Question Bank – OOPS Using Java – solved by Kanhu Charan

Unit – I

Short Answer Questions:

1. Describe the uses of parseInt() in Java programming.

Ans- parseInt() is a method provided by the Integer class that is used to parse a string representation of an integer and convert it into an actual integer value.

The general syntax for using parseInt() is as follows:

int intValue = Integer.parseInt(String str);

Parsing a String to an Integer:

String strNumber = "123";

int intValue = Integer.parseInt(strNumber);

In this example, the string "123" is converted to the integer 123.

2. Write the code to check the biggest among three numbers using the ternary operator.

Ans-

```
public class FindLargest {
    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 20;
        int num3 = 15;

        // Using ternary operator to find the largest number
        int largest = (num1 > num2) ? ((num1 > num3) ? num1 : num3) : ((num2 > num3) ? 
num2 : num3);

        System.out.println("The largest number among " + num1 + ", " + num2 + ", and " 
+ num3 + " is: " + largest);
    }
}
```

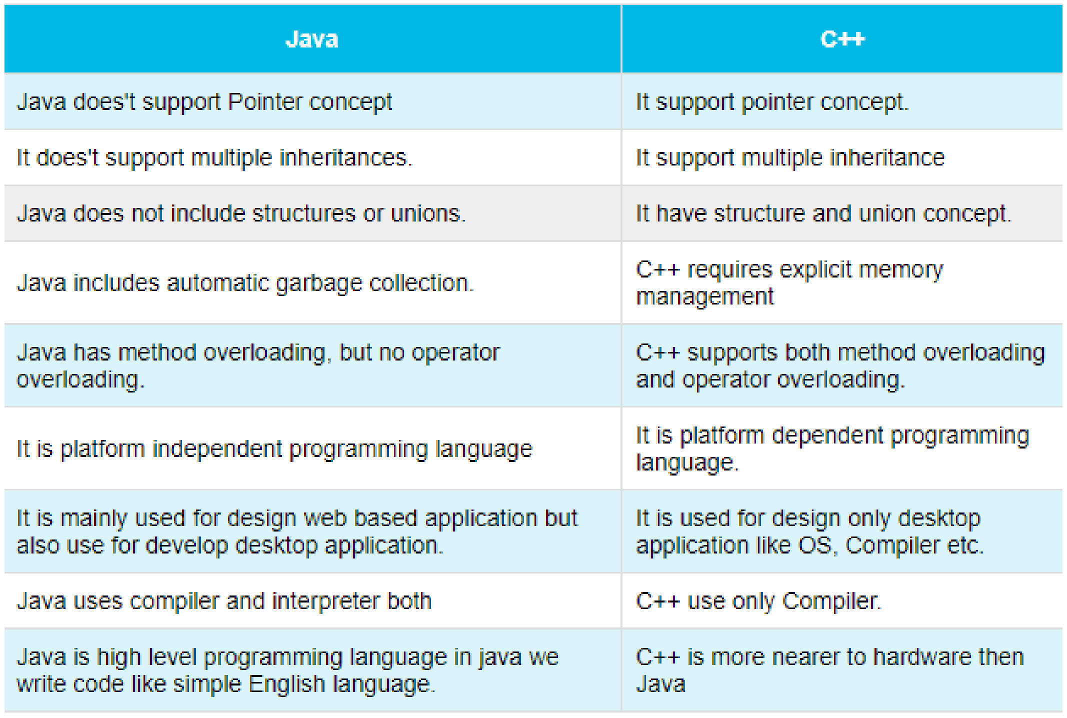
3. Why is Java called a platform-independent language?

Ans- Java is called a platform-independent language because it uses bytecode and the Java Virtual Machine (JVM). Code is compiled to bytecode, which can run on any system with a JVM, making Java applications portable across different platforms without the need for recompilation.

4. Why is the main() method always declared as static in Java?

Ans- The main() method in Java is always declared as `static` because it serves as the entry point for a program, and making it static allows it to be called by the JVM without the need to create an instance of the class. This ensures simplicity, consistency, and efficiency in program execution.

