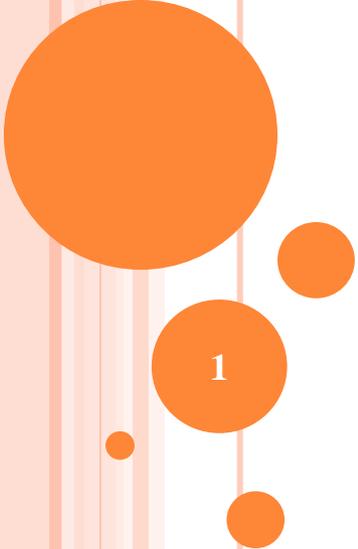


CHAPTER 4 LOOPS



1

MOTIVATIONS

Suppose that you need to print a string (e.g., "Welcome to Java!") a hundred times. It would be tedious to have to write the following statement a hundred times:

```
System.out.println("Welcome to Java!");
```

So, how do you solve this problem?

OPENING PROBLEM

Problem:

100
times

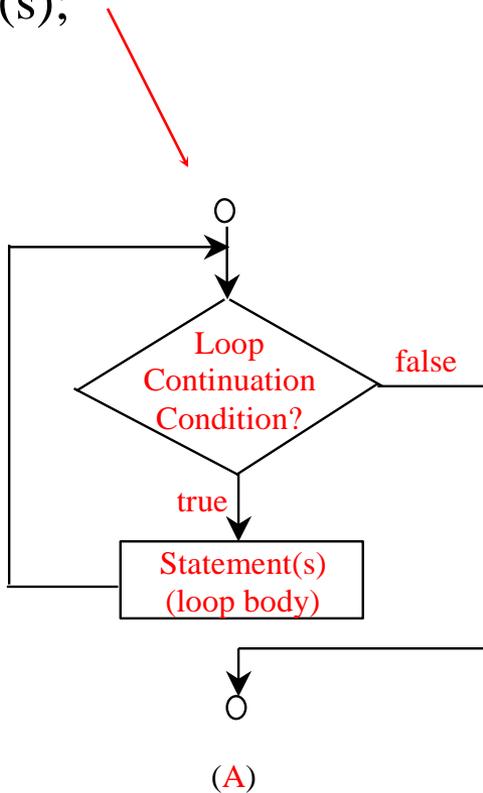
```
System.out.println("Welcome to Java!");  
...  
...  
...  
System.out.println("Welcome to Java!");  
System.out.println("Welcome to Java!");  
System.out.println("Welcome to Java!");
```

INTRODUCING WHILE LOOPS

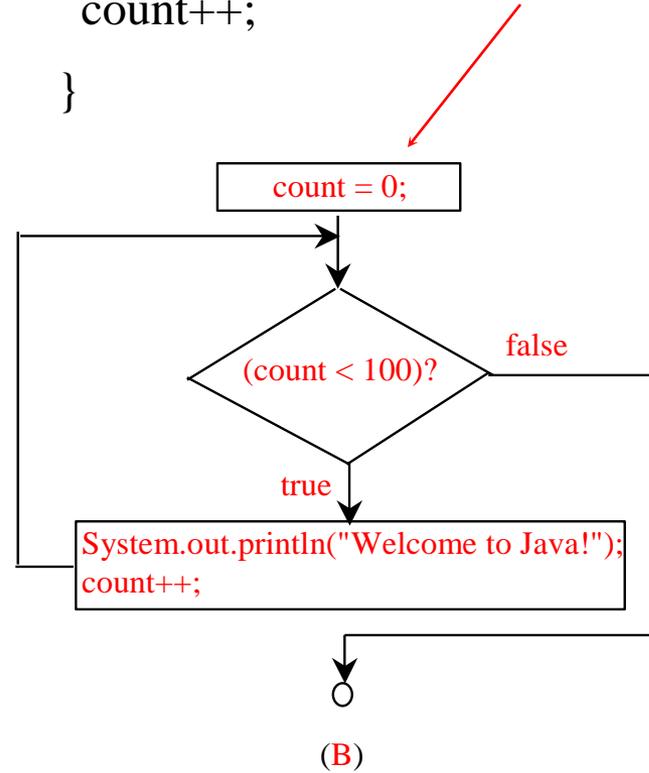
```
int count = 0;  
while (count < 100) {  
    System.out.println("Welcome to Java");  
    count++;  
}
```

