### **Break Statement:**

The `break` statement is used to prematurely exit a loop. When a `break` statement is encountered within a loop, the loop immediately terminates, and the program execution continues with the statement following the loop.

Here's an example of using the `break` statement:

```java
for (int i = 1; i <= 10; i++) {
    if (i == 5) {
        break; // Terminate the loop when i equals 5
    }
    System.out.println(i);
}
```

In this example, the loop iterates from 1 to 10, but when `i` becomes 5, the `break` statement is encountered, and the loop is terminated. So, only numbers 1 through 4 will be printed.

### Continue Statement:

The `continue` statement is used to skip the current iteration of a loop and move to the next iteration. When a `continue` statement is encountered within a loop, the remaining code within the current iteration is skipped, and the loop continues with the next iteration.

Here's an example of using the `continue` statement:

```java
for (int i = 1; i <= 10; i++) {
    if (i % 2 == 0) {
        continue; // Skip even numbers and move to the next iteration
    }
    System.out.println(i);
}
```

In this example, the loop iterates from 1 to 10, but when `i` is an even number, the `continue` statement is encountered, and the remaining code within that iteration is skipped. So, only odd numbers will be printed.

### **Using Break and Continue with Nested Loops:**

Both `break` and `continue` statements can also be used within nested loops to control the flow of execution in more complex scenarios.

```java
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 5; j++) {
        if (i == 3 && j == 3) {
            break; // Exit the inner loop when i is 3 and j is 3
        }
        System.out.println("i: " + i + ", j: " + j);
    }
}
```

In this nested loop example, when `i` is 3 and `j` is 3, the `break` statement is encountered, causing only the inner loop to terminate. The outer loop continues its iterations.

Both `break` and `continue` statements provide you with more control over loops, allowing you to skip iterations or exit loops when specific conditions are met. However, their usage should be carefully considered to maintain code readability and avoid creating overly complex logic.