5. Write any two differences between C++ and Java.



Java	C++
Java does't support Pointer concept	It support pointer concept.
It does't support multiple inheritances.	It support multiple inheritance
Java does not include structures or unions.	It have structure and union concept.
Java includes automatic garbage collection.	C++ requires explicit memory management
Java has method overloading, but no operator overloading.	C++ supports both method overloading and operator overloading.
It is platform independent programming language	It is platform dependent programming language.
It is mainly used for design web based application but also use for develop desktop application.	It is used for design only desktop application like OS, Compiler etc.
Java uses compiler and interpreter both	C++ use only Compiler.
Java is high level programming language in java we write code like simple English language.	C++ is more nearer to hardware then Java

6. What is casting? Explain the need for typecasting with examples.

Ans- Casting in programming refers to the process of explicitly converting a value from one data type to another. This is necessary when you want to assign a value of one type to a variable of another type or when performing operations involving different data types. Casting can be broadly categorized into two types:
implicit casting (automatic) and explicit casting (manual or forced).

Implicit Casting:

Implicit casting, also known as automatic type conversion, is performed by the compiler when there is no loss of information in the conversion. For example:

```
int intValue = 42;
double doubleValue = intValue; // Implicit casting from int to double
```

In this example, the integer value 42 is implicitly cast to a double value. Since there is no loss of precision, the compiler handles the conversion automatically.

Explicit Casting:

Explicit casting, also known as narrowing or manual type conversion, is performed by the programmer when there may be a potential loss of information. For example:

```
double doubleValue = 3.14;
int intValue = (int) doubleValue; // Explicit casting from double to int
```

In this example, the double value 3.14 is explicitly cast to an integer. This involves removing the fractional part, and the programmer explicitly indicates the conversion with (int).

7. What is a jagged array? Explain with an example.

Ans- A jagged array is an array of arrays in which each element of the main array is an array itself. Unlike a multidimensional array, jagged arrays allow each row to have a different length. This flexibility is useful when dealing with irregular data structures.

Here's an example in Java to illustrate a jagged array:

```java
public class JaggedArrayExample {
    public static void main(String[] args) {
        // Declaration and instantiation of a jagged array
        int[][] jaggedArray = new int[3][];

        // Initializing individual arrays with different lengths
        jaggedArray[0] = new int[] {1, 2, 3};
        jaggedArray[1] = new int[] {4, 5, 6, 7};
        jaggedArray[2] = new int[] {8, 9};

        // Accessing and printing elements of the jagged array
        for (int i = 0; i < jaggedArray.length; i++) {
            for (int j = 0; j < jaggedArray[i].length; j++) {
                System.out.print(jaggedArray[i][j] + " ");
            }
            System.out.println(); // Move to the next line after each row
        }
    }
}
```

In this example:

int[][] jaggedArray = new int[3][]; declares a jagged array with three rows. However, the length of each row is not specified initially.

jaggedArray[0], jaggedArray[1], and jaggedArray[2] are then assigned as one-dimensional arrays with different lengths.

The nested for loops are used to iterate through the jagged array and print its elements.

The output of the program would be:

```
1 2 3
4 5 6 7
8 9
```

8. Assume A[2][3] is a matrix. Write to code and create a transpose matrix of it.

Ans –

```java
public class MatrixTranspose {
    public static void main(String[] args) {
        // Assuming A[2][3] is the original matrix
        int[][] A = {
            {1, 2, 3},
            {4, 5, 6}
        };

        // Finding the dimensions of the original matrix
        int rows = A.length;
        int columns = A[0].length;

        // Creating a new matrix B for the transpose
        int[][] B = new int[columns][rows];

        // Transposing the matrix A
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                B[j][i] = A[i][j];
            }
        }

        // Displaying the original and transposed matrices
        System.out.println("Original Matrix A:");
        displayMatrix(A);

        System.out.println("\nTransposed Matrix B:");
        displayMatrix(B);
    }

    // Utility method to display a matrix
    public static void displayMatrix(int[][] matrix) {
        for (int[] row : matrix) {
            for (int element : row) {
                System.out.print(element + " ");
            }
            System.out.println();
        }
    }
}
```

