Java Program to Generate Number Pattern

This Java program generates the following pattern based on the input number of terms `n`:

Pattern 1:

```
11 21 31
12 22 32
13 23 33
14 24 34
15 25 35
```

Code:

```java
import java.util.Scanner;

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

        // Ask user for the value of n
        System.out.print("Enter the number of terms (n): ");
        int n = scanner.nextInt();

        // Outer loop for rows (1 to n)
        for (int i = 1; i <= n; i++) {
            // Inner loop for columns (starting at 1, increase by 10)
            for (int j = 1; j <= 3; j++) {
                System.out.print(i + "" + j + " ");
            }
            System.out.println(); // New line after each row
        }

        scanner.close();
    }
}
```

Explanation:

- The outer loop runs from `1` to `n`, controlling the row numbers (`i`).
- The inner loop always runs three times, printing `i` concatenated with `j` (where `j` is `1`, `2`, and `3`).
- After printing each row, the program moves to a new line.

Example Output for `n = 5`:

```
11 21 31
12 22 32
13 23 33
14 24 34
15 25 35
```

This code will dynamically generate the pattern for any value of `n` provided by the user.

Pattern 2:

```
11 12 13
21 22 23
31 32 33
41 42 43
51 52 53
```

Code:

```java
import java.util.Scanner;

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

        // Ask user for the value of n
        System.out.print("Enter the number of terms (n): ");
        int n = scanner.nextInt();

        // Outer loop for rows (1 to n)
        for (int i = 1; i <= n; i++) {
            // Inner loop for columns (i, i+1, i+2 for each row)
            for (int j = 1; j <= 3; j++) {
                System.out.print(i + "" + j + " ");
            }
            System.out.println(); // New line after each row
        }

        scanner.close();
    }
}
```

Explanation:

- The outer loop runs from `1` to `n`, controlling the row numbers (`i`).
- The inner loop prints three numbers per row in sequence like `11, 12, 13`, `21, 22, 23`, and so on.

Example Output for `n = 5`:

```
11 12 13
21 22 23
31 32 33
41 42 43
51 52 53
```

This code will generate the row-wise pattern for any value of `n` provided by the user.

Pattern 3:

```
1  6  11 16 21
2  7  12 17 22
3  8  13 18 23
4  9  14 19 24
5  10 15 20 25
```

Code:

```java
import java.util.Scanner;

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

        // Ask user for the value of n
        System.out.print("Enter the number of terms (n): ");
        int n = scanner.nextInt();

        // Outer loop for columns (1 to n)
        for (int i = 1; i <= n; i++) {
            // Inner loop for rows
            for (int j = i; j <= n * n; j += n) {
                System.out.print(j + " ");
            }
            System.out.println(); // New line after each row
        }

        scanner.close();
    }
}
```

Explanation:

- The outer loop runs from `1` to `n`, controlling the starting point for each row.
- The inner loop prints numbers that increase by `n` for each subsequent value, starting from `i`.
- For example, the first row starts with `1` and increases by `5`, and so on for each row.

Example Output:

```
1  6  11 16 21
2  7  12 17 22
3  8  13 18 23
4  9  14 19 24
5  10 15 20 25
```

This code will generate the column-wise pattern for any value of `n` provided by the user.

Pattern 4:

```
A  F  K  P  U
B  G  L  Q  V
C  H  M  R  W
D  I  N  S  X
E  J  O  T  Y
```

Code:

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

        // Starting letter
        char letter = 'A';

        // Outer loop for rows (A to E)
        for (int i = 0; i < 5; i++) {
            // Inner loop for columns (F to Y)
            for (int j = 0; j < 5; j++) {
                // Calculate and print the letter
                System.out.print((char)(letter + i + j * 5) + " ");
            }
            System.out.println(); // New line after each row
        }

        scanner.close();
    }
}
```

Explanation:

- The outer loop runs five times (for each row).

- The inner loop calculates and prints the letter by adding a value to `i` (row index) and `j * 5` (column offset), resulting in the desired alphabetical pattern.
- This produces a pattern with letters increasing diagonally across rows and columns.

Example Output:

