Java Assignment - 1

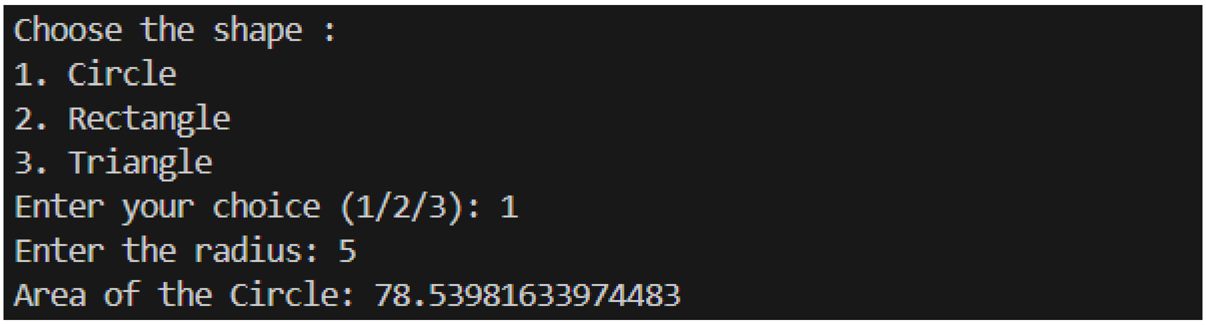
Basic Java:

1) Write a program to calculate the area of a circle, rectangle, or triangle based on user input.

CODE:

```
import java.util.Scanner;
public class Area{
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.println("Choose the shape to calculate area:");
System.out.println("1. Circle");
System.out.println("2. Rectangle");
System.out.println("3. Triangle");
System.out.print("Enter your choice (1/2/3): ");
int choice = sc.nextInt();
switch (choice) {
case 1:  // Circle
System.out.print("Enter the radius: ");
double radius = sc.nextDouble();
double circleArea = Math.PI * radius * radius;
System.out.println("Area of the Circle: " + circleArea);
break;

case 2:  // Rectangle
System.out.print("Enter the length: ");
double length = sc.nextDouble();
System.out.print("Enter the width: ");
double width = sc.nextDouble();
double rectangleArea = length * width;
System.out.println("Area of the Rectangle: " + rectangleArea);
break;

case 3:  // Triangle
System.out.print("Enter the base of the triangle: ");
double base = sc.nextDouble();
System.out.print("Enter the height of the triangle: ");
double height = sc.nextDouble();
double triangleArea = 0.5 * base * height;
System.out.println("Area of the Triangle: " + triangleArea);
break;
default:
System.out.println("Invalid choice!!!");
}
}
}
```

OUTPUT:



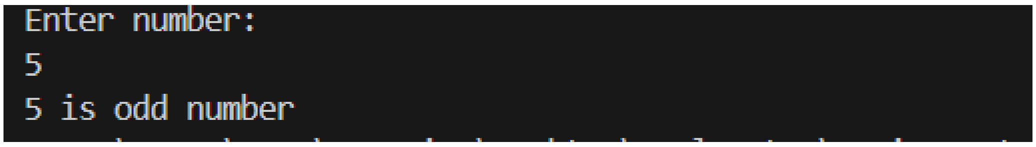
```
Choose the shape :
1. Circle
2. Rectangle
3. Triangle
Enter your choice (1/2/3): 1
Enter the radius: 5
Area of the Circle: 78.53981633974483
```

2) Create a program to check if a number is even or odd.

CODE:

```
import java.util.Scanner;
public class EvenOrOdd {
public static void main(String args[]){
Scanner sc = new Scanner(System.in);
System.out.println("Enter number:");
int num = sc.nextInt();
if(num%2 == 0)
System.out.println(num+ " is even number");
else{
System.out.println(num+ " is odd number");
}
}
}
```

OUTPUT:



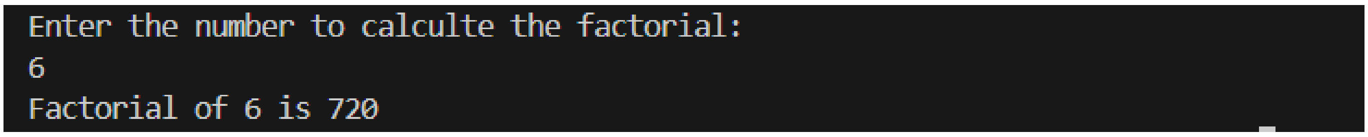
```
Enter number:
5
5 is odd number
```

3) Implement a program to find the factorial of a given number.

CODE:

```
import java.util.Scanner;
public class Factorial {
public static int fact(int num){
if(num == 0){
return 1;
}
else{
int fact = 1;
for(int i = 1;i<=num;i++){
fact = fact * i;
}
return fact;
}
}
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter the number to calculte the factorial:");
int num = sc.nextInt();
System.out.println("Factorial of "+num+" is "+fact(num));
}
}
```

OUTPUT:



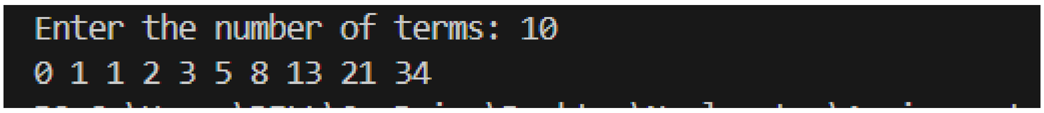
```
Enter the number to calculte the factorial:
6
Factorial of 6 is 720
```

4) Write a program to print the Fibonacci sequence up to a specified number.

CODE:

```
import java.util.Scanner;
public class Fibonacci{
public static void fib(int num){
if (num <= 0) {
System.out.println("Please enter a positive number.");
} else {
int first = 0, second = 1;
// Print first two terms
System.out.print(first + " " + second + " ");
for (int i = 3; i <= num; i++) {
int next = first + second;
System.out.print(next + " ");
first = second;
second = next;
}
}
}
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter the number of terms: ");
int n = sc.nextInt();
fib(n);
}
}
```

OUTPUT:



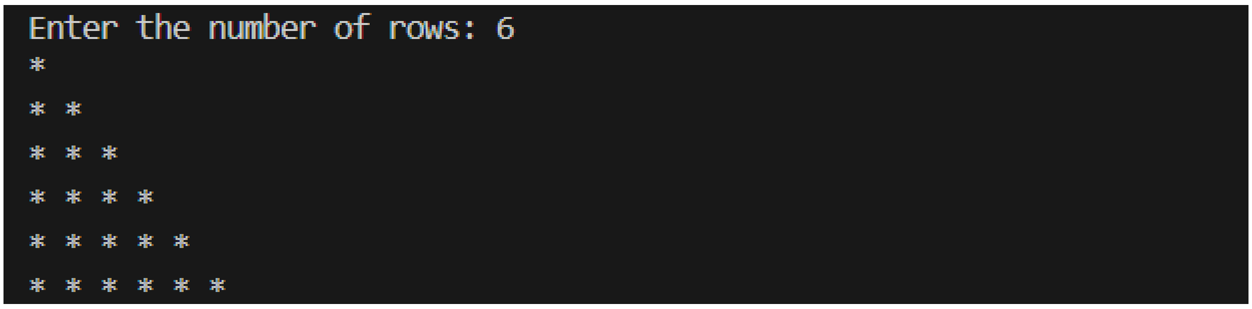
```
Enter the number of terms: 10
0 1 1 2 3 5 8 13 21 34
```

5) Use loops to print patterns like a triangle or square.