WHILE LOOP FLOW CHART

```
while (loop-continuation-condition) {  
    // loop-body;  
    Statement(s);  
}
```



```
int count = 0;  
while (count < 100) {  
    System.out.println("Welcome to Java!");  
    count++;  
}
```



TRACE WHILE LOOP

Initialize count

```
int count = 0;
```

```
while (count < 2) {
```

```
    System.out.println("Welcome to Java!");
```

```
    count++;
```

```
}
```

TRACE WHILE LOOP, CONT.

```
int count = 0;
```

```
while (count < 2) {  
    System.out.println("Welcome to Java!");  
    count++;  
}
```

(count < 2) is true

TRACE WHILE LOOP, CONT.

```
int count = 0;
```

```
while (count < 2) {
```

```
    System.out.println("Welcome to Java!");
```

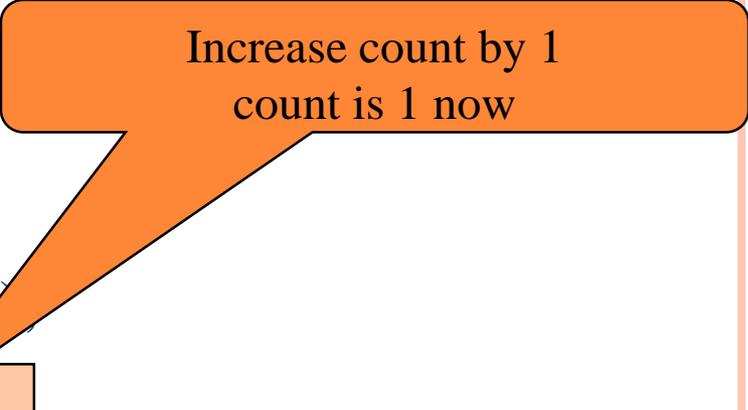
```
    count++;
```

```
}
```

Print Welcome to Java

TRACE WHILE LOOP, CONT.

```
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
```



Increase count by 1
count is 1 now

TRACE WHILE LOOP, CONT.

```
int count = 0;
```

```
while (count < 2) {
```

```
    System.out.println("Welcome to Java!");
```

```
    count++;
```

```
}
```

(count < 2) is still true since count is 1

TRACE WHILE LOOP, CONT.

```
int count = 0;
```

```
while (count < 2) {
```

```
    System.out.println("Welcome to Java!");
```

```
    count++;
```

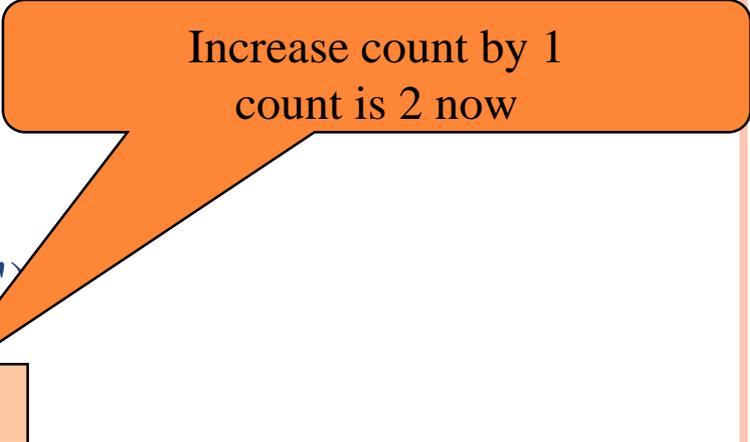
```
}
```



Print Welcome to Java

TRACE WHILE LOOP, CONT.

```
int count = 0;  
while (count < 2) {  
    System.out.println("Welcome to Java!")  
    count++;  
}
```



Increase count by 1
count is 2 now

TRACE WHILE LOOP, CONT.

```
int count = 0;
```

```
while (count < 2) {
```

```
    System.out.println("Welcome to Java!");
```

