Simple2.txt** counting the number of words within the file.
- It finishes counting words and displays the count:

Note: use **FileNotFoundException**

Sample Test Case:

```
Files available to read:
1. Simple1.txt
2. Simple2.txt
3. Simple3.txt
Enter the file number to read:
3
Number of words in Simple3.txt: 45
```

Source Code:

```
q1579/FileWordCounter.java
```

```
// package q1579;
// import java.io.File;
// import java.io.FileNotFoundException;
// import java.util.Scanner;

// public class FileWordCounter {
//     public static void main(String[] args) {
//         String[] fileNames = {"Simple1.txt", "Simple2.txt", "Simple3.txt"};



//         catch (FileNotFoundException e) {
//                 System.out.println("File not found or could not be read: " +
e.getMessage());
//             } catch (Exception e) {
//                 System.out.println("An error occurred: " + e.getMessage());
//             }
//         } else {
//             System.out.println("Invalid file number!");
//         }

//         userInput.close();
//     }
// }

package q1579;

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

public class FileWordCounter {
    public static void main(String[] args) {
        String[] fileNames = {"Simple1.txt", "Simple2.txt", "Simple3.txt"};

        // Display available files to the user
        System.out.println("Files available to read:");
        for (int i = 0; i < fileNames.length; i++) {
            System.out.println((i + 1) + ". " + fileNames[i]);
        }

        // Scanner to read user input
        Scanner userInput = new Scanner(System.in);

        // Prompt the user to enter the file number
        System.out.println("Enter the file number to read:");

        // Read the file number entered by the user
        if (userInput.hasNextInt()) {
            int fileNumber = userInput.nextInt();

            // Check if the file number is valid
            if (fileNumber >= 1 && fileNumber <= fileNames.length) {
                // Subtract 1 to match the array index
                String selectedFile = fileNames[fileNumber - 1];
```

```
                    File file = new File(selectedFile);

                    // Check if the file exists
                    if (file.exists()) {
                        // Use a try-with-resources statement to automatically close the
Scanner
                        try (Scanner fileScanner = new Scanner(file)) {
                            // Count the number of words in the file
                            int wordCount = 0;
                            while (fileScanner.hasNext()) {
                                fileScanner.next(); // Move to the next word
                                wordCount++;
                            }

                            // Display the number of words in the selected file
                            System.out.println("Number of words in " + selectedFile + ": " +
wordCount);
                        }
                    } else {
                        System.out.println("File not found!");
                    }
                } catch (FileNotFoundException e) {
                    System.out.println("File not found or could not be read: " +
e.getMessage());
                } catch (Exception e) {
                    System.out.println("An error occurred: " + e.getMessage());
                }
            } else {
                System.out.println("Invalid file number!");
            }
        } else {
            System.out.println("Invalid input! Please enter a valid file number.");
        }

        // Close the Scanner
        userInput.close();
    }
}
```

Simple1.txt

```
String is an immutable class which means a constant and cannot be changed once created and
if wish to change ,
we need to create an new object and even the functionality it provides like toupper,
tolower, etc all these return a new object ,
its not modify the original object. It is automatically thread safe.
```

Simple2.txt

```
All objects in Java are stored in a heap.
The reference variable is to the object stored in the stack area
or they can be contained in other objects which puts them in the heap area also.
```

Simple3.txt

```
To make Java more memory efficient, the concept of string literal is used.
By the use of the ‘new’ keyword, The JVM will create a new string object in the normal heap area
even if the same string object is present in the string pool.
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Files available to read:
1. Simple1.txt
2. Simple2.txt
3. Simple3.txt
Enter the file number to read:
1
Number of words in Simple1.txt: 56
```

S.No: 112	Exp. Name: ***Read the content of the file, print the no.of lines and count the no.of words.***	**Date: 2023-12-16**

Aim:
Assume you have a set of text files, each containing different types of data. You want to analyze these files to understand their word count, line count, and content for further processing. Here's how you might use this program
You have three text files named **"Sample1.txt**", "**Sample2.txt**", and "**Sample3.txt**".

- Upon running the program, it displays the available files for reading.
- The user inputs the number corresponding to the file they want to analyze.

Example Let's say the user selects "**Sample2.txt**" by entering '2' as the file number.

- The program then processes "**Sample2.txt**".
- It provides information:Word count in "**Sample2.txt**".Line count in "**Sample2.txt**".Displays the content of "**Sample2.txt**" without an extra empty line at the end.