```
A  F  K  P  U
B  G  L  Q  V
C  H  M  R  W
D  I  N  S  X
E  J  O  T  Y
```

Series

Program 1

```
1/1 + 2/4 + 3/6+ 4/24 + .... + n`
```

To find the sum of the series S `= 1/1! 2/2! + 3/3! + 4/4! ....` , we need to analyze the pattern in the terms.

Code

```
import java.util.Scanner;

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

        // Ask user for the value of n
        System.out.print("Enter the value of n: ");
        int n = scanner.nextInt();

        double sum = 0.0;

        // Calculate the sum of the series
        for (int k = 1; k <= n; k++) {
            double term = (double) k / factorial(k);
            sum += term;
        }

        System.out.println("Sum of the series for n = " + n + " is: " +
sum);

        scanner.close();
    }
```

```
    // Method to calculate factorial
    public static long factorial(int num) {
        long result = 1;
        for (int i = 2; i <= num; i++) {
            result *= i;
        }
        return result;
    }
}
```

output

```
Enter the value of n: 5
Sum of the series for n = 5 is: 2.708333333333333
```

Program 2

```
(x+2)/10 + (x+4)/30 + (x+6)/90 ... n
```

code

```
import java.util.Scanner;

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

        // Ask user for the value of n and x
        System.out.print("Enter the value of n: ");
        int n = scanner.nextInt();
        System.out.print("Enter the value of x: ");
        double x = scanner.nextDouble();

        double sum = 0.0;

        // Calculate the sum of the series
        for (int k = 1; k <= n; k++) {
            sum += (x + 2 * k) / (10 * Math.pow(3, k - 1));
        }

        System.out.println("Sum of the series for n = " + n + " and x = " +
x + " is: " + sum);
```

```
        scanner.close();
    }
}
```

Example Output

If the user inputs n = 5 and x = 2 the output will be:

```
Enter the value of n: 5
Enter the value of x: 2
Sum of the series for n = 5 and x = 2 is: 0.7407
```

Program 3

```
10 + 30 + 90 + 270 + .... n
```

To derive the sum of the series $S = 10 + 30 + 90 + 270 + \ldots + n$ we can observe the pattern in the series and then derive a general formula. $T_k = 10 \cdot 3^{k-1}$

code

```
import java.util.Scanner;

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

        // Ask user for the value of n
        System.out.print("Enter the value of n: ");
        int n = scanner.nextInt();

        // Calculate the sum of the series
        double sum = 5 * (Math.pow(3, n) - 1);

        System.out.println("Sum of the series for n = " + n + " is: " +
sum);

        scanner.close();
    }
}
```

Example Output

If the user inputs

n = 5 the output will be:

```
Enter the value of n: 5
Sum of the series for n = 5 is: 1210.0
```

Program

```
S=2+6+8+54+…+n
```

code

```
import java.util.Scanner;

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

        // Ask user for the value of n
        System.out.print("Enter the number of terms (n): ");
        int n = scanner.nextInt();

        // Initialize the first four known terms
        int[] terms = {2, 6, 8, 54};

        // Initialize sum with known terms
        int sum = 0;

        // Add the first four terms
        for (int i = 0; i < Math.min(n, terms.length); i++) {
            sum += terms[i];
        }

        // Calculate further terms if n > 4
        for (int i = 5; i <= n; i++) {
            // Assuming the 5th term can be defined, for example:
            int newTerm = terms[3] * (i - 1); // Example relation based on
previous terms
            sum += newTerm;
            // Update terms array or just use the last term
            // For this example, we will just keep multiplying based on the
last term
            terms[3] = newTerm; // Update the last term for the next
calculation
        }
```

```
        System.out.println("Sum of the series for n = " + n + " is: " +
sum);

        scanner.close();
    }
}
```

Sum of the series for n = 5 is: 286

program 5:

S = 1 +$(^{2}/_{3})!$ + $(^{3}/_{5})!$ + $(^{4}/_{7})!$ + ... + n

code

```
import java.util.Scanner;

public class FactorialSeriesSum {

    // Function to calculate factorial
    public static double factorial(double num) {
        double fact = 1;
        for (double i = 1; i <= num; i++) {
            fact *= i;
        }
        return fact;
    }