9. Write a program to initialize an integer array and print the sum and average of the array.

Ans-

```
public class ArraySumAndAverage {
    public static void main(String[] args) {
        // Initialize an integer array
        int[] numbers = {10, 5, 8, 12, 7};
        // Calculate the sum of the array elements
        int sum = 0;
        for (int number : numbers) {
            sum += number;
        }

        // Calculate the average of the array elements
        double average = (double) sum / numbers.length;

        // Print the sum and average
        System.out.println("Array Elements: "+java.util.Arrays.toString(numbers));
        System.out.println("Sum: " + sum);
        System.out.println("Average: " + average);
    }
}
```

Long Answer Questions:

1. Explain all the characteristics of "Object-oriented programming".

Ans- Object-oriented programming (OOP) is a programming paradigm that uses objects, which are instances of classes, to represent and manipulate data and operations. OOP is based on several fundamental characteristics that contribute to its principles and practices. Here are the main characteristics of Object-Oriented Programming:

1. Encapsulation:

- Definition: Encapsulation refers to the bundling of data (attributes) and methods (functions) that operate on the data into a single unit called a class.
- Purpose: It helps in hiding the internal details of the object and exposing only what is necessary. This promotes modularity and prevents unauthorized access to certain aspects of the object.

2. Abstraction:

- Definition: Abstraction is the process of simplifying complex systems by modeling classes based on the essential properties and behaviors they share.
- Purpose: It allows developers to focus on high-level concepts without getting into the nitty-gritty details. Abstract classes and interfaces provide a way to define abstract data types and their common functionalities.

3. Inheritance:

- Definition: Inheritance is a mechanism that allows a new class (subclass or derived class) to inherit properties and behaviors of an existing class (superclass or base class).
- Purpose: It promotes code reuse, extensibility, and the creation of a hierarchy of related classes. Subclasses can inherit and extend the functionality of their parent classes.

4. Polymorphism:

- Definition: Polymorphism allows objects of different types to be treated as objects of a common type. It includes method overloading and method overriding.
- Purpose: It enables flexibility and extensibility in the code. Polymorphism allows a single interface (method or operation) to be used for different types of objects, providing a way for code to be more adaptable to different scenarios.

5. Modularity:

- Definition: Modularity involves breaking down a program into smaller, manageable units or modules. Each class represents a module in OOP.
- Purpose: It simplifies the development process by dividing complex systems into smaller, more understandable units. Each module can be developed, tested, and modified independently, promoting code organization and maintenance.

These characteristics collectively contribute to the principles of OOP, such as encapsulation, inheritance, and polymorphism, leading to code that is modular, flexible, and easier to understand and maintain. OOP is widely used in software development due to its ability to model real-world entities and relationships effectively.

2. Write a Java program to read a number (using Scanner class) and test whether it is prime or not.

Ans-

```java
import java.util.Scanner;

public class PrimeNumberChecker {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        // Read the number from the user
        int number = scanner.nextInt();

        // Check if the number is prime
        boolean isPrime = isPrimeNumber(number);

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

    // Method to check if a number is prime
    private static boolean isPrimeNumber(int num) {
        if (num <= 1) {
            return false; // Numbers less than or equal to 1 are not prime
        }
        // Check for divisibility from 2 to the square root of the number
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) {
                return false; // If divisible, not a prime number
            }
        }

        return true; // If not divisible by any number, it is a prime number
    }
}
```

3. Briefly explain all the features of Java.

Ans-

Java is a versatile and widely-used programming language known for its platform independence, simplicity, and robustness. Here are some key features of Java:

1. Simple:

- Java was designed to be easy to learn and use. It eliminates complex features like operator overloading, explicit pointers, etc., making the language simple and straightforward.