CODE:

```
import java.util.Scanner;
public class Pattern {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter the number of rows: ");
int rows = sc.nextInt();
for (int i = 1; i <= rows; i++) {
for (int j = 1; j <= i; j++) {
System.out.print("* ");
}
System.out.println(); // New line after each row
}
}
}
```

OUTPUT:



```
Enter the number of rows: 6
*
* *
* * *
* * * *
* * * * *
* * * * * *
```

Data Types and Operators:

1) Explain the difference between primitive and reference data types with examples.

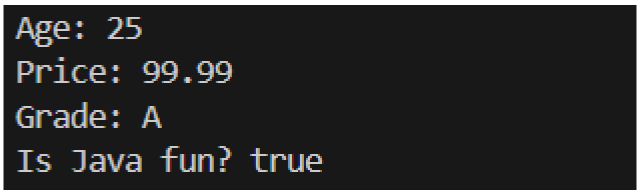
Primitive Data Types

- It stores simple values directly in memory.
- Stored as actual values (not references).
- Size is fixed and depends on the type.
- Faster in execution due to direct memory access.
- Cannot store complex objects but only single values.
- **Examples** – int, double, char, boolean etc.

CODE:

```
public class PrimitiveExample {
public static void main(String[] args) {
int age = 25;      // Integer type
double price = 99.99; // Floating-point type
char grade = 'A';  // Character type
boolean isJavaFun = true; // Boolean type

System.out.println("Age: " + age);
System.out.println("Price: " + price);
System.out.println("Grade: " + grade);
System.out.println("Is Java fun? " + isJavaFun);
}
}
```

OUTPUT:



```
Age: 25
Price: 99.99
Grade: A
Is Java fun? true
```

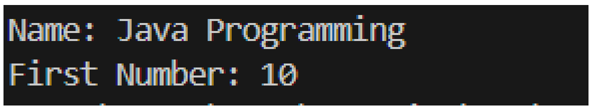
Reference Data Types

- It stores memory addresses that point to objects.
- Hold references to objects rather than actual values.
- Size varies based on the object stored.
- Slower compared to primitive types due to memory referencing.
- Can store complex objects like arrays, strings, and custom classes.
- **Example-** String, Array, Class, Interface

CODE:

```
public class ReferenceExample {
public static void main(String[] args) {
String name = "Java Programming"; // String is a reference type
int[] numbers = {10, 20, 30}; // Array is a reference type

System.out.println("Name: " + name);
System.out.println("First Number: " + numbers[0]);
}
}
```

OUTPUT:



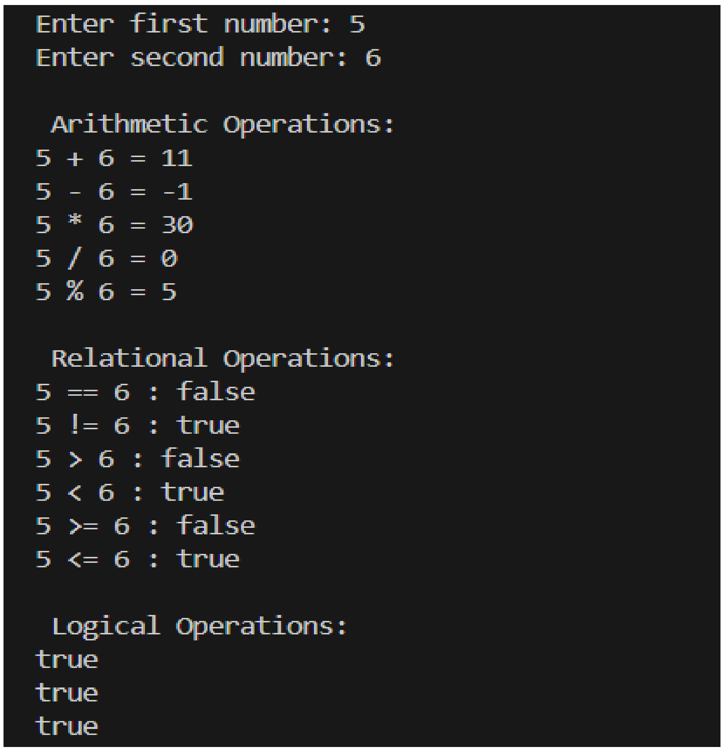
```
Name: Java Programming
First Number: 10
```

2) Write a program to demonstrate the use of arithmetic, logical, and relational operators.

CODE:

```
import java.util.Scanner;
public class OperatorsExample {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter first number: ");
int num1 = sc.nextInt();
System.out.print("Enter second number: ");
int num2 = sc.nextInt();

//  Arithmetic Operators
System.out.println("\n Arithmetic Operations:");
System.out.println(num1 + " + " + num2 + " = " + (num1 + num2));
System.out.println(num1 + " - " + num2 + " = " + (num1 - num2));
System.out.println(num1 + " * " + num2 + " = " + (num1 * num2));
System.out.println(num1 + " / " + num2 + " = " + (num1 / num2));
System.out.println(num1 + " % " + num2 + " = " + (num1 % num2));

//  Relational Operators
System.out.println("\n Relational Operations:");
System.out.println(num1 + " == " + num2 + " : " + (num1 == num2));
System.out.println(num1 + " != " + num2 + " : " + (num1 != num2));
System.out.println(num1 + " > " + num2 + " : " + (num1 > num2));
System.out.println(num1 + " < " + num2 + " : " + (num1 < num2));
System.out.println(num1 + " >= " + num2 + " : " + (num1 >= num2));
System.out.println(num1 + " <= " + num2 + " : " + (num1 <= num2));

//  Logical Operators
System.out.println("\n Logical Operations:");
//And operator
boolean condition1 = (num1 > 0 && num2 > 0);
System.out.println( condition1);

//Or operator
boolean condition2 = (num1 % 2 == 0 || num2 % 2 == 0);
System.out.println(condition2);

//Not operator
boolean condition3 = !(num1 == num2);
System.out.println(condition3);
}
}
```