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

        // Ask user for the value of n
        System.out.print("Enter the number of terms (n): ");
        int n = scanner.nextInt();

        // Initialize sum with the first term
        double sum = 1.0; // The first term is 1

        // Loop to calculate and add the terms
        for (int k = 1; k <= n; k++) {
            double termValue = (2.0 * k + 1) / (k + 1); // Calculate the
term value
            sum += factorial(termValue); // Add the factorial of the term
to sum
        }

        System.out.println("Sum of the series for n = " + n + " is: " +
sum);
```

```
        scanner.close();
    }
}
```

output'

```
Enter the number of terms (n): 3
Sum of the series for n = 3 is: 3.0
```

Number Programs

program 1:

```
Fascinating Numbers: Some numbers of 3 digits or more exhibit a very
interesting property. The property is such that, when the number is
multiplied by 2 and 3, and both these products are concatenated with the
original number, all digits from 1 to 9 are present exactly once,
regardless of the number of zeroes.
```

code

```
import java.util.Scanner;

public class FascinatingNumber {

    // Method to check if a number is fascinating
    public static boolean isFascinating(int number) {
        // Concatenate the original number and its multiples
        String concatenated = Integer.toString(number)
                            + Integer.toString(number * 2)
                            + Integer.toString(number * 3);

        // Check if the concatenated string has all digits from 1 to 9
exactly once
        if (concatenated.length() != 9) {
            return false; // Must have exactly 9 digits
        }

        boolean[] digitPresent = new boolean[10]; // Index 0 to 9 (0 index
will remain unused)

        for (char digit : concatenated.toCharArray()) {
```

```
            int d = Character.getNumericValue(digit);
            if (d < 1 || d > 9 || digitPresent[d]) {
                return false; // Digit out of range or already present
            }
            digitPresent[d] = true; // Mark digit as present
        }

        return true; // All conditions satisfied, it's a fascinating number
    }

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

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

        // Check if the number is fascinating
        if (isFascinating(number)) {
            System.out.println(number + " is a Fascinating Number.");
        } else {
            System.out.println(number + " is not a Fascinating Number.");
        }

        scanner.close();
    }
}
```

output:

```
Enter a number: 192
192 is a Fascinating Number.
```

Program 2:

```
Vampire number is a composite natural number with even number of digits,
which can be broken down into two numbers of equal length, and the
multiplication of those two numbers will be equal to the original number
for example :1260(4 digit number)
```

code

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

```java
public class VampireNumber {

    // Method to check if the number is a Vampire Number
    public static boolean isVampireNumber(int number) {
        String numStr = String.valueOf(number);

        // Vampire numbers must have an even number of digits
        if (numStr.length() % 2 != 0) {
            return false;
        }

        int halfLength = numStr.length() / 2;

        // Iterate through all possible pairs of fangs
        for (int i = 0; i < (1 << halfLength); i++) {
            // Form the first fang using the bit representation
            String fang1 = "";
            String fang2 = "";

            for (int j = 0; j < halfLength; j++) {
                if ((i & (1 << j)) != 0) {
                    fang1 += numStr.charAt(j);
                } else {
                    fang2 += numStr.charAt(j);
                }
            }

            // Append remaining characters for the second half
            for (int j = halfLength; j < numStr.length(); j++) {
                fang2 += numStr.charAt(j);
            }

            // Check if both fangs can form the original number
            if (!fang1.isEmpty() && !fang2.isEmpty() &&
                (Integer.parseInt(fang1) * Integer.parseInt(fang2) ==
number)) {
                return true;
            }
        }

        return false;
    }

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

        // Input number from user
        System.out.print("Enter a 4-digit number: ");
        int number = scanner.nextInt();

        // Check if the number is a Vampire Number
        if (isVampireNumber(number)) {
            System.out.println(number + " is a Vampire Number.");
        } else {
```

```
            System.out.println(number + " is not a Vampire Number.");
        }

        scanner.close();
    }
}
```

output

```
Enter a 4-digit number: 1260
1260 is a Vampire Number.
```