Patterns

Introduction

Patterns in Java are a common programming exercise involving the creation of specific shapes or sequences of characters using loops. They serve as a valuable practice for understanding control structures and enhancing problem-solving skills. In this lesson, we'll explore three types of patterns: square patterns, triangle patterns, and pyramid patterns. We'll provide step-by-step explanations for each type, along with code examples using both stars and numbers.

Before printing any pattern, you must consider the following three things:

- The first step in printing any pattern is to figure out the number of rows that the pattern requires.
- Next, you should know how many columns are there in the i'th row.
- Once you have figured out the number of rows and columns, then focus on the pattern to print.

1. Square Patterns

- Printing a square pattern of stars.

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

Algorithm to follow:

Refer to the code block below simultaneously for easy understanding.

- Input the size of the square (e.g., size = 5).
- Use a nested loop structure with an outer loop controlling the rows and an inner loop controlling the columns.
- For each row, print asterisks (*) equal to the size of the square.
- Move to the next row by printing a newline character (System.out.println()).

```
public class SquarePatternStars {
    public static void main(String[] args) {
        int size = 5; // Size of the square
        for (int i = 1; i <= size; i++) {
            for (int j = 1; j <= size; j++) {
                System.out.print("*"+" ");
            }
            System.out.println();
        }
    }
}
```

- Printing a square pattern using numbers.
  Expected output:

```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

Algorithm to follow:

Refer to the code block below simultaneously for easy understanding.

- Input the size of the square (e.g., size = 5).
- Use a nested loop structure with an outer loop controlling the rows and an inner loop controlling the columns.
- For each row, print the value of i, where i is the row number, in each column.
- Move to the next row by printing a newline character (System.out.println()).

```
public class SquarePatternNumbers {
    public static void main(String[] args) {
        int size = 5; // Size of the square
        for (int i = 1; i <= size; i++) {
            for (int j = 1; j <= size; j++) {
                System.out.print(i + " ");
            }
            System.out.println();
        }
    }
}
```

2. Triangular Patterns

- To print a triangle pattern with a given number of rows, using asterisks.

Expected output:

```
*
* *
* * *
* * * *
* * * * *
```

Algorithm to follow:

Refer to the code block below simultaneously for easy understanding.

- Input the number of rows for the triangle (e.g., size = 5).
- Use a nested loop structure with an outer loop controlling the rows and an inner loop controlling the columns.

- In each row, print asterisks (*) equal to the row number. For the first row, print one asterisk, for the second row, print two asterisks, and so on.
- Move to the next row by printing a newline character (System.out.println()).

```
public class TrianglePatternStars {
    public static void main(String[] args) {
        int size = 5; // Number of rows
        for (int i = 1; i <= size; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print("* ");
            }
            System.out.println();
        }
    }
}
```

- Similarly for printing numbers with the pattern shown below:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Algorithm to follow:

(Refer to the code block below simultaneously for easy understanding).

- Input the number of rows for the triangle (e.g., size = 5).
- Use a nested loop structure with an outer loop controlling the rows and an inner loop controlling the columns.
- In each row, print numbers from 1 to i, where i is the row number. For the first row, print 1, for the second row, print 1 2, and so on.
- Move to the next row by printing a newline character (System.out.println()).

```
public class TrianglePatternNumbers {
    public static void main(String[] args) {
        int size = 5; // Number of rows
        for (int i = 1; i <= size; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print(j + " ");
            }
            System.out.println();
        }
    }
}
```

- Same can be done for numbers with decreasing order:

```
5
5 4
5 4 3
5 4 3 2
5 4 3 2 1
```

Solution:

```
public class TrianglePatternNumbersDescending {
    public static void main(String[] args) {
        int size = 5; // Number of rows
        for (int i = 1; i <= size; i++) {
            for (int j = size; j >= i; j--) {
                System.out.print(j + " ");
            }
            System.out.println();
        }
    }
}
```

3. Pyramid Pattern

- Pyramid pattern with a given number of rows using asterisks (*).
  Expected output:

```
      *
     * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

Algorithm to follow:

(Refer to the code block below simultaneously for easy understanding.)

- Input the number of rows for the pyramid (e.g., size = 5).
- Use a nested loop structure with an outer loop controlling the rows and an inner loop controlling the columns.
- In each row, start by printing spaces for indentation. The number of spaces is equal to size - row number.
- After printing the spaces, print asterisks (*). The number of asterisks in each row is equal to 2 * row number - 1.
- Move to the next row by printing a newline character (System.out.println()).

```
public class PyramidPatternStars {
    public static void main(String[] args) {
        int size = 5; // Number of rows
        for (int i = 1; i <= size; i++) {
            // Print spaces for indentation
            for (int j = 1; j <= size - i; j++) {
                System.out.print("  ");
            }
            // Print asterisks in an ascending order
            for (int k = 1; k <= 2 * i - 1; k++) {
                System.out.print("* ");
            }
            System.out.println();
        }
    }
}
```

- Pyramid pattern with a given number of rows using numbers.
  Expected Output:

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

Algorithm to follow:

(Refer to the code block below simultaneously for easy understanding.)

- Input the number of rows for the pyramid (e.g., size = 5).
- Use a nested loop structure with an outer loop controlling the rows and an inner loop controlling the columns.
- In each row, start by printing spaces for indentation. The number of spaces is equal to (size - row) number.
- After printing the spaces, print numbers in an ascending order. The numbers start from 1 and go up to 2 * row number - 1.
- Move to the next row by printing a newline character (System.out.println()).

```
public class PyramidPatternNumbers {
    public static void main(String[] args) {
        int size = 5; // Number of rows
        for (int i = 1; i <= size; i++) {
            // Print spaces for indentation
            for (int j = 1; j <= size - i; j++) {
```

```
                System.out.print("  ");
            }
            // Print numbers in an ascending order
            for (int k = 1; k <= 2 * i - 1; k++) {
                System.out.print(k + " ");
            }
            System.out.println();
        }
    }
}
```

Conclusion

Patterns in Java offer an engaging and instructive programming exercise that involves using loops to create visually appealing shapes and sequences of characters. In this exploration of patterns, we've covered square patterns, triangle patterns, and pyramid patterns, each providing unique challenges and opportunities for enhancing programming skills. Here are key takeaways:

- **Fundamental Control Structures**: Creating patterns reinforces the understanding of fundamental control structures, including loops and conditional statements, which are essential in Java programming.
- **Algorithmic Thinking**: Patterns encourage algorithmic thinking, as designers must strategize on how to print characters or numbers in specific arrangements to achieve the desired pattern.
- **Nested Loops**: Nested loops play a crucial role in creating patterns. Understanding how to control inner and outer loops is fundamental to achieving complex designs.
- **Variety of Patterns**: Java allows for an array of patterns, from simple squares and triangles to more intricate pyramids and custom designs. Patterns can be adapted to suit various requirements and aesthetics.