2. Object-Oriented:

- Java follows the principles of object-oriented programming (OOP). It supports the creation of classes and objects, encapsulation, inheritance, and polymorphism, providing a modular and organized approach to software development.

3. Platform-Independent:

- One of the most significant features of Java is its platform independence. Java programs are compiled into bytecode, which can be executed on any device with a Java Virtual Machine (JVM). This "write once, run anywhere" capability allows Java applications to be highly portable.

4. Distributed Computing:

- Java supports distributed computing through its networking capabilities. It includes a rich set of libraries for developing distributed applications, making it well-suited for building networked and web-based applications.

5. Multithreading:

- Java has built-in support for multithreading, allowing the concurrent execution of multiple threads within a single program. This feature is crucial for developing efficient and responsive applications that can handle multiple tasks simultaneously.

6. Dynamic:

- Java is a dynamically extensible language. New methods and classes can be easily added, and existing ones can be modified or removed without disrupting existing code. This promotes adaptability and ease of maintenance.

7. Secure:

- Java provides a robust security model with features like bytecode verification, runtime security checks, and a secure class loader. This helps protect against various security threats, making Java suitable for developing secure applications, especially in web environments.

8. Robust:

- Java incorporates strong memory management, exception handling, and type checking, contributing to its robust nature. It minimizes the risk of runtime errors and crashes, making it a reliable choice for developing large-scale applications.

9. Architecture-Neutral:

- Java's architecture-neutral design allows compiled Java bytecode to be executed on any JVM, regardless of the underlying hardware and operating system. This makes Java adaptable to different environments without modification.

10. High Performance:

- Java provides high performance through the use of Just-In-Time (JIT) compilers, which translate bytecode into native machine code at runtime. While interpreted, Java bytecode is optimized for efficient execution.

11. Rich Standard Library:

- Java comes with a comprehensive standard library (Java API) that includes packages and classes for various tasks such as data structures, networking, I/O, GUI development, and more. This extensive library simplifies development and reduces the need for developers to write code from scratch.

These features collectively contribute to Java's popularity and versatility, making it a preferred choice for a wide range of application domains, including web development, mobile app development, enterprise systems, and more.

4. Write a program to read a number say ‘n’ (using command line arguments) and generate a Fibonacci series up to ‘n.

Ans-

```
public class FibonacciSeries {
   public static void main(String[] args) {
      // Parse the command line argument to get the value of 'n'
```

```
        int n = Integer.parseInt(args[0]);

        // Display the Fibonacci series up to 'n'
        System.out.println("Fibonacci series up to " + n + ":");
        for (int i = 0; i <= n; i++) {
            System.out.print(fibonacci(i) + " ");
        }
    }

    // Method to calculate the Fibonacci value for a given index
    private static int fibonacci(int index) {
        if (index <= 1) {
            return index;
        } else {
            return fibonacci(index - 1) + fibonacci(index - 2);
        }
    }
}
```

5. Differentiate between the Scanner class and Buffered Reader class in Java with a suitable example.

The Scanner class and BufferedReader class are both used for reading input in Java, but they have different features and use cases. Here's a brief differentiation between the two along with examples:

Scanner Class:

Usage:

The Scanner class is a part of the java.util package and is commonly used for parsing primitive data types and strings.

Ease of Use:

It provides convenient methods for reading different data types, making it easy to use for simple input parsing.

Tokenization

Scanner uses whitespace as the default delimiter and tokenizes input into separate parts.

Example

```
import java.util.Scanner;

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

        // Read an integer
        System.out.print("Enter an integer: ");
        int num = scanner.nextInt();
        System.out.println("You entered: " + num);

        // Read a string
```

```java
        System.out.print("Enter a string: ");
        String str = scanner.next();
        System.out.println("You entered: " + str);

        // Close the Scanner to release resources
        scanner.close();
    }
}
```

BufferedReader Class:

Usage:

The BufferedReader class is a part of the java.io package and is commonly used for reading characters from a character-based input stream.

Buffering:

It provides buffering of data, which can enhance performance by reducing the number of reads from the underlying input stream.

