1. Learning to work with Java IDE and Writing Simple Conversion Programs.

Aim: Write a program that converts temperature from Fahrenheit to Celsius using an IDE. (Formula Celsius temp = (Fahrenheit-32)* 5/9)

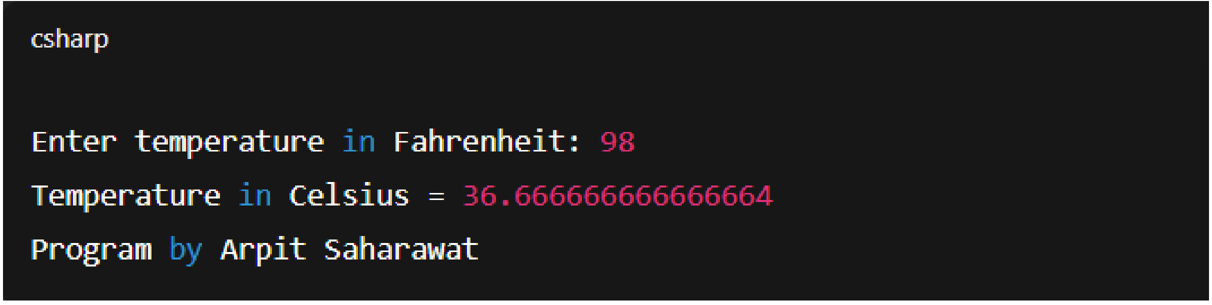
Procedure:

1. Start the program.
2. Input temperature in Fahrenheit.
3. Apply formula: Celsius = (Fahrenheit − 32) × 5/9.
4. Display the converted Celsius temperature.
5. End program.

Code:

```
import java.util.Scanner;

public class TempConversion {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter temperature in Fahrenheit: ");
        double f = sc.nextDouble();
        double c = (f - 32) * 5 / 9;
        System.out.println("Temperature in Celsius = " + c);
        System.out.println("Program by Arpit Saharawat");
    }
}
```

Output:



```
Enter temperature in Fahrenheit: 98
Temperature in Celsius = 36.666666666666664
Program by Arpit Saharawat
```

Result:

Thus, the program was successfully executed and temperature was converted from Fahrenheit to Celsius.

2. Program to implement the Sorting Operation using Control Statements.

Aim: Write a java program to accept 10 integer values from the user, store them in an array,

1. arrange the array in ascending and descending order,
2. find the Maximum, minimum and average.
3. Print only either Odd or Even

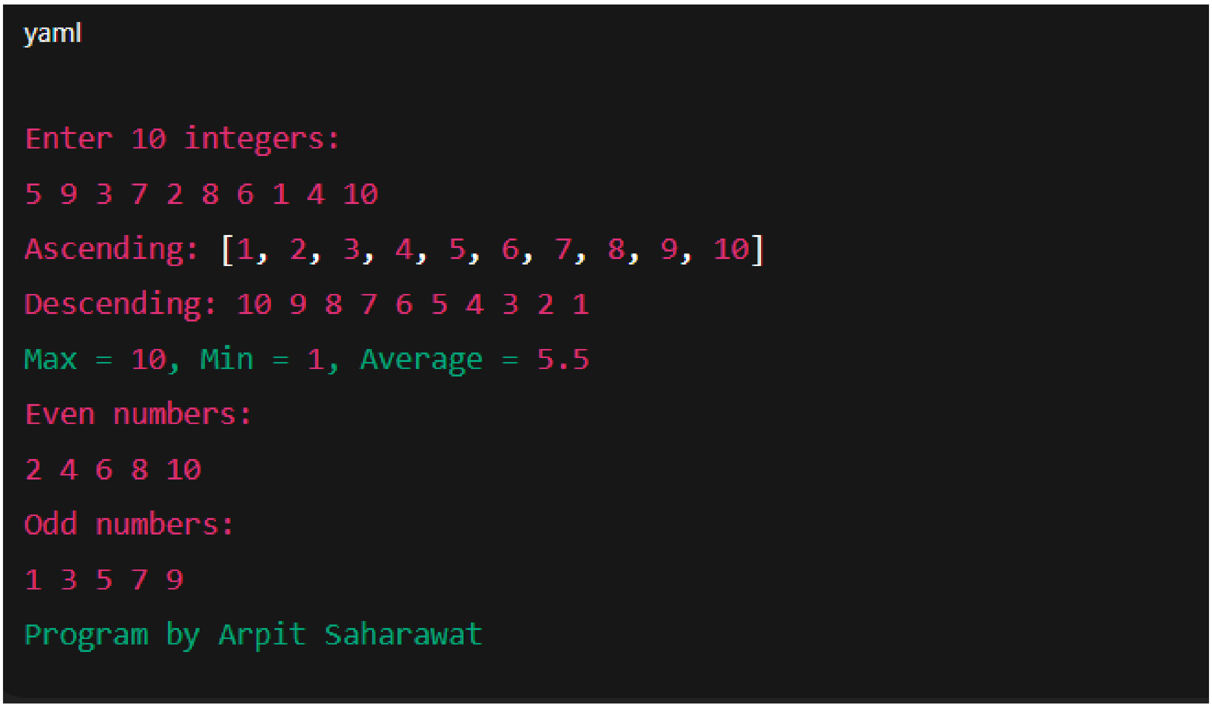
Procedure:

1. Read 10 integers into an array.
2. Sort the array.
3. Print ascending and descending order.
4. Find maximum, minimum, and average.
5. Print only odd and even numbers separately.
6. Display output.

Code:

```
import java.util.*;

public class ArrayOperations {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] arr = new int[10];

        System.out.println("Enter 10 integers:");
        for(int i=0; i<10; i++) arr[i] = sc.nextInt();

        Arrays.sort(arr);
        System.out.println("Ascending: " + Arrays.toString(arr));

        System.out.print("Descending: ");
        for(int i=arr.length-1; i>=0; i--) System.out.print(arr[i] + " ");
        System.out.println();

        int max = arr[arr.length-1], min = arr[0];
        double avg = Arrays.stream(arr).average().getAsDouble();
        System.out.println("Max = " + max + ", Min = " + min + ", Average = " + avg);

        System.out.println("Even numbers:");
        for(int x : arr) if(x % 2 == 0) System.out.print(x + " ");

        System.out.println("\nOdd numbers:");
        for(int x : arr) if(x % 2 != 0) System.out.print(x + " ");

        System.out.println("\nProgram by Arpit Saharawat");
    }
}
```

Output:



```
Enter 10 integers:
5 9 3 7 2 8 6 1 4 10
Ascending: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Descending: 10 9 8 7 6 5 4 3 2 1
Max = 10, Min = 1, Average = 5.5
Even numbers:
2 4 6 8 10
Odd numbers:
1 3 5 7 9
Program by Arpit Saharawat
```

Result:

The program successfully displayed array operations including sorting, max, min, average, and odd/even filtering.

3. Program to implement the Stack operations using Array.

Aim: To implement the fundamental stack operations (push, pop, peek, and display) using an array in Java and understand the behavior of stack data structure following Last-In-First-Out (LIFO) principle.