OUTPUT:



```
Enter first number: 5
Enter second number: 6

 Arithmetic Operations:
5 + 6 = 11
5 - 6 = -1
5 * 6 = 30
5 / 6 = 0
5 % 6 = 5

 Relational Operations:
5 == 6 : false
5 != 6 : true
5 > 6 : false
5 < 6 : true
5 >= 6 : false
5 <= 6 : true

 Logical Operations:
true
true
true
```

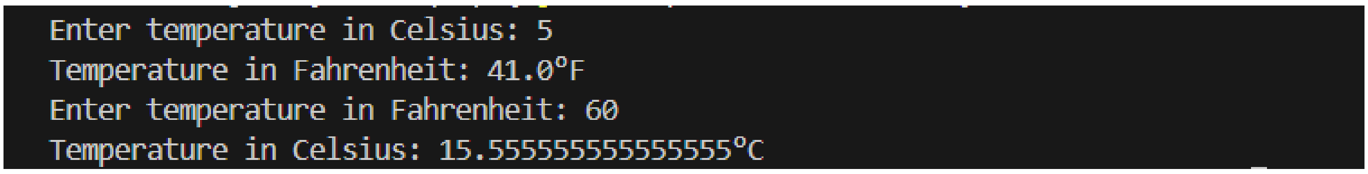
3) Create a program to convert a temperature from Celsius to Fahrenheit and vice versa.

CODE:

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

```
public class TemperatureConverter{

// Function to convert Celsius to Fahrenheit
public static double celsiusToFahrenheit(double celsius) {
return (celsius * 9 / 5) + 32;
}

// Function to convert Fahrenheit to Celsius
public static double fahrenheitToCelsius(double fahrenheit) {
return (fahrenheit - 32) * 5 / 9;
}

public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter temperature in Celsius: ");
double celsius = sc.nextDouble();
System.out.println("Temperature in Fahrenheit: " + celsiusToFahrenheit(celsius) + "°F");
System.out.print("Enter temperature in Fahrenheit: ");
double fahrenheit = sc.nextDouble();
System.out.println("Temperature in Celsius: " + fahrenheitToCelsius(fahrenheit) + "°C");
}
}
```

OUTPUT:



```
Enter temperature in Celsius: 5
Temperature in Fahrenheit: 41.0°F
Enter temperature in Fahrenheit: 60
Temperature in Celsius: 15.555555555555555°C
```

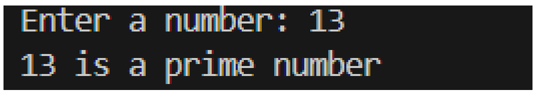
Control Flow Statements:

1) Write a program to check if a given number is prime using an if-else statement.

CODE:

```
import java.util.Scanner;
public class Prime{
public static boolean isPrime(int num) {
if (num <= 1) {
return false;
}
for (int i = 2; i <= num / 2; i++) {
if (num % i == 0) {
return false;
}
}
return true;
}
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter a number: ");
int num = sc.nextInt();

// Check if the number is prime
if (isPrime(num)) {
System.out.println(num + " is a prime number");
} else {
System.out.println(num + " is not a prime number");
}
}
}
```

OUTPUT:



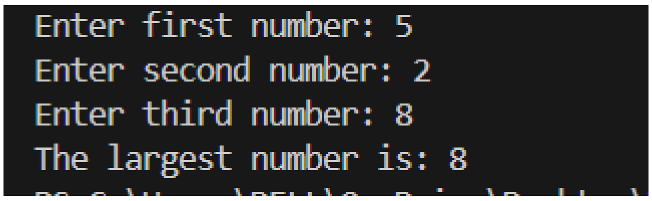
```
Enter a number: 13
13 is a prime number
```

2) Implement a program to find the largest number among three given numbers using a conditional statement.

CODE:

```
import java.util.Scanner;
public class LargestNum{
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter first number: ");
int num1 = sc.nextInt();
System.out.print("Enter second number: ");
int num2 = sc.nextInt();
System.out.print("Enter third number: ");
int num3 = sc.nextInt();
int largest;

// Using if-else to find the largest number
if (num1 >= num2 && num1 >= num3) {
largest = num1;
} else if (num2 >= num1 && num2 >= num3) {
largest = num2;
} else {
largest = num3;
}
System.out.println("The largest number is: " + largest);
}
}
```

OUTPUT:



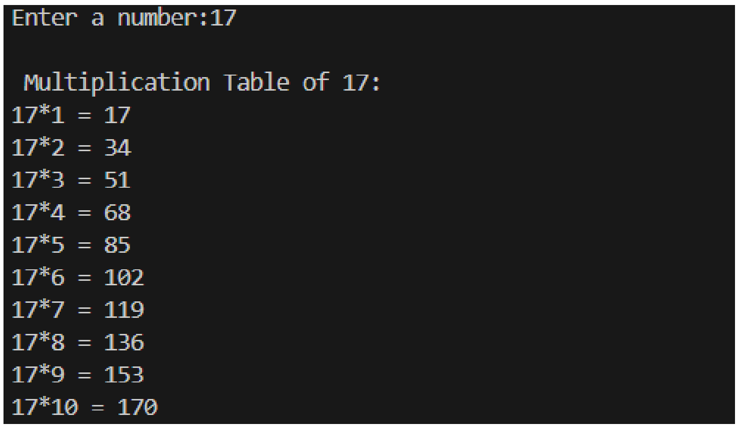
```
Enter first number: 5
Enter second number: 2
Enter third number: 8
The largest number is: 8
```

3) Use a for loop to print a multiplication table.

CODE:

```
import java.util.Scanner;
public class MultiplicationTable {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter a number:");
int num = sc.nextInt();
// print the multiplication table
System.out.println("\n Multiplication Table of " + num + ":");
for (int i = 1; i <= 10; i++) {
System.out.println(num + "*" + i + " = " + (num * i));
}
}
}
```

OUTPUT:



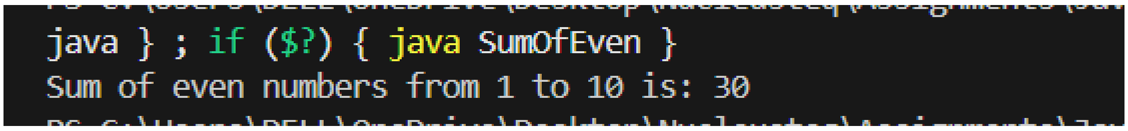
```
Enter a number:17

 Multiplication Table of 17:
17*1 = 17
17*2 = 34
17*3 = 51
17*4 = 68
17*5 = 85
17*6 = 102
17*7 = 119
17*8 = 136
17*9 = 153
17*10 = 170
```

4) Create a program to calculate the sum of even numbers from 1 to 10 using a while loop.

