Loops in JAVA:

Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true. There are several types of loops in Java:

1. While Loop

The `while` loop is used to execute a block of code as long as a specified condition is true.

A while loop begins by checking a condition. If the condition is true, it executes the code inside the loop. If the condition is false, it skips the loop and moves to the next line of code. This is why it's called an entry control loop.

When the condition is true, the loop runs its statements, often updating a variable for the next iteration. The loop keeps running as long as the condition stays true.

Once the condition turns false, the loop stops, ending its cycle.

Syntax:

```
while (condition) {
    // code to be executed
}
```

Example:

```
public class WhileLoopExample {
    public static void main(String[] args) {
        int i = 0;
        while (i < 5) {
            System.out.println("Iteration: " + i);
            i++;
        }
    }
}
```

2. Do-While Loop

The `do-while` loop is similar to the `while` loop, but it guarantees that the code block is executed at least once because the condition is evaluated after the block of code is executed.

A do-while loop begins by executing its statements without checking any condition initially. After running the statements and updating the variable, it then checks the condition to see if it's true or false. If the condition is true, the loop runs again.

When the condition turns false, the loop stops, ending its cycle. It's important to note that a do-while loop always executes its statements at least once before checking the condition, which is why it's known as an exit control loop.

Syntax:

```
do {
    // code to be executed
} while (condition);
```

Example:

```
public class DoWhileLoopExample {
    public static void main(String[] args) {
        int i = 0;
        do {
            System.out.println("Iteration: " + i);
            i++;
        } while (i < 5);
    }
}
```

3. For Loop

The `for` loop is used when you know in advance how many times you want to execute a statement or a block of statements.

- **Initialization**: This is where the loop's variable gets its starting value. This can be a variable that has already been declared or a new one that's declared specifically for the loop.
- **Testing**: This checks if the loop should continue running or stop. It needs to return a true or false value and is evaluated before the loop executes its statements, making it an entry control loop.
- **Statement Execution**: If the testing condition is true, the code inside the loop runs.
- **Increment/Decrement**: This updates the loop's variable, preparing it for the next round of the loop.
- **Loop Termination**: When the testing condition is false, the loop ends, signaling the completion of its cycle.

Syntax:

```
for (initialization; condition; increment/decrement) {
    // code to be executed
}
```

Example:

```
public class ForLoopExample {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            System.out.println("Iteration: " + i);
```

```
        }
    }
}
```

4. Enhanced For Loop (for-each Loop)

The enhanced `for` loop is used to iterate over arrays or collections. It is a simplified version of the `for` loop.

It starts with the keyword for like a normal for-loop.

Instead of declaring and initializing a loop counter variable, you declare a variable that is the same type as the base type of the array, followed by a colon, which is then followed by the array name.

In the loop body, you can use the loop variable you created rather than using an indexed array element.

It's commonly used to iterate over an array or a Collections class (eg, ArrayList)

Syntax:

```
for (type variable : array) {
    // code to be executed
}
```

Example:

```
public class ForEachLoopExample {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5};
        for (int number : numbers) {
            System.out.println("Number: " + number);
        }
    }
}
```

Continue and Break Statements in Java

The `continue` and `break` statements are used to control the flow of loops (like `for`, `while`, and `do-while` loops) in Java.

Break Statement

The break statement is used to exit a loop prematurely. When a break statement is encountered inside a loop, the loop is immediately terminated, and the control is transferred to the statement following the loop.

- **Syntax**:

```
break;
```

Real-Time Example

Imagine you are processing a list of student grades, and you want to calculate the average grade. If a grade is negative (which might indicate some error in the data entry), you want to skip that grade and move on to the next one without exiting the loop.

Example

```
public class ContinueExample {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            if (i == 5) {
                continue; // Skip the rest of the code for this iteration
            }
            System.out.println(i);
        }
    }
}
```

Continue Statement

The continue statement is used to skip the current iteration of a loop and proceed to the next iteration. When a continue statement is encountered inside a loop, the remaining code inside the loop for the current iteration is skipped, and the loop proceeds with the next iteration.

- **Syntax**:

```
continue;
```

Real-Time Example

Suppose you are searching for a specific value in an array. Once you find the value, you want to stop searching further.

Example

```
public class BreakExample {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int searchValue = 7;

        for (int number : numbers) {
            if (number == searchValue) {
                System.out.println("Found the number: " + number);
                break; // Exit the loop once the value is found
            }
        }
    }
}
```