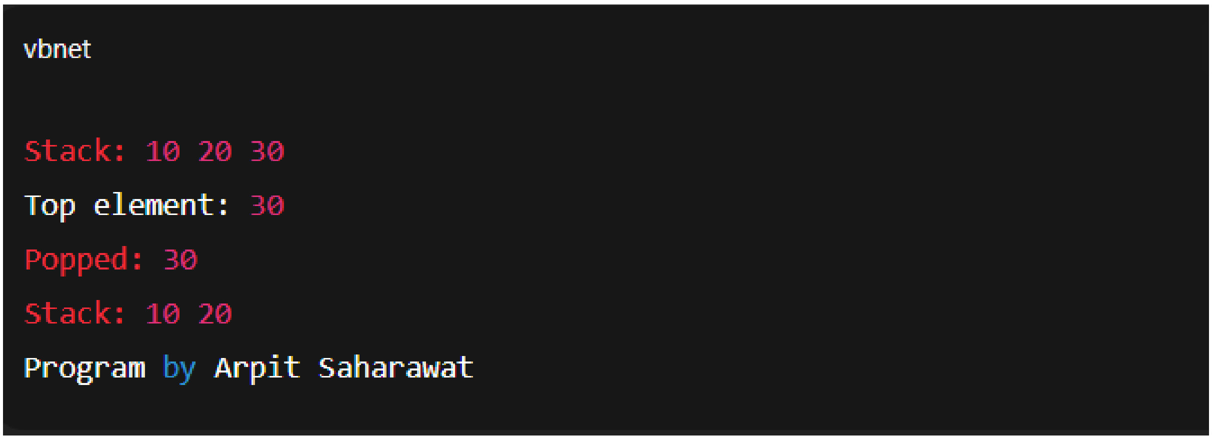
Procedure:

1. Create a stack class with an array.
2. Implement push, pop, peek, and display functions.
3. Demonstrate stack operations using LIFO principle.
4. Display results.

Code:

```
class Stack {
    int top, size;
    int[] arr;

    Stack(int size) {
        this.size = size;
        arr = new int[size];
        top = -1;
    }

    void push(int x) {
        if(top == size-1) System.out.println("Stack Overflow");
        else arr[++top] = x;
    }

    void pop() {
        if(top == -1) System.out.println("Stack Underflow");
        else System.out.println("Popped: " + arr[top--]);
    }

    void peek() {
        if(top == -1) System.out.println("Stack Empty");
        else System.out.println("Top element: " + arr[top]);
    }

    void display() {
        if(top == -1) System.out.println("Stack Empty");
        else {
            System.out.print("Stack: ");
            for(int i=0; i<=top; i++) System.out.print(arr[i] + " ");
            System.out.println();
        }
    }
}

public class StackDemo {
    public static void main(String[] args) {
        Stack st = new Stack(5);
        st.push(10); st.push(20); st.push(30);
        st.display();
        st.peek();
        st.pop();
        st.display();
        System.out.println("Program by Arpit Saharawat");
    }
}
```

Output:



```
vbnet

Stack: 10 20 30
Top element: 30
Popped: 30
Stack: 10 20
Program by Arpit Saharawat
```

Result:

The stack was successfully implemented using array with push, pop, peek, and display operations.

4. Program to implement the Queue operations using Classes and Objects.

Aim: To implement the basic Queue operations (enqueue, dequeue, peek, and display) using classes and objects in Java, and understand the First-In-First-Out (FIFO) behavior of the queue data structure.

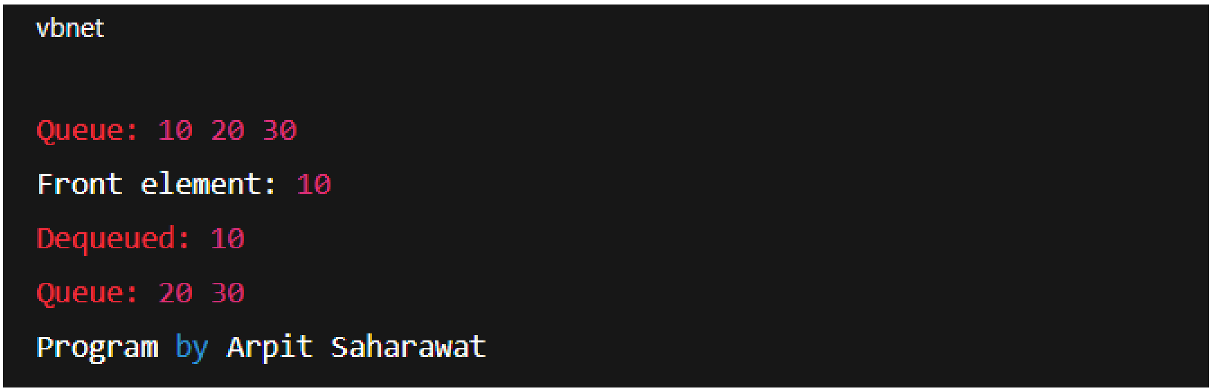
Procedure:

1. Create a Queue class with front, rear, and array.
2. Implement enqueue, dequeue, peek, and display methods.
3. Demonstrate queue operations following FIFO principle.
4. Display results.