CODE:

```
public class SumOfEven{
public static void main(String[] args) {
int num = 2;
int sum = 0;
while (num <= 10) {
sum += num;
num += 2;
}
System.out.println("Sum of even numbers from 1 to 10 is: " + sum);
}
}
```

OUTPUT:



```
java } ; if ($?) { java SumOfEven }
Sum of even numbers from 1 to 10 is: 30
```

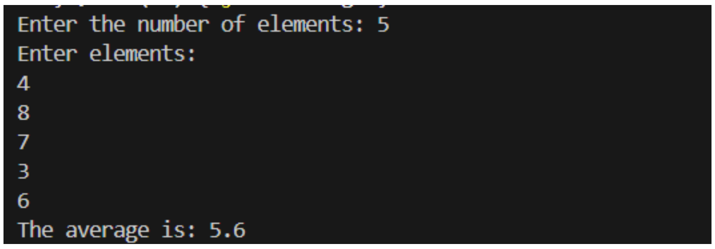
Arrays:

1) Write a program to find the average of elements in an array.

CODE:

```
import java.util.Scanner;
public class Average {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter the number of elements: ");
int size = sc.nextInt();
int[] arr = new int[size];
System.out.println("Enter elements:");
float sum = 0;
for (int i = 0; i < size; i++) {
arr[i] = sc.nextInt();
sum += arr[i]; // Calculating sum
}
float average = sum / size;    // Calculating the average
System.out.println("The average is: " + average);
}
}
```

OUTPUT:



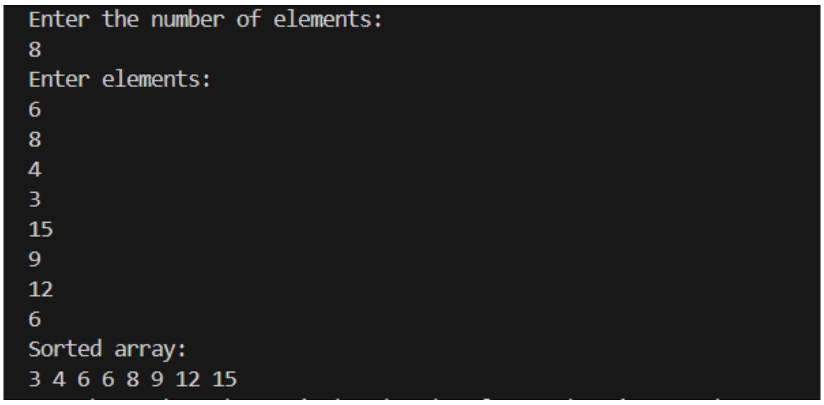
```
Enter the number of elements: 5
Enter elements:
4
8
7
3
6
The average is: 5.6
```

2) Implement a function to sort an array in ascending order using bubble sort or selection sort.

CODE:

```
import java.util.Scanner;
public class SortArray {
public static void bubbleSort(int[] arr) {
int n = arr.length;
for (int i = 0; i < n - 1; i++) {
for (int j = 0; j < n - i - 1; j++) {
if (arr[j] > arr[j + 1]) { // Swap if elements are in wrong order
int temp = arr[j];
arr[j] = arr[j + 1];
arr[j + 1] = temp;
}
}
}
}
public static void main(String[] args) {
Scanner sc= new Scanner(System.in);
System.out.print("Enter the number of elements: ");
int size = sc.nextInt();
int[] numbers = new int[size];
System.out.println("Enter elements:");
for (int i = 0; i < size; i++) {
numbers[i] = sc.nextInt();
}

// Sort the array
bubbleSort(numbers);

// Displaying the sorted array
System.out.println("Sorted array:");
for (int num : numbers) {
System.out.print(num + " ");
}
}
}
```

OUTPUT:



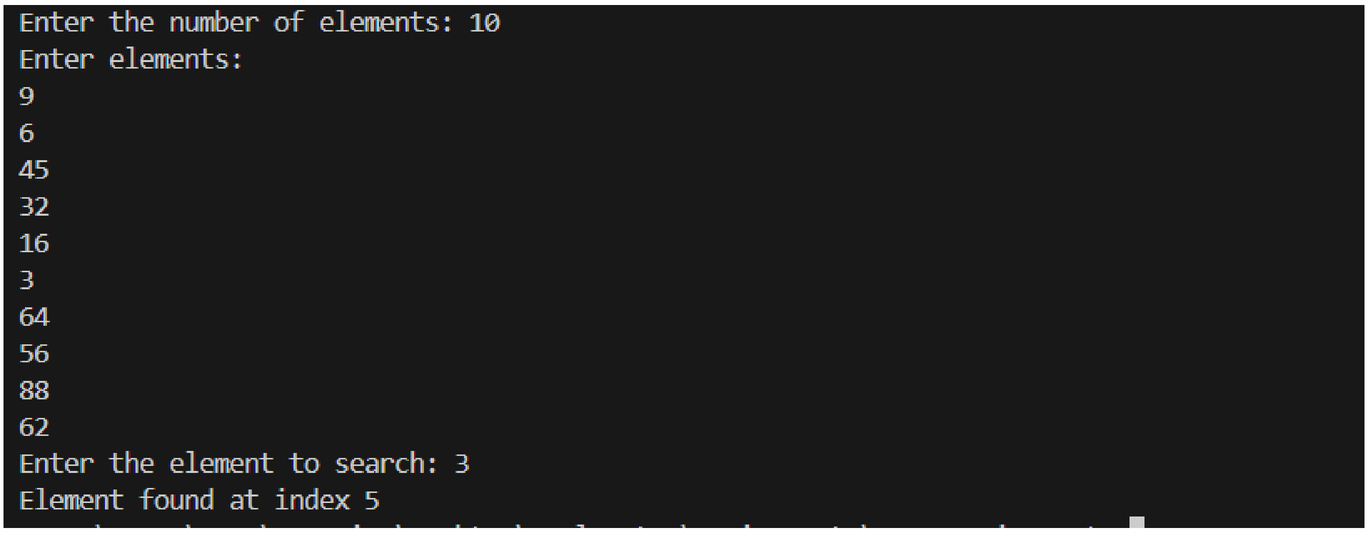
```
Enter the number of elements:
8
Enter elements:
6
8
4
3
15
9
12
6
Sorted array:
3 4 6 6 8 9 12 15
```

3) Create a program to search for a specific element within an array using linear search.

