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.
```

Java Program to Check for Keith Number

A Keith Number is an integer N with 'd' digits that appears in a specific sequence. This sequence starts with the digits of N and each subsequent term is the sum of the previous 'd' terms. If N appears in this sequence, it's a Keith Number.

Code:

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

public class KeithNumber {
    public static boolean isKeithNumber(int n) {
        ArrayList<Integer> sequence = new ArrayList<>();
        int temp = n, digits = 0;

        // Count digits and add them to the sequence
        while (temp > 0) {
            sequence.add(0, temp % 10);
            temp /= 10;
            digits++;
        }

        // Generate sequence until it reaches or exceeds n
        while (sequence.get(sequence.size() - 1) < n) {
            int sum = 0;
            for (int i = sequence.size() - digits; i < sequence.size();
i++) {
                sum += sequence.get(i);
            }
            sequence.add(sum);
        }
```

```
        // Check if n is in the sequence
        return sequence.get(sequence.size() - 1) == n;
    }

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

        System.out.print("Enter a number to check if it's a Keith Number:
");
        int number = scanner.nextInt();

        if (isKeithNumber(number)) {
            System.out.println(number + " is a Keith Number.");
        } else {
            System.out.println(number + " is not a Keith Number.");
        }

        scanner.close();
    }
}
```

Output

```
Enter a number to check if it's a Keith Number: 197
197 is a Keith Number.
```

Stormer Numbers

A Stormer Number is a positive integer 'i' such that the greatest prime factor of (i*i + 1) is greater than or equal to 2*i. Here's a Java program to check if a given number is a Stormer number:

```
import java.util.Scanner;

public class StormerNumber {
    public static boolean isStormerNumber(int n) {
        long square = (long) n * n + 1;
        int largestPrimeFactor = findLargestPrimeFactor(square);
        return largestPrimeFactor >= 2 * n;
    }

    private static int findLargestPrimeFactor(long number) {
        int largestFactor = 1;
        for (int i = 2; i <= Math.sqrt(number); i++) {
            if (number % i == 0) {
                largestFactor = i;
                while (number % i == 0) {
                    number /= i;
                }
```

```
            }
        }
        if (number > 1) {
            largestFactor = (int) number;
        }
        return largestFactor;
    }

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

        System.out.print("Enter a number to check if it's a Stormer Number:
");
        int number = scanner.nextInt();

        if (isStormerNumber(number)) {
            System.out.println(number + " is a Stormer Number.");
        } else {
            System.out.println(number + " is not a Stormer Number.");
        }

        scanner.close();
    }
}
```

Output

```
Enter a number to check if it's a Stormer Number: 10
10 is a Stormer Number.
```

Abundant Number

An Abundant Number is a positive integer for which the sum of its proper divisors (excluding the number itself) is greater than the number. The difference between this sum and the number is called the abundance.

Mathematical Definition

For a number n to be an Abundant Number: sum of proper divisors of n > n

Abundance = (sum of proper divisors of n) - n

Code

```
import java.util.Scanner;

public class AbundantNumber {
    public static boolean isAbundantNumber(int number) {
        int sum = 1; // Start with 1 as it's always a proper divisor
```

```
        for (int i = 2; i <= Math.sqrt(number); i++) {
            if (number % i == 0) {
                sum += i;
                if (i != number / i) {
                    sum += number / i;
                }
            }
        }
        return sum > number;
    }

    public static int calculateAbundance(int number) {
        int sum = 1;
        for (int i = 2; i <= Math.sqrt(number); i++) {
            if (number % i == 0) {
                sum += i;
                if (i != number / i) {
                    sum += number / i;
                }
            }
        }
        return sum - number;
    }

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

        System.out.print("Enter a number to check if it's an Abundant
Number: ");
        int number = scanner.nextInt();

        if (isAbundantNumber(number)) {
            int abundance = calculateAbundance(number);
            System.out.println(number + " is an Abundant Number.");
            System.out.println("Its abundance is: " + abundance);
        } else {
            System.out.println(number + " is not an Abundant Number.");
        }

        scanner.close();
    }
}
```

Output

```
Enter a number to check if it's an Abundant Number: 12
12 is an Abundant Number.
Its abundance is: 4
```

Smith Numbers

A Smith number is a composite number for which the sum of its digits equals the sum of the digits of its prime factors (excluding 1). Here's a Java program to check if a given number is a Smith number:

Code

```
import java.util.Scanner;

public class SmithNumber {
    public static boolean isSmithNumber(int n) {
        if (isPrime(n)) return false;

        return sumOfDigits(n) == sumOfPrimeFactorDigits(n);
    }

    private static boolean isPrime(int n) {
        if (n <= 1) return false;
        for (int i = 2; i <= Math.sqrt(n); i++) {
            if (n % i == 0) return false;
        }
        return true;
    }

    private static int sumOfDigits(int n) {
        int sum = 0;
        while (n > 0) {
            sum += n % 10;
            n /= 10;
        }
        return sum;
    }

    private static int sumOfPrimeFactorDigits(int n) {
        int sum = 0;
        for (int i = 2; i <= n; i++) {
            while (n % i == 0) {
                sum += sumOfDigits(i);
                n /= i;
            }
        }
        return sum;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number to check if it's a Smith Number:
");
        int number = scanner.nextInt();

        if (isSmithNumber(number)) {
            System.out.println(number + " is a Smith Number.");
        } else {
            System.out.println(number + " is not a Smith Number.");
```

```
        }

        scanner.close();
    }
}
```

Output

```
Enter a number to check if it's a Smith Number: 666
666 is a Smith Number.
```

Java Program to Check for Kaprekar Number

This program checks whether a given number is a Kaprekar number or not. A Kaprekar number is a positive integer with an interesting property: when squared and split into two parts, the sum of these parts equals the original number.

Definition of a Kaprekar Number:

- Take a positive integer n with d digits.
- Square the number (n^2).
- Split the result into two parts:
  - A right-hand part with d digits.
  - A left-hand part with the remaining d or d-1 digits.
- If the sum of these two parts equals the original number, it's a Kaprekar number.

Examples of Kaprekar Numbers:

1. 9: 9^2 = 81, 8 + 1 = 9
2. 45: 45^2 = 2025, 20 + 25 = 45
3. 297: 297^2 = 88209, 88 + 209 = 297

Code:

```
import java.util.Scanner;

public class KaprekarNumber {
    public static boolean isKaprekar(int num) {
        long square = (long) num * num;
        String squareStr = String.valueOf(square);
        int length = squareStr.length();

        for (int i = 1; i < length; i++) {
            String left = squareStr.substring(0, i);
            String right = squareStr.substring(i);
```

```
            if (right.equals("0")) continue;

            int leftNum = Integer.parseInt(left);
            int rightNum = Integer.parseInt(right);

            if (leftNum + rightNum == num) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number to check if it's a Kaprekar
Number: ");
        int number = scanner.nextInt();

        if (isKaprekar(number)) {
            System.out.println(number + " is a Kaprekar Number.");
        } else {
            System.out.println(number + " is not a Kaprekar Number.");
        }

        scanner.close();
    }
}
```

Output

```
Enter a number to check if it's a Kaprekar Number: 297
297 is a Kaprekar Number.
```

Circular Prime

A Circular Prime is a prime number that remains prime under cyclic shifts of its digits. When the leftmost digit is removed and replaced at the end of the remaining string of digits, the generated number is still prime. This process is repeated until the original number is reached again.

A number is considered prime if it has only two factors: 1 and itself.

Example:

For n = 131:

- 131 is prime
- 311 is prime
- 113 is prime

Hence, 131 is a circular prime.

Code

```
import java.util.Scanner;

public class CircularPrime {
    public static boolean isCircularPrime(int n) {
        if (!isPrime(n)) return false;

        int digits = String.valueOf(n).length();
        int num = n;
        for (int i = 0; i < digits - 1; i++) {
            num = rotateNumber(num);
            if (!isPrime(num)) return false;
        }
        return true;
    }

    private static boolean isPrime(int n) {
        if (n <= 1) return false;
        for (int i = 2; i <= Math.sqrt(n); i++) {
            if (n % i == 0) return false;
        }
        return true;
    }

    private static int rotateNumber(int n) {
        String numStr = String.valueOf(n);
        return Integer.parseInt(numStr.substring(1) + numStr.charAt(0));
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number to check if it's a Circular Prime:
");
        int number = scanner.nextInt();

        if (isCircularPrime(number)) {
            System.out.println(number + " is a Circular Prime.");
        } else {
            System.out.println(number + " is not a Circular Prime.");
        }

        scanner.close();
    }
}
```