Code:

```
class Queue {
   int front, rear, size;
   int[] arr;

   Queue(int size) {
      this.size = size;
      arr = new int[size];
      front = rear = -1;
   }

   void enqueue(int x) {
      if(rear == size-1) System.out.println("Queue Overflow");
      else {
         if(front == -1) front = 0;
         arr[++rear] = x;
      }
   }

   void dequeue() {
      if(front == -1 || front > rear) System.out.println("Queue Underflow");
      else System.out.println("Dequeued: " + arr[front++]);
   }

   void peek() {
      if(front == -1 || front > rear) System.out.println("Queue Empty");
      else System.out.println("Front element: " + arr[front]);
   }

   void display() {
      if(front == -1 || front > rear) System.out.println("Queue Empty");
      else {
         System.out.print("Queue: ");
         for(int i=front; i<=rear; i++) System.out.print(arr[i] + " ");
         System.out.println();
      }
   }
}

public class QueueDemo {
   public static void main(String[] args) {
      Queue q = new Queue(5);
      q.enqueue(10); q.enqueue(20); q.enqueue(30);
      q.display();
      q.peek();
      q.dequeue();
      q.display();
      System.out.println("Program by Arpit Saharawat");
   }
}
```

Output:



```
vbnet

Queue: 10 20 30
Front element: 10
Dequeued: 10
Queue: 20 30
Program by Arpit Saharawat
```

Result:
The queue was successfully implemented with enqueue, dequeue, peek, and display operations.

5. Implement Overloading Methods, Constructors program.

Aim: Write a java program to create a calculator. Use classes and methods to perform +,-,*,/,%.

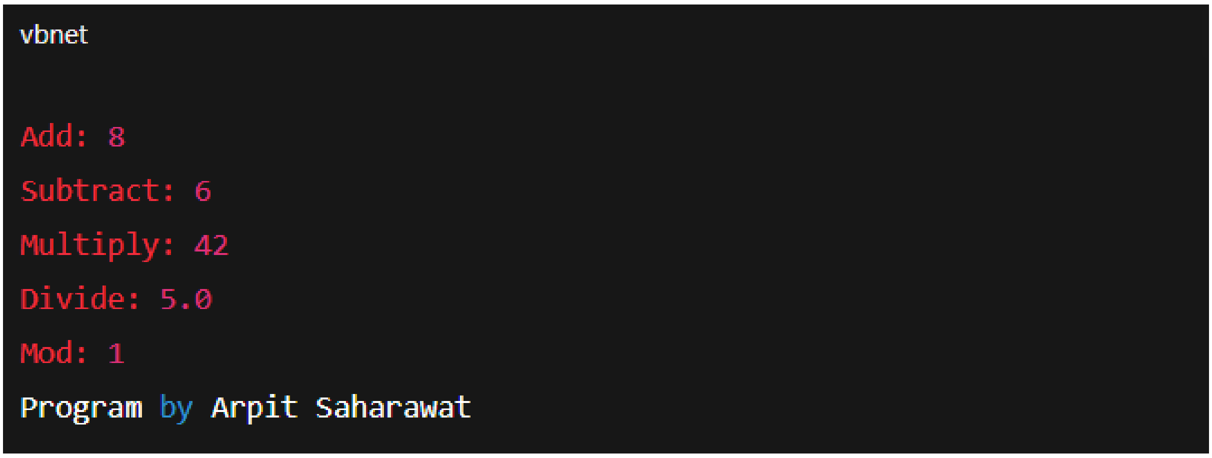
Procedure:

1. Create Calculator class with overloaded methods for +, -, *, /, %.
2. Demonstrate operations using objects.
3. Print results.