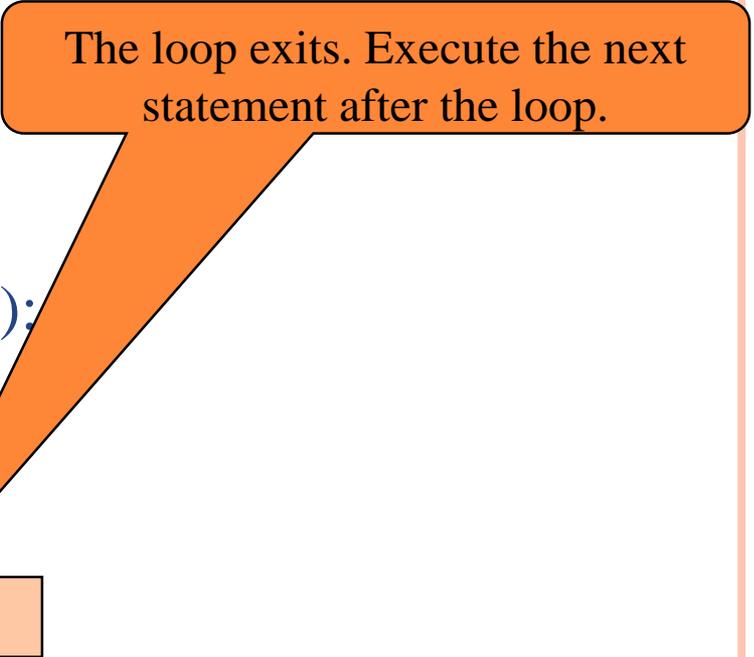
```
    count++;
```

```
}
```

(count < 2) is false since count is 2
now

TRACE WHILE LOOP

```
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
```



The loop exits. Execute the next statement after the loop.

ENDING A LOOP WITH A SENTINEL VALUE

Often the number of times a loop is executed is not predetermined. You may use an input value to signify the end of the loop. Such a value is known as a *sentinel value*.

Write a program that reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.

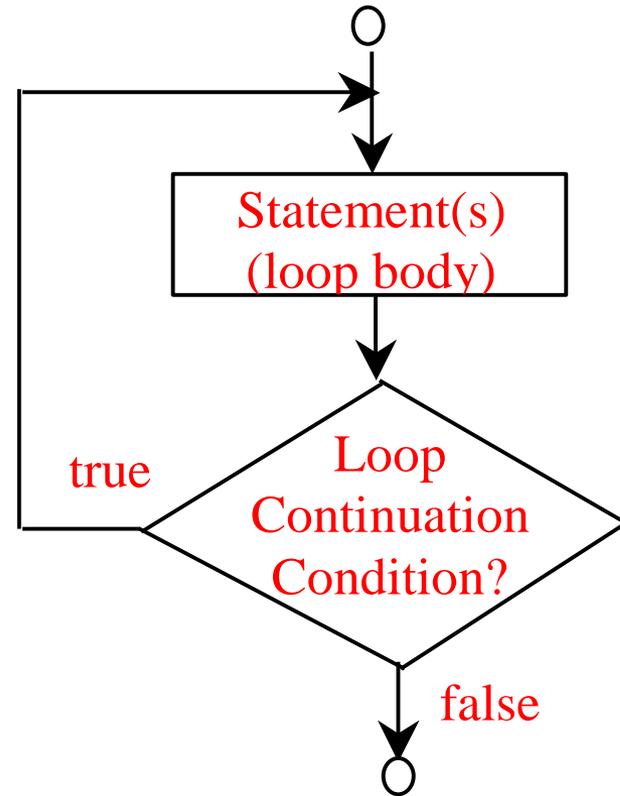
CAUTION

Don't use floating-point values for equality checking in a loop control. Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results. Consider the following code for computing $1 + 0.9 + 0.8 + \dots + 0.1$:

```
double item = 1; double sum = 0;
while (item != 0) { // No guarantee item will be 0
    sum += item;
    item -= 0.1;
}
System.out.println(sum);
```