Output

```
Enter a number to check if it's a Circular Prime: 197
197 is a Circular Prime.
```

Bouncy Number

A Bouncy Number is a positive integer that is neither an increasing number nor a decreasing number. Here's a Java program to check if a given number is a Bouncy Number:

Code

```
import java.util.Scanner;

public class BouncyNumber {
    public static boolean isBouncyNumber(int number) {
        String numStr = String.valueOf(number);
        boolean increasing = true;
        boolean decreasing = true;

        for (int i = 1; i < numStr.length(); i++) {
            if (numStr.charAt(i) > numStr.charAt(i - 1)) {
                decreasing = false;
            } else if (numStr.charAt(i) < numStr.charAt(i - 1)) {
                increasing = false;
            }

            if (!increasing && !decreasing) {
                return true;
            }
        }

        return false;
    }

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

        System.out.print("Enter a number to check if it's a Bouncy Number:
");
        int number = scanner.nextInt();

        if (isBouncyNumber(number)) {
            System.out.println(number + " is a Bouncy Number.");
        } else {
            System.out.println(number + " is not a Bouncy Number.");
        }

        scanner.close();
    }
}
```

```
### Output
```

Enter a number to check if it's a Bouncy Number: 123 123 is not a Bouncy Number.

```
Enter a number to check if it's a Bouncy Number: 3012
3012 is a Bouncy Number.
```

ISBN Validator

This program validates a 10-digit ISBN (International Standard Book Number) code.

Code

```java
import java.util.Scanner;

public class ISBNValidator {
    public static boolean isValidISBN(String isbn) {
        // Remove any hyphens or spaces
        isbn = isbn.replaceAll("[\\-\\s]", "");

        // Check if the ISBN is 10 digits long
        if (isbn.length() != 10) {
            return false;
        }

        int sum = 0;
        for (int i = 0; i < 9; i++) {
            char digit = isbn.charAt(i);
            if (!Character.isDigit(digit)) {
                return false;
            }
            sum += (digit - '0') * (10 - i);
        }

        // Check the last character (can be 'X' or a digit)
        char lastChar = isbn.charAt(9);
        if (lastChar == 'X') {
            sum += 10;
        } else if (Character.isDigit(lastChar)) {
            sum += (lastChar - '0');
        } else {
            return false;
        }

        // The ISBN is valid if the sum is divisible by 11
        return sum % 11 == 0;
    }
```

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

        System.out.print("Enter an ISBN to validate: ");
        String isbn = scanner.nextLine();

        if (isValidISBN(isbn)) {
            System.out.println(isbn + " is a valid ISBN.");
        } else {
            System.out.println(isbn + " is not a valid ISBN.");
        }

        scanner.close();
    }
}
```

Output

```
Enter an ISBN to validate: 01293899120
01293899120 is not a valid ISBN.
```

Function Overloading

1. Function Overloading in Java

Function overloading is a feature in Java that allows a class to have multiple methods with the same name but different parameters. This program demonstrates function overloading with three different `display` methods:

1. `display()`: Prints a pattern of '&' and '@' characters.
2. `display(int n)`: Counts and displays the frequency of digits in a given number.
3. `display(String st)`: Arranges a given string in alphabetical order.

Code:

```
import java.util.Arrays;

public class DisplayOverload {
    // Method to print the pattern
    public void display() {
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= i; j++) {
                if (j % 2 == 0) {
                    System.out.print("@");
                } else {
                    System.out.print("&");
```

```
                }
            }
            System.out.println();
        }
    }

    // Method to count and print frequency of digits
    public void display(int n) {
        int[] frequency = new int[10];
        String number = String.valueOf(n);

        // Count frequency of each digit
        for (char digit : number.toCharArray()) {
            frequency[digit - '0']++;
        }

        System.out.println("Digit\tFrequency");
        for (int i = 0; i < 10; i++) {
            if (frequency[i] > 0) {
                System.out.println(i + "\t" + frequency[i]);
            }
        }
    }

    // Method to arrange string in alphabetical order
    public void display(String st) {
        System.out.println("Original string: " + st);

        char[] chars = st.toCharArray();
        Arrays.sort(chars);
        String sortedString = new String(chars);

        System.out.println("String in alphabetical order: " +
sortedString);
    }

    public static void main(String[] args) {
        DisplayOverload obj = new DisplayOverload();