Note: use **StringBuilder** for file content
Source Code:

```
q1585/FileWordCounter.java
```

```
// package q1585;
// import java.io.File;
// import java.io.FileNotFoundException;
// import java.util.Scanner;

// public class FileWordCounter {
//     public static void main(String[] args) {
//         String[] fileNames = {"Sample1.txt", "Sample2.txt", "Sample3.txt"};

//         System.out.println("Files available to read:");
//         for (int i = 0; i < fileNames.length; i++) {
//             System.out.println((i + 1) + ". " + fileNames[i]);
//         }

//                      catch (FileNotFoundException e) {
//                 System.out.println("File not found or could not be read: " +
e.getMessage());
//             } catch (Exception e) {
//                 System.out.println("An error occurred: " + e.getMessage());
//             }
//         } else {
//             System.out.println("Invalid file number!");
//         }

//         userInput.close();
//     }
// }
package q1585;

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

public class FileWordCounter {
    public static void main(String[] args) {
        String[] fileNames = {"Sample1.txt", "Sample2.txt", "Sample3.txt"};

        System.out.println("Files available to read:");
        for (int i = 0; i < fileNames.length; i++) {
            System.out.println((i + 1) + ". " + fileNames[i]);
        }

        // Scanner to read user input
        Scanner userInput = new Scanner(System.in);

        // Prompt the user to enter the file number
        System.out.println("Enter the file number to read:");

        // Read the file number entered by the user
        if (userInput.hasNextInt()) {
            int fileNumber = userInput.nextInt();

            // Check if the file number is valid
            if (fileNumber >= 1 && fileNumber <= fileNames.length) {
                // Subtract 1 to match the array index
                try {
                    // Create a File object representing the selected file
                    File file = new File(selectedFile);

                    // Check if the file exists
                    if (file.exists()) {
                        // Use a try-with-resources statement to automatically close the
Scanner
                        try (Scanner fileScanner = new Scanner(file)) {
                            // Count the number of words and lines in the file
                            int wordCount = 0;
                            int lineCount = 0;

                            // StringBuilder to store file content
                            StringBuilder fileContent = new StringBuilder();

                            while (fileScanner.hasNextLine()) {
                                String line = fileScanner.nextLine();
                                fileContent.append(line).append("\n");
                                wordCount += line.split("\\s+").length;
                                lineCount++;
                            }

                            // Display the information
                            System.out.println("Number of words in " + selectedFile + ": " +
wordCount);
                            System.out.println("Number of lines in " + selectedFile + ": " +
lineCount);
                            System.out.println("Content of " + selectedFile + ":\n" +
fileContent.toString());
                        }
                    } else {
                        System.out.println("File not found!");
                    }
                } catch (FileNotFoundException e) {
                    System.out.println("File not found or could not be read: " +
e.getMessage());
                } catch (Exception e) {
                    System.out.println("An error occurred: " + e.getMessage());
                }
            } else {
                System.out.println("Invalid file number!");
            }
        } else {
            System.out.println("Invalid input! Please enter a valid file number.");
        }

        // Close the Scanner
        userInput.close();
    }
}
```

Sample1.txt

```
Singleton class is a class whose only one instance can be created at any given time,
in one JVM. A class can be made singleton by making its constructor private.
```

Sample2.txt

```
The parameterized constructor in Java,
is the constructor which is capable of initializing
the instance variables with the provided values. In other words,
the constructors which take the arguments are called parameterized constructors.
```

Sample3.txt

```
default constructor is the one
which does not take any inputs.
In other words, default constructors are
the no argument constructors
which will be created by default in case
you no other constructor is defined by the user.
Its main purpose is to initialize
the instance variables with the default values.
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Files available to read:
1. Sample1.txt
2. Sample2.txt
3. Sample3.txt
Enter the file number to read:
3
Number of words in Sample3.txt: 51
Number of lines in Sample3.txt: 8
Content of Sample3.txt:
default constructor is the one
which does not take any inputs.
In other words, default constructors are
the no argument constructors
which will be created by default in case
you no other constructor is defined by the user.
Its main purpose is to initialize
the instance variables with the default values.
```

S.No: 113	Exp. Name: ***Using BufferedWriter Class***	**Date: 2023-12-16**

Aim:

Enter the text to write into the file

Write Java Program How the program **WriteToFile** interacts with the user, takes their input, and writes the provided data into a file named Simple.txt.