Read Line by Line:

BufferedReader is often used to read lines of text using the readLine() method.

Example:

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class BufferedReaderExample {
    public static void main(String[] args) throws IOException {
        // Create a BufferedReader object to read input
        BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));

        // Read an integer
        System.out.print("Enter an integer: ");
        int num = Integer.parseInt(reader.readLine());
        System.out.println("You entered: " + num);

        // Read a string
        System.out.print("Enter a string: ");
        String str = reader.readLine();
        System.out.println("You entered: " + str);

        // Close the BufferedReader to release resources
        reader.close();
    }
}
```

6. Describe JVM and explain and draw the Architecture of JVM.

The Java Virtual Machine (JVM) is a crucial component of the Java Runtime Environment (JRE) and plays a central role in executing Java bytecode. It provides a platform-independent abstraction over the hardware, allowing Java programs to run on any device or operating system that has a compatible JVM.

<u>Architecture of the JVM :</u>

The architecture of the JVM can be divided into several key components:

Class Loader:

- Responsible for loading Java classes into the JVM.
- Classes are typically loaded from the local file system or a network location.
- Classes are loaded as needed during runtime.

Class Area (Method Area):

- Stores class-level information such as class metadata, static fields, and constant pool.
- It is shared among all threads and is a part of the JVM's memory.

Heap:

- Memory area used for dynamic memory allocation.
- Objects and their instance variables are stored in the heap.
- The heap is divided into two main areas: the Young Generation and the Old Generation.

Stack:

- Each thread in the JVM has its own stack.
- It stores local variables, partial results, and control flow information.
- Divided into frames, each representing a method call.

Program Counter (PC) Register:

- Keeps track of the current instruction being executed by the thread.
- Each thread has its own PC register.

Native Method Stack:

- Contains native method information.
- It is used when executing native methods, which are written in languages other than Java.

Execution Engine:

- Responsible for executing Java bytecode.
- It interprets the bytecode or uses Just-In-Time (JIT) compilation to convert bytecode into native machine code for improved performance.

Java Native Interface (JNI):

- Allows Java code to call and be called by applications and libraries written in other languages (e.g., C, C++).
- Enables interaction between Java code and native applications.

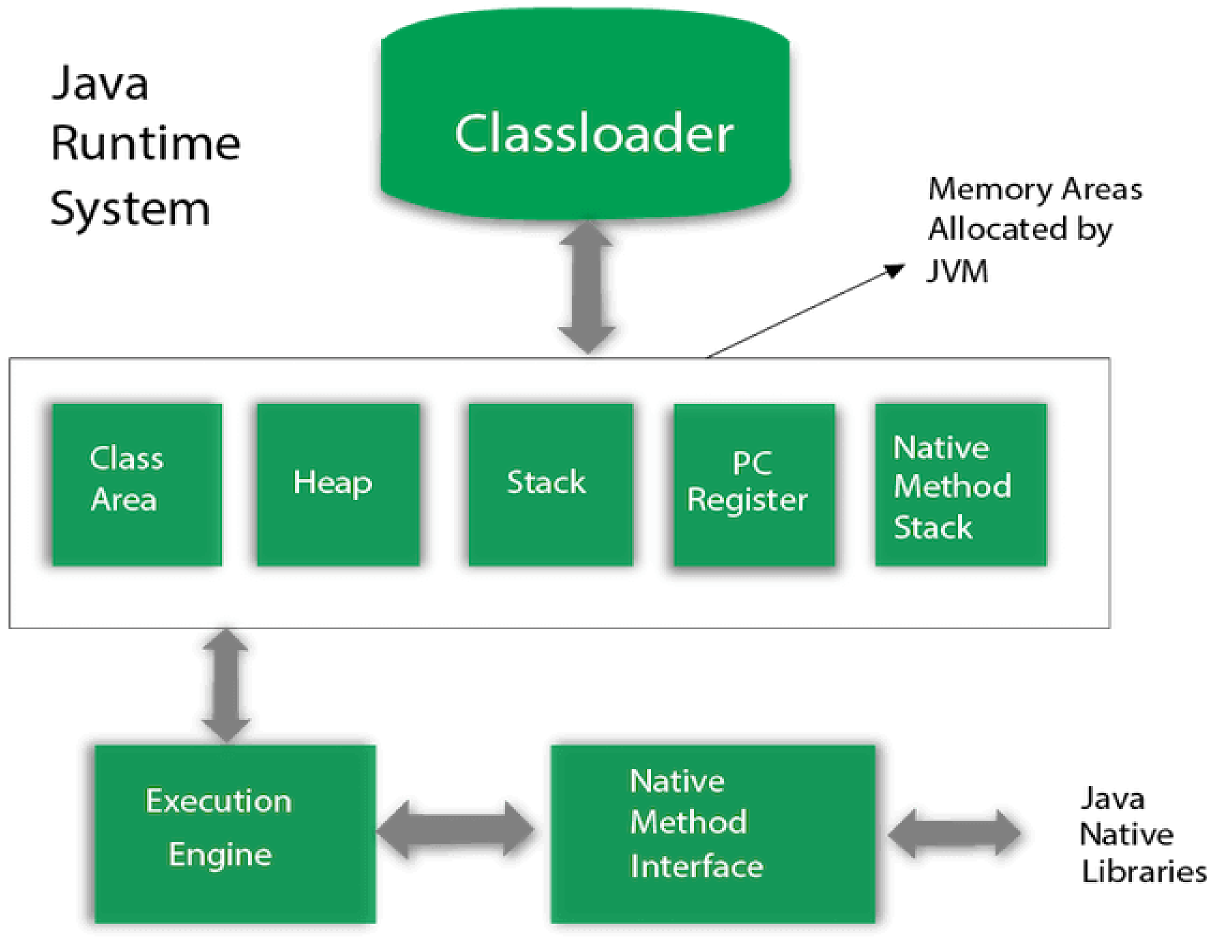
Native Method Interface (NMI):

- A part of the Native Method Stack.
- Allows the execution of native methods.

Direct Memory:

- Memory used by the JVM outside the Java heap.
- Managed explicitly by the application (not subject to garbage collection).

Here is a simplified diagram of the JVM architecture:



7. Write a Java program to create and display unique three-digit numbers using 1, 2, 3, and 4 and print a total count of three-digit numbers.

Ans-