        System.out.println("Pattern:");
        obj.display();

        System.out.println("\nDigit Frequency:");
        obj.display(44514621);

        System.out.println("\nString Sorting:");
        obj.display("hello");
    }
}
```

Output

```
```

```
Pattern:
@
&@
@&@
&@&@
@&@&@

Digit Frequency:
Digit   Frequency
1       2
2       1
4       3
5       1
6       1

String Sorting:
Original string: hello
String in alphabetical order: ehllo
```

2. Method Overloading

```
class MethodOverload {
    // Method to print a pattern
    void display() {
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print(combination(i - 1, j - 1) + " ");
            }
            System.out.println();
        }
    }

    // Helper method to calculate combinations
    private int combination(int n, int r) {
        if (r == 0 || r == n) return 1;
        return combination(n - 1, r - 1) + combination(n - 1, r);
    }

    // Method to print Fibonacci or Tribonacci series
    void display(int n, char ch) {
        if (ch == 'f') {
            int a = 0, b = 1;
            System.out.print(a + " " + b + " ");
            for (int i = 2; i < n; i++) {
                int c = a + b;
                System.out.print(c + " ");
                a = b;
                b = c;
            }
        } else if (ch == 't') {
```

```
            int a = 0, b = 1, c = 1;
            System.out.print(a + " " + b + " " + c + " ");
            for (int i = 3; i < n; i++) {
                int d = a + b + c;
                System.out.print(d + " ");
                a = b;
                b = c;
                c = d;
            }
        }
        System.out.println();
    }

    // Method to check if a number is a beam number
    boolean display(int n) {
        int sum = 0;
        int temp = n;
        while (temp > 0) {
            int digit = temp % 10;
            sum += digit * digit;
            temp /= 10;
        }
        return sum > n;
    }
}

public class DisplayOverload {
    // ... (existing code)

    public static void main(String[] args) {
        DisplayOverload obj = new DisplayOverload();
        MethodOverload mo = new MethodOverload();

        System.out.println("Pattern:");
        obj.display();

        System.out.println("\nDigit Frequency:");
        obj.display(44514621);

        System.out.println("\nString Sorting:");
        obj.display("hello");

        System.out.println("\nPascal's Triangle:");
        mo.display();

        System.out.println("\nFibonacci Series (10 terms):");
        mo.display(10, 'f');

        System.out.println("\nTribonacci Series (10 terms):");
        mo.display(10, 't');

        System.out.println("\nBeam Number Check:");
        System.out.println("Is 25 a beam number? " + mo.display(25));
        System.out.println("Is 7 a beam number? " + mo.display(7));
```

```
    }
}
```

Output

```
Pattern:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1

Fibonacci Series (10 terms):
0 1 1 2 3 5 8 13 21 34

Tribonacci  Series (10 terms):
0 1 1 2 4 7 13 24 44 81

Beam Number Check:
Is 25 a beam number? true
Is 7 a beam number? false
```

3. Method Overloading

Code

```
class MethodOverload {
    // Method to print the pattern
    public void perform() {
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= 5; j++) {
                if (i == 1 || j == i || (i == 1 && j <= 5) || (j == 5 && i
<= 5)) {
                    System.out.print(j + " ");
                } else {
                    System.out.print("  ");
                }
            }
            System.out.println();
        }
    }

    // Method to display multiplication table
    public void perform(int m, char n) {
        for (int i = 1; i <= (n - '0'); i++) {
            System.out.println(m + " x " + i + " = " + (m * i));
        }
    }
```

```
    // Method to display series
    public void perform(int n) {
        int sum = 0;
        for (int i = 1; i <= n; i++) {
            sum += i;
            System.out.print(sum);
            if (i < n) {
                System.out.print(", ");
            }
        }
        System.out.println();
    }
}
public static void main(String[] args) {
    MethodOverload mo = new MethodOverload();

    System.out.println("\nPattern:");
    mo.perform();

    System.out.println("\nMultiplication Table of 5 up to 7:");
    mo.perform(5, '7');

    System.out.println("\nSeries (10 terms):");
    mo.perform(10);
}
```

Output

```
Pattern:
1 2 3 4 5
2     5
3    5
4 5
5

Multiplication Table of 5 up to 7:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35

Series (10 terms):
1, 3, 6, 10, 15, 21, 28, 36, 45, 55
```