CODE:

```
import java.util.Scanner;
public class LinearSearch {
public static int linearSearch(int[] arr, int key) {
for (int i = 0; i < arr.length; i++) {
if (arr[i] == key) {
return i; // Return index if element is found
}
}
return -1; // Return -1 if element is not found
}
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter the number of elements: ");
int size = sc.nextInt();
int[] numbers = new int[size];
System.out.println("Enter elements:");
for (int i = 0; i < size; i++) {
numbers[i] = sc.nextInt();
}
// Taking the element to search
System.out.print("Enter the element to search: ");
int key = sc.nextInt();
// Searching the element using linear search
int result = linearSearch(numbers, key);
// Displaying the result
if (result != -1) {
System.out.println("Element found at index " + result);
} else {
System.out.println("Element not found in the array");
}
}
}
```

OUTPUT:



```
Enter the number of elements: 10
Enter elements:
9
6
45
32
16
3
64
56
88
62
Enter the element to search: 3
Element found at index 5
```

Object Oriented Programming (OOP):

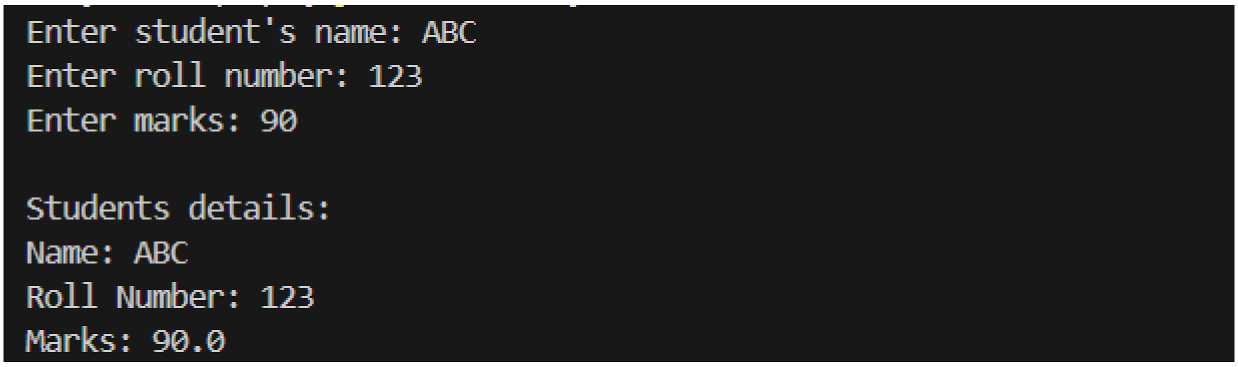
1) Create a class to represent a student with attributes like name, roll number, and marks.

CODE:

```
import java.util.Scanner;
class Student {
// Attributes
private String name;
private int rollNumber;
private double marks;
// Constructor
public Student(String name, int rollNumber, double marks) {
this.name = name;
this.rollNumber = rollNumber;
this.marks = marks;
}
// Method to display student details
public void display() {
System.out.println("\nStudents details:");
System.out.println("Name: " + name);
System.out.println("Roll Number: " + rollNumber);
System.out.println("Marks: " + marks);
}
}
public class Main{
public static void main(String[] args) {
```

```
Scanner sc = new Scanner(System.in);
System.out.print("Enter student's name: ");
String name = sc.nextLine();
System.out.print("Enter roll number: ");
int rollNumber = sc.nextInt();
System.out.print("Enter marks: ");
double marks = sc.nextDouble();
// Creating Student object
Student student = new Student(name, rollNumber, marks);
// Displaying student details
student.display();
}
}
```

OUTPUT:



```
Enter student's name: ABC
Enter roll number: 123
Enter marks: 90

Students details:
Name: ABC
Roll Number: 123
Marks: 90.0
```

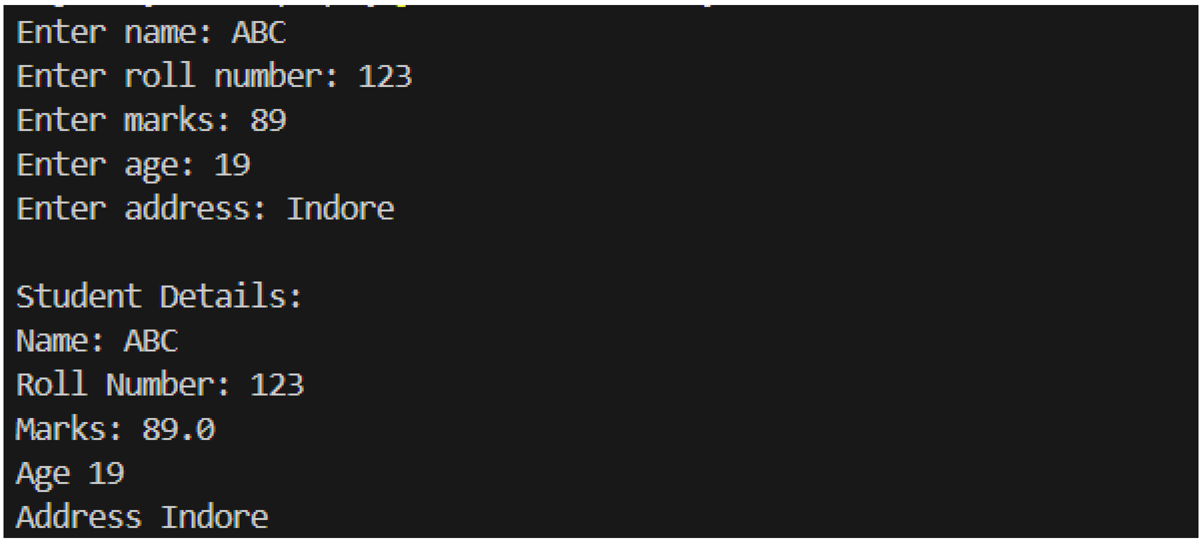
2) Implement inheritance to create a "GraduateStudent" class that extends the "Student" class with additional features.

CODE:

```
import java.util.Scanner;
// Base class
class Student {
// Attributes
private String name;
private int rollNumber;
private double marks;

// Constructor
public Student(String name, int rollNumber, double marks) {
this.name = name;
this.rollNumber = rollNumber;
this.marks = marks;
}
// Method to display student details
public void display() {
System.out.println("\nStudent Details:");
System.out.println("Name: " + name);
System.out.println("Roll Number: " + rollNumber);
System.out.println("Marks: " + marks);
}
}
// Derived class (Child)
class GraduateStudent extends Student {
private int age;
private String address;
// Constructor
public GraduateStudent(String name, int rollNumber, double marks,int age,String address) {
super(name, rollNumber, marks); // Calling the parent class constructor
this.age = age;
this.address = address;
}
// Method to display Graduate Student details
public void displayInfo() {
super.display(); // Call base class method
System.out.println("Age: " + age);
System.out.println("Address: " + address);
}
}
public class Inheritance {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter name: ");
String name = sc.nextLine();
System.out.print("Enter roll number: ");
int rollNumber = sc.nextInt();
System.out.print("Enter marks: ");
double marks = sc.nextDouble();
```

```
System.out.print("Enter age: ");
int age = sc.nextInt();
sc.nextLine();
System.out.print("Enter address: ");
String address= sc.nextLine();
// Creating GraduateStudent object
GraduateStudent st = new GraduateStudent(name, rollNumber, marks, age, address);
// Displaying graduate student details
st.displayInfo();
}
}
```