- The program takes the user's input data.
- It creates a file named **Simple.txt** in the same directory as the program (as specified in the code).
- The content provided by the user is written into the **Simple.txt** file.
- The program closes the writer after writing the data into the file.

note: use Using**BufferedWriter** class

Source Code:

q1580/WriteToFile.java

```java
package q1580;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class WriteToFile {
    public static void main(String[] args) {
        Scanner userInput = new Scanner(System.in);
        System.out.println("Enter the text to write into the file:");
        // Read user input
        String inputText = userInput.nextLine();

        // Specify the file name
        String fileName = "Simple.txt";

        // Create a File object
        File file = new File(fileName);

        // Use try-with-resources to automatically close resources
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {
            // Write the user input into the file
            writer.write(inputText);
            System.out.println("Data has been written to the file.");
        } catch (IOException e) {
            System.out.println("An error occurred while writing to the file: " +
e.getMessage());
        }

        // Close the Scanner
        userInput.close();
    }
  }
```

Simple.txt

```
Insert text here : 1702373271337
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the text to write into the file:
Codetantra,Hyderabad
Data has been written to the file.
```

S.No: 114	Exp. Name: ***Using writeString() method***	Date: 2023-12-16

Aim:

This Java program **UserInput** allows a user to input text which is then written into a file named "**Sample.txt**". After writing the content, it reads the file and displays the content back to the user.
fetch the path of file using **path.of("filename")**
To write the data use **writeString()** method here

Source Code:

UserInput.java

```java
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.Scanner;

public class UserInput {
    public static void main(String[] args) throws IOException {
        Scanner userInput = new Scanner(System.in);
        System.out.println("Enter the text to write into the file:");
        // Read user input
        String inputText = userInput.nextLine();

        // Specify the file name
        String fileName = "Sample.txt";

        // Use Path.of to get the Path object for the file
        Path filePath = Path.of(fileName);

        // Write the user input into the file using writeString method
        Files.writeString(filePath, inputText);

        System.out.println("Content written to the file:");

        // Read the content from the file using readString method
        String fileContent = Files.readString(filePath);

        // Display the content back to the user
        System.out.println(fileContent);

        // Close the Scanner
        userInput.close();
    }
}
```

Sample.txt

```
Insert text here : 1702379427695
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the text to write into the file:
java is opensource and free
Content written to the file:
java is opensource and free
```

Test Case - 2

User Output

```
Enter the text to write into the file:
JSE is used to create programs for desktop
computer
Content written to the file:
JSE is used to create programs for desktop computer
```

S.No: 115	Exp. Name: ***Using FileWriter Class***	**Date: 2023-12-16**

Aim:

Develop a simple user activity logger program that allows a user to log their actions along with timestamps into a file named "user_activity.doc".

- It begins by prompting the user to input their actions.
- The system awaits user input.
- The user interacts with the program by entering actions they are performing..
- The user can continue logging actions in a loop until they decide to exit.
- When the user inputs "exit", the program recognizes this as a signal to stop logging.
- It displays a message "Logging out..." and terminates the logging loop.
- For each action entered by the user, the program formats a log message indicating when the action was logged.
- The program includes a timestamp, followed by the user's action, in the format: "- Action: ".
- It appends new logs to the existing content in the file, ensuring a chronological record of user actions.
- Upon exiting the program, the "user_activity.doc" file contains a log of actions, providing a history of user activities during the program's runtime.

Sample test case:

```
Enter your action (or 'exit' to quit):
upload
Action logged: upload
Enter your action (or 'exit' to quit):
delete
Action logged: delete
Enter your action (or 'exit' to quit):
exit
Logging out...
```

Source Code:

```
q1582/UserActivityLogger.java
```

```java
package q1582;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class UserActivityLogger {
    private static final String LOG_FILE_NAME = "user_activity.doc";

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

        try (FileWriter fileWriter = new FileWriter(LOG_FILE_NAME, true)) {
            System.out.println("Enter your action (or 'exit' to quit):");

            while (true) {
                String userInput = scanner.nextLine();

                if ("exit".equalsIgnoreCase(userInput)) {
                    System.out.println("Logging out...");
                    break;
                }

                logActionToFile(fileWriter, userInput);
                System.out.println("Enter your action (or 'exit' to quit):");
            }
        } catch (IOException e) {
            System.out.println("An error occurred while logging activities: " +
e.getMessage());
        } finally {
            scanner.close();
        }
    }

    public static void logActionToFile(FileWriter fileWriter, String action) throws
IOException {

        System.out.println("Action logged: " + action);
        }
    public static void writeLogToFile(FileWriter fileWriter, String logMessage) throws
IOException {
        fileWriter.write(logMessage + "\n");
        fileWriter.flush();
    }
}
```

user_activity.doc

```
Insert text here : 1702391508794
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter your action (or 'exit' to quit):
exit
Logging out...
```

Test Case - 2

User Output

```
Enter your action (or 'exit' to quit):
read
Action logged: read
Enter your action (or 'exit' to quit):
write
Action logged: write
Enter your action (or 'exit' to quit):
exit
Logging out...
```