```
public class UniqueThreeDigitNumbers {
       public static void main(String[] args) {
           int count = 0;

           System.out.println("Unique three-digit numbers using 1, 2, 3, and 4:");

           for (int i = 1; i <= 4; i++) {
               for (int j = 1; j <= 4; j++) {
                   for (int k = 1; k <= 4; k++) {
                       if (i != j && i != k && j != k) {
                           int threeDigitNumber = i * 100 + j * 10 + k;
                           System.out.println(threeDigitNumber);
                           count++;
                       }
                   }
               }
           }

           System.out.println("Total count of unique three-digit numbers: " +
   count);
       }
```

```
    }
```

8. Write a Java program to convert a binary number to a decimal number.

Ans-

```
import java.util.Scanner;

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

        // Input binary number from the user
        System.out.print("Enter a binary number: ");
        String binaryString = scanner.nextLine();

        // Validate the input as a binary number
        if (!isValidBinary(binaryString)) {
            System.out.println("Invalid binary number. Please enter a valid binary
number.");
            return;
        }

        // Convert binary to decimal
        int decimalNumber = Integer.parseInt(binaryString, 2);

        System.out.println("Decimal equivalent: " + decimalNumber);
    }

    // Method to check if the input is a valid binary number
    static boolean isValidBinary(String binaryString) {
        // A binary number should only contain 0s and 1s
        return binaryString.matches("[01]+");
    }
}
```

9. Demonstrate a Java program to accept a number at run time using scanner class and check whether the given number is Armstrong or not.

Ans-

```
import java.util.Scanner;

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

        // Input: Accept a number from the user
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();