OUTPUT:



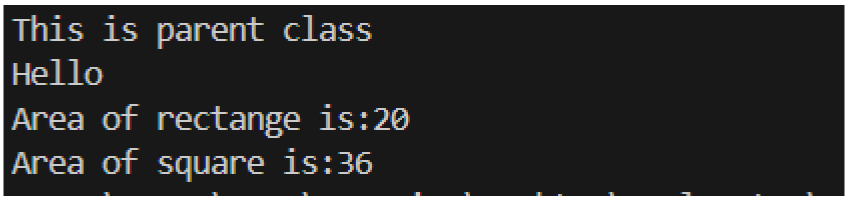
```
Enter name: ABC
Enter roll number: 123
Enter marks: 89
Enter age: 19
Enter address: Indore

Student Details:
Name: ABC
Roll Number: 123
Marks: 89.0
Age 19
Address Indore
```

3) Demonstrate polymorphism by creating methods with the same name but different parameters in a parent and child class.

CODE:

```
class Shape{
public void area(){
System.out.println("This is parent class");
}
public void area(String s){
System.out.println(s);
}
}
class Rectangle extends Shape{
public void area(int l,int b){
System.out.println("Area of rectange is:"+(l*b));
}
}
class Square extends Shape{
public void area(int a){
System.out.println("Area of square is:"+(a*a));
}
}
public class Polymorphism{
public static void main(String[] args) {
Shape s = new Shape();
s.area(); // Calls parent class method
s.area("Hello");
Rectangle r = new Rectangle();
r.area(4, 5); // Calls Rectangle's method
Square sq = new Square();
sq.area(6); // Calls Square's method
}
}
```

OUTPUT:



```
This is parent class
Hello
Area of rectange is:20
Area of square is:36
```

4) Explain the concept of encapsulation with a suitable example.

Encapsulation is the process of wrapping data (variables) and methods into a single unit (class). It restricts direct access to data by making fields `private` and provides controlled access through getter and setter methods.

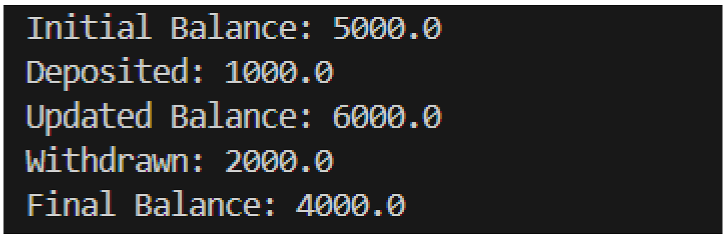
Key Features of Encapsulation

- **Data Hiding**: Fields are declared `private` so they cannot be accessed directly from outside the class.
- **Controlled Access**: Public getter and setter methods allow controlled modification of data.
- **Improved Security**: Prevents unauthorized access and modifications.
- **Code Maintainability**: Easier to update and manage code without affecting other parts.

CODE:

```
class BankAccount {

// Private variables (data hiding)
private String accountHolder;
private double balance;

// Constructor
public BankAccount(String accountHolder, double balance) {
this.accountHolder = accountHolder;
this.balance = balance;
}
// Getter method to access private variable
public double getBalance() {
return balance;
}
// Setter method to modify private variable
public void deposit(double amount) {
if (amount > 0) {
balance += amount;
System.out.println("Deposited: " + amount);
} else {
System.out.println("Invalid deposit amount!");
}
}
// Withdraw method with validation
public void withdraw(double amount) {
if (amount > 0 && amount <= balance) {
balance -= amount;
System.out.println("Withdrawn: " + amount);
} else {
System.out.println("Insufficient balance or invalid amount!");
}
}
}
public class EncapsulationExample {
public static void main(String[] args) {
// Creating an object of BankAccount
BankAccount myAccount = new BankAccount("John Doe", 5000.00);
// Accessing data using methods
System.out.println("Initial Balance: " + myAccount.getBalance());
myAccount.deposit(1000); // Depositing money
System.out.println("Updated Balance: " + myAccount.getBalance());
myAccount.withdraw(2000); // Withdrawing money
System.out.println("Final Balance: " + myAccount.getBalance());
}
}
```

OUTPUT:



```
Initial Balance: 5000.0
Deposited: 1000.0
Updated Balance: 6000.0
Withdrawn: 2000.0
Final Balance: 4000.0
```

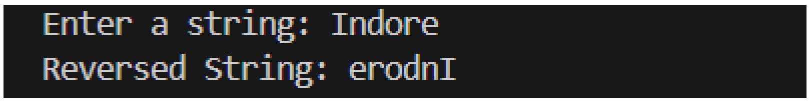
String Manipulation:

1) Write a program to reverse a given string.

CODE:

```
import java.util.Scanner;
public class StringReverse{
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter a string: ");
String str = sc.nextLine();
String reverse= "";
for (int i = str.length() - 1; i >= 0; i--) {
reverse += str.charAt(i);
}
// Displaying the reversed string
System.out.println("Reversed String: " + reverse);
}
}
```

OUTPUT:



```
Enter a string: Indore
Reversed String: erodnI
```

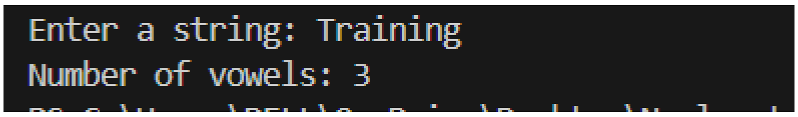
2) Implement a function to count the number of vowels in a string.

CODE:

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

```
public class VowelCount {
public static int countVowels(String str) {
int count = 0;
str = str.toLowerCase(); // Convert to lowercase
for (int i = 0; i < str.length(); i++) {
char ch = str.charAt(i);
if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
count++;
}
}
return count;
}
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter a string: ");
String str = sc.nextLine();
int vowelCount = countVowels(str);
System.out.println("Number of vowels: " + vowelCount);
}
}
```

OUTPUT:



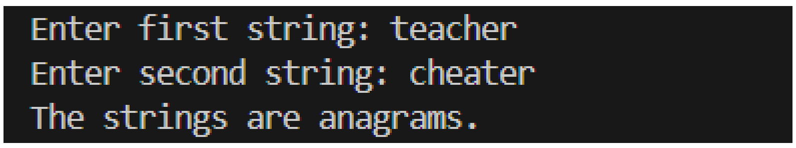
```
Enter a string: Training
Number of vowels: 3
```

3) Create a program to check if two strings are anagrams.