DO-WHILE LOOP

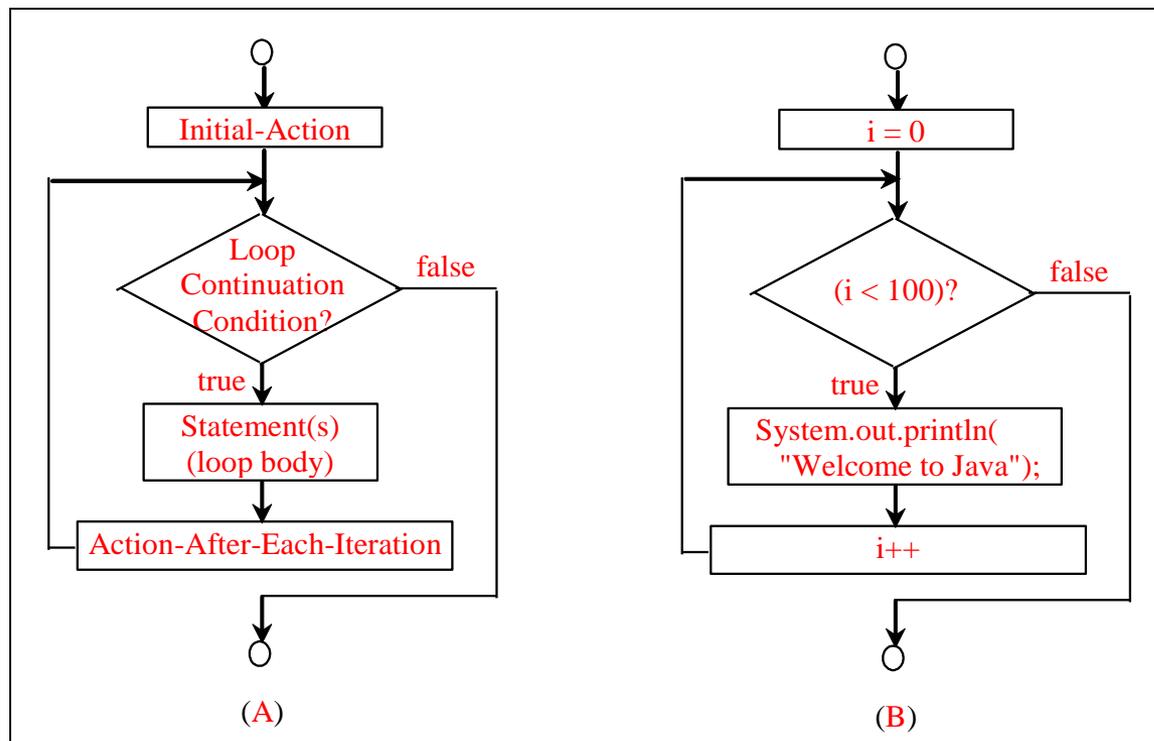
```
do {  
    // Loop body;  
    Statement(s);  
} while (loop-continuation-condition);
```



FOR LOOPS

```
for (initial-action; loop-  
continuation-condition;  
action-after-each-iteration)  
{  
// loop body;  
Statement(s);  
}
```

```
int i;  
for (i = 0; i < 100; i++) {  
    System.out.println(  
        "Welcome to Java!");  
}
```



TRACE FOR LOOP

Declare i

```
int i;
```

```
for (i = 0; i < 2; i++) {  
    System.out.println(  
        "Welcome to Java!");  
}
```

TRACE FOR LOOP, CONT.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println(  
        "Welcome to Java!");  
}
```

Execute initializer
i is now 0

TRACE FOR LOOP, CONT.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println( "Welcome to Java!");  
}
```

(i < 2) is true
since i is 0

TRACE FOR LOOP, CONT.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println("Welcome to Java!");  
}
```



Print Welcome to Java



System.out.println("Welcome to Java!");

TRACE FOR LOOP, CONT.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println("Welcome to Java!");  
}
```

Execute adjustment statement
i now is 1

TRACE FOR LOOP, CONT.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println("Welcome to Java!");  
}
```

(i < 2) is still true
since i is 1

TRACE FOR LOOP, CONT.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println("Welcome to Java!");  
}
```



