**Branching Statements**

**The break Statement**
The break statement has two forms: labeled and unlabeled. You saw the unlabeled form in the previous discussion of the switch statement. You can also use an unlabeled break to terminate a for, while, or do-while loop, as shown in the following BreakDemo program:

class BreakDemo {
 public static void main(String[] args) {

 int[] arrayOfInts =
 { 32, 87, 3, 589,
 12, 1076, 2000,
 8, 622, 127 };
 int searchfor = 12;

 int i;
 boolean foundIt = false;

 for (i = 0; i < arrayOfInts.length; i++) {
 if (arrayOfInts[i] == searchfor) {
 foundIt = true;
 break;
 }
 }

 if (foundIt) {
 System.out.println("Found " + searchfor + " at index " + i);
 } else {
 System.out.println(searchfor + " not in the array");
 }
 }
}
This program searches for the number 12 in an array. The break statement, shown in boldface, terminates the for loop when that value is found. Control flow then transfers to the statement after the for loop. This program's output is:

Found 12 at index 4
An unlabeled break statement terminates the innermost switch, for, while, or do-while statement, but a labeled break terminates an outer statement. The following program, BreakWithLabelDemo, is similar to the previous program, but uses nested for loops to search for a value in a two-dimensional array. When the value is found, a labeled break terminates the outer for loop (labeled "search"):

class BreakWithLabelDemo {
 public static void main(String[] args) {

 int[][] arrayOfInts = {
 { 32, 87, 3, 589 },
 { 12, 1076, 2000, 8 },
 { 622, 127, 77, 955 }
 };
 int searchfor = 12;

 int i;
 int j = 0;
 boolean foundIt = false;

 search:
 for (i = 0; i < arrayOfInts.length; i++) {
 for (j = 0; j < arrayOfInts[i].length;
 j++) {
 if (arrayOfInts[i][j] == searchfor) {
 foundIt = true;
 break search;
 }
 }
 }

 if (foundIt) {
 System.out.println("Found " + searchfor + " at " + i + ", " + j);
 } else {
 System.out.println(searchfor + " not in the array");
 }
 }
}
This is the output of the program.

Found 12 at 1, 0
The break statement terminates the labeled statement; it does not transfer the flow of control to the label. Control flow is transferred to the statement immediately following the labeled (terminated) statement.

**The continue Statement**
The continue statement skips the current iteration of a for, while , or do-while loop. The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop. The following program, ContinueDemo , steps through a String, counting the occurences of the letter "p". If the current character is not a p, the continue statement skips the rest of the loop and proceeds to the next character. If it is a "p", the program increments the letter count.

class ContinueDemo {
 public static void main(String[] args) {

 String searchMe = "peter piper picked a " + "peck of pickled peppers";
 int max = searchMe.length();
 int numPs = 0;

 for (int i = 0; i < max; i++) {
 // interested only in p's
 if (searchMe.charAt(i) != 'p')
 continue;

 // process p's
 numPs++;
 }
 System.out.println("Found " + numPs + " p's in the string.");
 }
}
Here is the output of this program:

Found 9 p's in the string.
To see this effect more clearly, try removing the continue statement and recompiling. When you run the program again, the count will be wrong, saying that it found 35 p's instead of 9.

A labeled continue statement skips the current iteration of an outer loop marked with the given label. The following example program, ContinueWithLabelDemo, uses nested loops to search for a substring within another string. Two nested loops are required: one to iterate over the substring and one to iterate over the string being searched. The following program, ContinueWithLabelDemo, uses the labeled form of continue to skip an iteration in the outer loop.

class ContinueWithLabelDemo {
 public static void main(String[] args) {

 String searchMe = "Look for a substring in me";
 String substring = "sub";
 boolean foundIt = false;

 int max = searchMe.length() -
 substring.length();

 test:
 for (int i = 0; i <= max; i++) {
 int n = substring.length();
 int j = i;
 int k = 0;
 while (n-- != 0) {
 if (searchMe.charAt(j++) != substring.charAt(k++)) {
 continue test;
 }
 }
 foundIt = true;
 break test;
 }
 System.out.println(foundIt ? "Found it" : "Didn't find it");
 }
}
Here is the output from this program.

Found it

**The return Statement**
The last of the branching statements is the return statement. The return statement exits from the current method, and control flow returns to where the method was invoked. The return statement has two forms: one that returns a value, and one that doesn't. To return a value, simply put the value (or an expression that calculates the value) after the return keyword.

return ++count;
The data type of the returned value must match the type of the method's declared return value. When a method is declared void, use the form of return that doesn't return a value.

Return;

**Refer to:** <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html>