        // Check if the number is an Armstrong number
        if (isArmstrongNumber(number)) {
            System.out.println(number + " is an Armstrong number.");
        } else {
            System.out.println(number + " is not an Armstrong number.");
        }
        scanner.close();
    }

    // Method to check if a number is an Armstrong number
    static boolean isArmstrongNumber(int num) {
        int originalNumber = num;
        int numberOfDigits = String.valueOf(num).length();
        int sum = 0;

        // Calculate the sum of each digit raised to the power of the number of digits
        while (num > 0) {
            int digit = num % 10;
            sum += Math.pow(digit, numberOfDigits);
            num /= 10;
        }

        // Check if the sum is equal to the original number
        return sum == originalNumber;
    }
}
```

10. Write a Java program to read a square matrix of 3x3 order and print the sum of all the elements of each row.

Ans-

```
import java.util.Scanner;

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

        // Input: Read a 3x3 matrix from the user
        System.out.println("Enter the elements of the 3x3 matrix:");

        int[][] matrix = new int[3][3];
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                matrix[i][j] = scanner.nextInt();
            }
        }

        // Calculate and print the sum of elements for each row
        System.out.println("Sum of elements in each row:");
        for (int i = 0; i < 3; i++) {
            int rowSum = 0;

            for (int j = 0; j < 3; j++) {
                rowSum += matrix[i][j];
            }

            System.out.println("Row " + (i + 1) + ": " + rowSum);
        }

        // Close the Scanner to prevent resource leak
        scanner.close();
    }
}
```

11. Write a program to read and convert seconds to hours, minutes, and seconds.

Ans-

```
import java.util.Scanner;

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

        // Input: Read the duration in seconds from the user
        System.out.print("Enter the duration in seconds: ");
        int totalSeconds = scanner.nextInt();

        // Calculate hours, minutes, and remaining seconds
        int hours = totalSeconds / 3600;
        int minutes = (totalSeconds % 3600) / 60;
        int seconds = totalSeconds % 60;

        // Display the result
        System.out.println("Duration in HH:MM:SS format:");
        System.out.printf("%02d:%02d:%02d%n", hours, minutes, seconds);

        // Close the Scanner to prevent resource leak
        scanner.close();
    }
}
```

12. Write a program to check if the program has received command line arguments or not. If the program has not received arguments then print "No Values", else print all the values in a single line separated by ,(comma).

Ans-

```
public class CommandLineArguments {
    public static void main(String[] args) {
        // Check if command line arguments are provided
        if (args.length == 0) {
            System.out.println("No Values");
        } else {
            // Print all values separated by commas
            System.out.print("Command Line Arguments: ");
            for (int i = 0; i < args.length; i++) {
                System.out.print(args[i]);
                // Add a comma if it's not the last argument
                if (i < args.length - 1) {
                    System.out.print(", ");
                }
            }
        }
    }
}
```

13. Write a program to receive a color code from the user (an Alphabhet). The program should then print the color name, based on the color code given. The following are the color codes and their corresponding color names. R->Red, B->Blue, G->Green, O>Orange, Y->Yellow, W->White. If color code provided by the user is not valid then print "Invalid Code".

Ans-

```
import java.util.Scanner;

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

        // Input: Read the color code from the user
        System.out.print("Enter a color code (R, B, G, O, Y, W): ");
        char colorCode = scanner.next().toUpperCase().charAt(0);

        // Check the color code and print the corresponding color name
        String colorName;
        switch (colorCode) {
            case 'R':
                colorName = "Red";
                break;
            case 'B':
                colorName = "Blue";
                break;
            case 'G':
                colorName = "Green";
                break;
            case 'O':
                colorName = "Orange";
                break;
            case 'Y':
                colorName = "Yellow";
                break;
            case 'W':
                colorName = "White";
                break;
            default:
                colorName = "Invalid Code";
        }

        // Display the result
        System.out.println("Color Name: " + colorName);