CODE:

```
import java.util.Arrays;
import java.util.Scanner;
public class Anagram{
public static boolean areAnagrams(String str1, String str2) {
// Convert to lower case
str1 = str1.toLowerCase();
str2 = str2.toLowerCase();

// If lengths are different, they can't be anagrams
if (str1.length() != str2.length()) {
return false;
}
// Convert strings to character arrays and sort them
char[] arr1 = str1.toCharArray();
char[] arr2 = str2.toCharArray();
Arrays.sort(arr1);
Arrays.sort(arr2);

// Compare sorted arrays
return Arrays.equals(arr1, arr2);
}
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter first string: ");
String str1 = sc.nextLine();
System.out.print("Enter second string: ");
String str2 = sc.nextLine();

// Check if they are anagrams
if (areAnagrams(str1, str2)) {
System.out.println("The strings are anagrams.");
} else {
System.out.println("The strings are not anagrams.");
}
}
}
```

OUTPUT:



```
Enter first string: teacher
Enter second string: cheater
The strings are anagrams.
```

Advanced Topics:

1) Explain the concept of interfaces and abstract classes with examples.

   Abstract Class

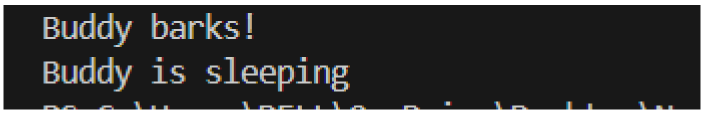
   - A class that contains the abstract keyword in its declaration is known as an abstract class.
   - Must have at least one abstract method (a method without implementation).
   - Can contain both abstract and concrete (non-abstract) methods.
   - Can have all four types of variables: static, non-static, final, and non-final.
   - Declared using the abstract keyword.

- Can have class members with private, protected, and public access modifiers.

CODE:

```
// Abstract class
abstract class Animal {
String name;
// Constructor
Animal(String name) {
this.name = name;
}
// Abstract method (without implementation)
abstract void Sound();
// Concrete method (with implementation)
void sleep() {
System.out.println(name + " is sleeping");
}
}
// Class extending the abstract class
class Dog extends Animal {
Dog(String name) {
super(name);
}
// Implementing the abstract method
void Sound() {
System.out.println(name + " barks!");
}
}
public class AbstractClass{
public static void main(String[] args) {
Dog myDog = new Dog("Buddy");
myDog.Sound(); // Calls the implemented method
myDog.sleep(); // Calls the inherited concrete method
}
}
```

OUTPUT:



```
Buddy barks!
Buddy is sleeping
```

Interface

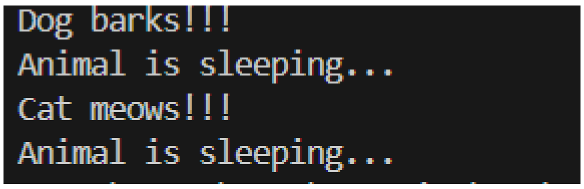
- A blueprint that is useful for implementing a class.
- Contains only abstract methods
- Can only have static and final variables.
- Declared using the interface keyword.
- Supports multiple inheritance
- By default, all members are public.

CODE:

```
// Interface
interface Animal {
// Abstract method
void makeSound();

// Default method
default void sleep() {
System.out.println("Animal is sleeping...");
}
}

// Implementing the interface in a class
class Dog implements Animal {
// Implementing the abstract method
public void makeSound() {
System.out.println("Dog barks!!!");
}
}

// Another class implementing the interface
class Cat implements Animal {
// Implementing the abstract method
public void makeSound() {
System.out.println("Cat meows!!!");
}
}
public class InterfaceExample {
public static void main(String[] args) {
Animal myDog = new Dog();
Animal myCat = new Cat();

myDog.makeSound(); // Calls the Dog class implementation
myDog.sleep();     // Calls the default method from the interface

myCat.makeSound(); // Calls the Cat class implementation
myCat.sleep();     // Calls the default method from the interface
}
```

}

OUTPUT:



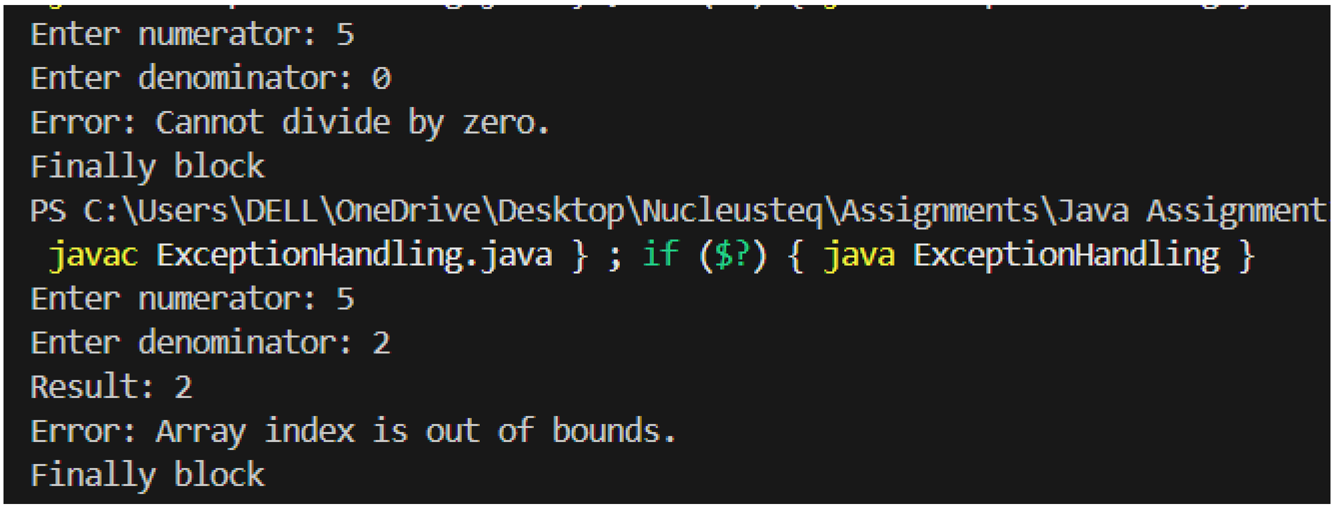
```
Dog barks!!!
Animal is sleeping...
Cat meows!!!
Animal is sleeping...
```

2) Create a program to handle exceptions using try-catch blocks.