Code:

```
class Calculator {
    int add(int a, int b) { return a+b; }
    double add(double a, double b) { return a+b; }

    int subtract(int a, int b) { return a-b; }
    int multiply(int a, int b) { return a*b; }
    double divide(double a, double b) { return a/b; }
    int mod(int a, int b) { return a%b; }
}

public class CalculatorDemo {
    public static void main(String[] args) {
        Calculator c = new Calculator();
        System.out.println("Add: " + c.add(5,3));
        System.out.println("Subtract: " + c.subtract(10,4));
        System.out.println("Multiply: " + c.multiply(6,7));
        System.out.println("Divide: " + c.divide(20,4));
        System.out.println("Mod: " + c.mod(10,3));
        System.out.println("Program by Arpit Saharawat");
    }
}
```

Output:



```
vbnet

Add: 8
Subtract: 6
Multiply: 42
Divide: 5.0
Mod: 1
Program by Arpit Saharawat
```

Result:

Calculator operations using method overloading and constructors were successfully implemented.

6. Implement Tower of Hanoi program using Recursion.

Aim:

Write a Java program to check if input number is part of Fibonacci series or not. Print Fibonacci series till that number.

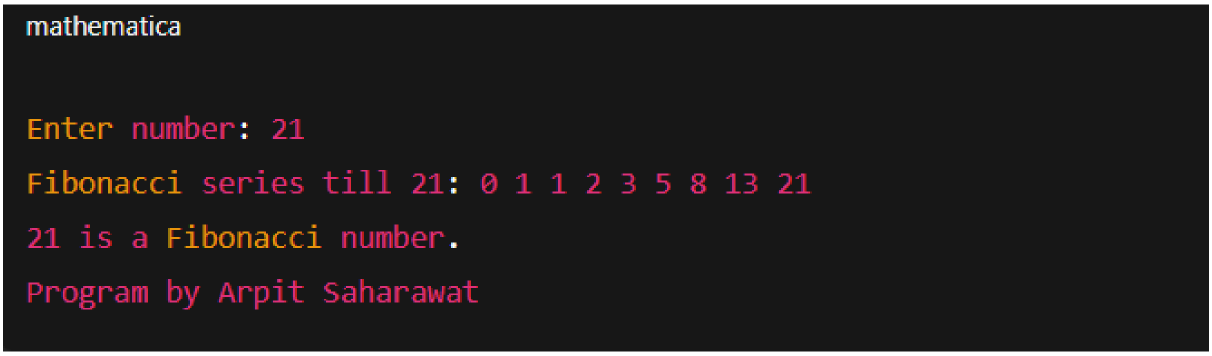
Procedure:

1. Input a number.
2. Generate Fibonacci series till the number.
3. Check if the number exists in Fibonacci sequence.
4. Print result.

Code:

```
import java.util.Scanner;

public class FibonacciCheck {
    static void printFibo(int n) {
        int a=0, b=1;
        System.out.print(a + " " + b);
        while(b < n) {
            int c = a+b;
            System.out.print(" " + c);
            a = b;
            b = c;
        }
        System.out.println();
    }

    static boolean isFibo(int n) {
        int a=0, b=1;
        while(b < n) {
            int c = a+b;
            a = b;
            b = c;
        }
        return (n==0 || n==1 || b==n);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number: ");
        int num = sc.nextInt();

        System.out.print("Fibonacci series till " + num + ": ");
        printFibo(num);

        if(isFibo(num)) System.out.println(num + " is a Fibonacci number.");
        else System.out.println(num + " is NOT a Fibonacci number.");
        System.out.println("Program by Arpit Saharawat");
    }
}
```

Output:



```
Enter number: 21
Fibonacci series till 21: 0 1 1 2 3 5 8 13 21
21 is a Fibonacci number.
Program by Arpit Saharawat
```

Result:

The program successfully generated Fibonacci series and checked whether the given number is part of the series.