        // Close the Scanner to prevent resource leak
        scanner.close();
    }
}
```

14. Write a program to print first 5 values which are divisible by 2, 3, and 5.

Ans-

```
public class DivisibleNumbers {
    public static void main(String[] args) {
        int count = 0;
        int number = 1;

        System.out.println("First 5 values divisible by 2, 3, and 5:");

        while (count < 5) {
            if (isDivisibleBy235(number)) {
                System.out.println(number);
                count++;
            }
            number++;
        }
    }

    // Method to check if a number is divisible by 2, 3, and 5
    private static boolean isDivisibleBy235(int num) {
        return (num % 2 == 0) && (num % 3 == 0) && (num % 5 == 0);
    }
}
```

15. Write a program to initialize a character variable in a program and print 'Alphabhet' if the initialized value is an alphabet, print 'Digit' if the initialized value is a number, and print 'Special Character', if the initialized value is anything else.

Ans-

```
public class CharTypeChecker {
    public static void main(String[] args) {
        char initializedChar = 'A'; // You can change this to any character

        // Check if the initialized value is an alphabet
        if ((initializedChar >= 'A' && initializedChar <= 'Z') || (initializedChar
>= 'a' && initializedChar <= 'z')) {
            System.out.println("Alphabet");
        }
        // Check if the initialized value is a digit
```

```
            else if (initializedChar >= '0' && initializedChar <= '9') {
                System.out.println("Digit");
            }
            // If not an alphabet or digit, it is a special character
            else {
                System.out.println("Special Character");
            }
        }
    }
```
16. Write a program to print the sum of all the digits of a given number.

Ans-
```
import java.util.Scanner;

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

        // Input: Read a number from the user
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();

        // Calculate the sum of digits
        int sum = calculateSumOfDigits(number);

        // Display the result
        System.out.println("Sum of digits: " + sum);

        // Close the Scanner to prevent resource leak
        scanner.close();
    }

    // Method to calculate the sum of digits of a number
    private static int calculateSumOfDigits(int num) {
        int sum = 0;

        // Iterate through each digit and add it to the sum
        while (num > 0) {
            int digit = num % 10;
            sum += digit;
            num /= 10;
        }

        return sum;
    }
}
```

17. Write a program to find the sum of digits of a number until the sum becomes a single digit.

Ans-
```
import java.util.Scanner;
```

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

        // Input: Read a number from the user
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();

        // Calculate the single-digit sum
        int singleDigitSum = calculateSingleDigitSum(number);

        // Display the result
        System.out.println("Single-digit sum: " + singleDigitSum);

        // Close the Scanner to prevent resource leak
        scanner.close();
    }

    // Method to calculate the single-digit sum of a number
    private static int calculateSingleDigitSum(int num) {
        int sum = 0;

        // Iterate until the sum becomes a single digit
        while (num > 0 || sum > 9) {
            if (num == 0) {
                // If num becomes zero, reset num to sum and reset sum
                num = sum;
                sum = 0;
            }

            sum += num % 10;
            num /= 10;
        }

        return sum;
    }
}
```

18. Write a program to initialize an integer array with values and check if a given number is present in the array or not. If the number is not found, it will print -1 else it will print the index value of the given number in the array.

Ans-

```java
import java.util.Scanner;

public class NumberSearch {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Initialize an integer array with values
        int[] numbers = {10, 25, 30, 45, 50, 65, 70, 85, 90, 100};

        // Input: Read a number from the user
        System.out.print("Enter a number to search in the array: ");
        int targetNumber = scanner.nextInt();

        // Check if the number is present in the array
        int index = findIndex(numbers, targetNumber);

        // Display the result
        if (index != -1) {
            System.out.println("Number found at index: " + index);
        } else {
            System.out.println("Number not found in the array: -1");
        }

        // Close the Scanner to prevent resource leak
        scanner.close();
    }

    // Method to find the index of a number in an array
    private static int findIndex(int[] array, int target) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == target) {
                return i; // Return the index if the number is found
            }
        }
        return -1; // Return -1 if the number is not found
    }
}
```