CODE:

```
import java.util.Scanner;
public class ExceptionHandling{
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
try {
// Input from user
System.out.print("Enter numerator: ");
int numerator = sc.nextInt();
System.out.print("Enter denominator: ");
int denominator = sc.nextInt();

// Division operation (may throw ArithmeticException)
int result = numerator / denominator;
System.out.println("Result: " + result);

// Array example (may throw ArrayIndexOutOfBoundsException)
int[] numbers = {10, 20, 30};
System.out.println("Accessing 4th element: " + numbers[3]);
} catch (ArithmeticException e) {
System.out.println("Error: Cannot divide by zero.");
} catch (ArrayIndexOutOfBoundsException e) {
System.out.println("Error: Array index is out of bounds.");
} catch (Exception e) {
System.out.println(e.getMessage());
} finally {
System.out.println("Finally block");
}
}
}
```

OUTPUT:



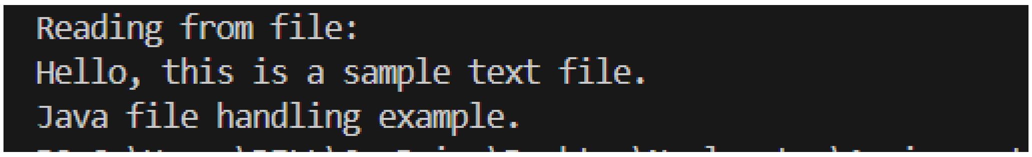
```
Enter numerator: 5
Enter denominator: 0
Error: Cannot divide by zero.
Finally block
PS C:\Users\DELL\OneDrive\Desktop\Nucleusteq\Assignments\Java Assignment
 javac ExceptionHandling.java } ; if ($?) { java ExceptionHandling }
Enter numerator: 5
Enter denominator: 2
Result: 2
Error: Array index is out of bounds.
Finally block
```

3) Implement a simple file I/O operation to read data from a text file.

CODE:

```
import java.io.*;
public class FileRead{
public static void main(String[] args) {
String fileName = "Test.txt"; // File to read
try {
// Create FileReader and BufferedReader
FileReader fileReader = new FileReader(fileName);
BufferedReader bufferedReader = new BufferedReader(fileReader);
String line;
System.out.println("Reading from file:");
while ((line = bufferedReader.readLine()) != null) { // Read line by line
System.out.println(line);
}
// Close
bufferedReader.close();
fileReader.close();
} catch (FileNotFoundException e) {
System.out.println("Error: File not found!");
} catch (IOException e) {
System.out.println("Error reading the file!");
}
}
}
```

OUTPUT:



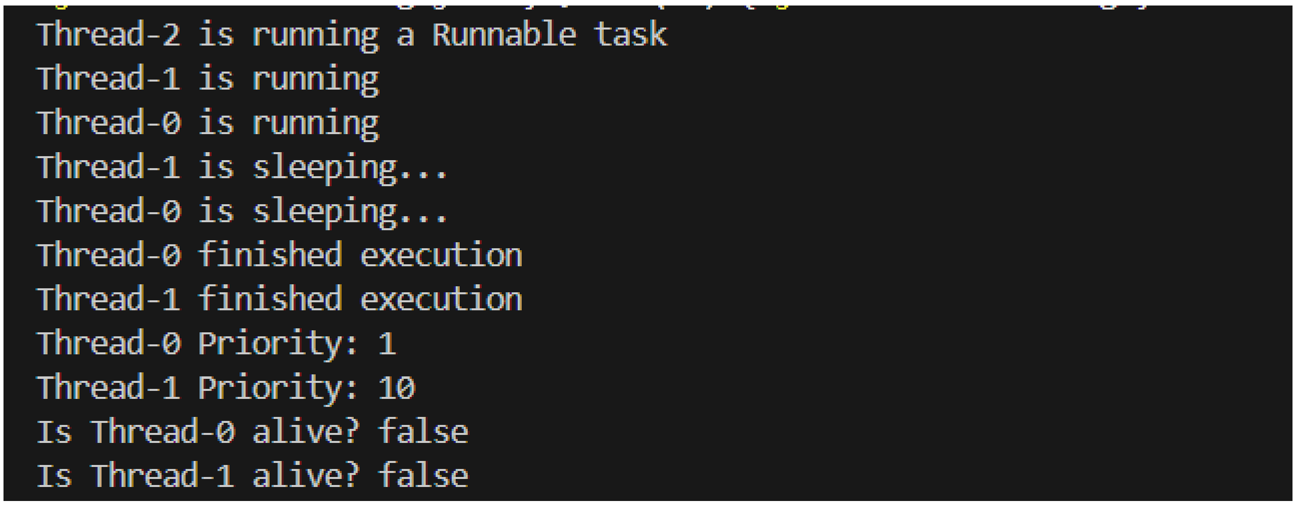
```
Reading from file:
Hello, this is a sample text file.
Java file handling example.
```

4) Explore multithreading in Java to perform multiple tasks concurrently.

CODE:

```
class MyThread extends Thread {
public void run() {
System.out.println(Thread.currentThread().getName() + " is running");
// Sleep method
try {
System.out.println(Thread.currentThread().getName() + " is sleeping...");
Thread.sleep(1000);
} catch (InterruptedException e) {
System.out.println(Thread.currentThread().getName() + " was interrupted!");
}
System.out.println(Thread.currentThread().getName() + " finished execution");
}
}
class MyRunnable implements Runnable {
public void run() {
System.out.println(Thread.currentThread().getName() + " is running a Runnable task");
}
}
public class Multithreading {
public static void main(String[] args) {
// Creating threads using Thread class
MyThread thread1 = new MyThread();
MyThread thread2 = new MyThread();

// Creating threads using Runnable interface
Thread thread3 = new Thread(new MyRunnable());

// Start the threads
thread1.start();
thread2.start();
thread3.start();

// join() method
try {
thread1.join(); // Main thread waits for thread1 to finish
} catch (InterruptedException e) {
System.out.println(e);
}

// Yield method
Thread.yield();

// Priority methods
thread1.setPriority(Thread.MIN_PRIORITY); // Priority 1
thread2.setPriority(Thread.MAX_PRIORITY); // Priority 10
System.out.println(thread1.getName() + " Priority: " + thread1.getPriority());
System.out.println(thread2.getName() + " Priority: " + thread2.getPriority());

// isAlive method
System.out.println("Is " + thread1.getName() + " alive? " + thread1.isAlive());
System.out.println("Is " + thread2.getName() + " alive? " + thread2.isAlive());

// Interrupt method
thread2.interrupt();
}
}
```

OUTPUT:



```
Thread-2 is running a Runnable task
Thread-1 is running
Thread-0 is running
Thread-1 is sleeping...
Thread-0 is sleeping...
Thread-0 finished execution
Thread-1 finished execution
Thread-0 Priority: 1
Thread-1 Priority: 10
Is Thread-0 alive? false
Is Thread-1 alive? false
```