Test Case - 3

User Output

```
Enter your action (or 'exit' to quit):
write
Action logged: write
Enter your action (or 'exit' to quit):
loop
Action logged: loop
Enter your action (or 'exit' to quit):
quit
Action logged: quit
Enter your action (or 'exit' to quit):
exit
Logging out...
```

Test Case - 4

User Output

```
Enter your action (or 'exit' to quit):
upload
Action logged: upload
Enter your action (or 'exit' to quit):
delete
Action logged: delete
Enter your action (or 'exit' to quit):
```

exit

```
Logging out...
```

S.No: 116	Exp. Name: ***Using FileOutputStream Class***	**Date: 2023-12-16**

Aim:

Suppose you are working on a system that allows users to create custom notes or save important information about java quickly into a text file. Here's how this program might be used:

- The program starts and prompts the user to input a filename.
- The user enters **"simple.txt"** as the file name.
- Next, the program prompts the user to enter data, which in this case, could be personal notes on java
- The program takes the entered notes and uses to **FileOutputStream** write this information into the **"simple.txt"** file..
- Upon successful writing of the data, the program displays a confirmation message: **"Data written to simple.txt"**

Note: Using FileOutputStream Class

Sample Test Case:

```
Enter the file name:
simple.txt
Enter the data to write:
Java was developed by James Gosling
Data written to simple.txt
```

Source Code:

q1583/FileWriteExample.java

```
package q1583;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Scanner;

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

        System.out.println("Enter the file name:");
        String fileName = scanner.nextLine();
        System.out.println("Enter the data to write:");
            String data = scanner.nextLine();

            // Using try-with-resources to automatically close the FileOutputStream
            try (FileOutputStream fileOutputStream = new FileOutputStream(fileName)) {
                // Convert the data to bytes and write to the file
                fileOutputStream.write(data.getBytes());
                System.out.println("Data written to " + fileName);
                        }
                 catch (IOException e) {
            e.printStackTrace();
        } finally {
            scanner.close();
        }
    }
}
```

simple.txt

```
Insert text here : 1702448357437
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the file name:
simple.txt
Enter the data to write:
java is a high-level programming language
Data written to simple.txt
```

Test Case - 2

User Output

```
Enter the file name:
simple.txt
Enter the data to write:
java is secured
Data written to simple.txt
```

S.No: 117	Exp. Name: ***Write With DataOutputStream***	Date: 2023-12-16

Aim:

You're developing a simple data entry tool that allows users to input various types of data into a file using a **DataOutputStream.**

33. The program initializes and prompts the user to enter data to be stored in a file named **"Sample.txt".**
34. Users start entering data into the program via the console
35. As users continue entering data, the program uses a **DataOutputStream** to write this input into the **"Sample.txt"** file
36. To signify the end of data entry, the user types **"exit"**
37. Upon receiving the "exit" command, the program stops accepting input and closes the file streams, ensuring the written data is safely stored in the **"Sample.txt"** file.
38. Finally, the program displays a confirmation message: **"Data has been written to Sample.txt".**

Note: use **FileOutputStream** to read file name and
use **DataOutputStream.**

Sample Test Case:

```
Enter data to write to the file (type 'exit' to finish):
default constructor
parameterized constructor
exit
Data has been written to Sample.txt
```

Source Code:

q1584/DataOutputStreamEx.java

```
package q1584;
import java.io.*;

public class DataOutputStreamEx {
    public static void main(String[] args) {
        String filename = "Sample.txt";
        try (DataOutputStream dataOutputStream = new DataOutputStream(new
FileOutputStream(filename))) {

            System.out.println("Enter data to write to the file (type 'exit' to finish):");

            for (String arg : args) {
                // Write command-line argument to the file
                dataOutputStream.writeUTF(arg);
            }
                        System.out.println("Data has been written to " + filename);
                }
        catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Sample.txt

```
Insert text here : 1702439211214
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter data to write to the file (type 'exit' to finish):
Unchecked Exception
Checked Exception
Error
exit
Data has been written to Sample.txt
```

Test Case - 2

User Output

```
Enter data to write to the file (type 'exit' to finish):
Java Runtime Environment
Java Development Kit
Java Virtual Machine
exit
Data has been written to Sample.txt
```

Test Case - 3

User Output

```
Enter data to write to the file (type 'exit' to finish):
platform independence
memory management
safety
performance
exit
Data has been written to Sample.txt
```

Test Case - 4

User Output

```
Enter data to write to the file (type 'exit' to finish):
public
private
protocted
exit
Data has been written to Sample.txt
```