Print Welcome to Java

TRACE FOR LOOP, CONT.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println("Welcome to Java!");  
}
```

Execute adjustment statement
i now is 2

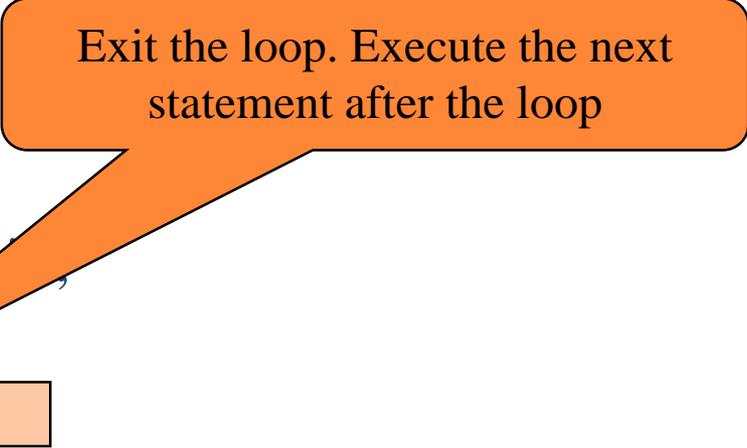
TRACE FOR LOOP, CONT.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println("Welcome to Java!");  
}
```

(i < 2) is false
since i is 2

TRACE FOR LOOP, CONT.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println("Welcome to Java");  
}
```



Exit the loop. Execute the next statement after the loop

NOTE

The initial-action in a for loop can be a list of zero or more comma-separated expressions. The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements. Therefore, the following two for loops are correct. They are rarely used in practice, however.

```
for (int i = 1; i < 100; System.out.println(i++));
```

```
for (int i = 0, j = 0; (i + j < 10); i++, j++) {
```

```
    // Do something
```

```
}
```

NOTE

If the loop-continuation-condition in a for loop is omitted, it is implicitly true. Thus the statement given below in (a), which is an infinite loop, is correct. Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:

```
for ( ; ; ) {  
    // Do something  
}
```

(a)

Equivalent

```
while (true) {  
    // Do something  
}
```

(b)

CAUTION

Adding a semicolon at the end of the for clause before the loop body is a common mistake, as shown below:

```
for (int i=0; i<10; i++);  
{  
    System.out.println("i is " + i);  
}
```

Logic Error



CAUTION, CONT.

Similarly, the following loop is also wrong:

```
int i=0;
while (i < 10); Logic Error
{
    System.out.println("i is " + i);
    i++;
}
```

In the case of the do loop, the following semicolon is needed to end the loop.

```
int i=0;
do {
    System.out.println("i is " + i);
    i++;
} while (i<10); Correct
```

WHICH LOOP TO USE?

The three forms of loop statements, while, do-while, and for, are expressively equivalent; that is, you can write a loop in any of these three forms. For example, a while loop in (a) in the following figure can always be converted into the following for loop in (b):

```
while (loop-continuation-condition) {  
    // Loop body  
}
```

(a)

Equivalent

```
for ( ; loop-continuation-condition; )  
    // Loop body  
}
```

(b)

A for loop in (a) in the following figure can generally be converted into the following while loop in (b) except in certain special cases:

```
for (initial-action;  
     loop-continuation-condition;  
     action-after-each-iteration) {  
    // Loop body;  
}
```

(a)

Equivalent

```
initial-action;  
while (loop-continuation-condition) {  
    // Loop body;  
    action-after-each-iteration;  
}
```

(b)

RECOMMENDATIONS

Use the one that is most intuitive and comfortable for you. In general, a for loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times. A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0. A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.

NESTED LOOPS

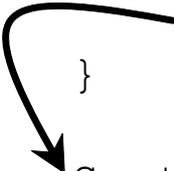
Problem: Write a program that uses nested for loops to print a multiplication table.

USING BREAK AND CONTINUE

Examples for using the break and continue keywords:

```
public class TestBreak {
    public static void main(String[] args) {
        int sum = 0;
        int number = 0;

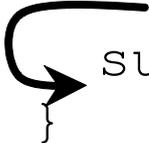
        while (number < 20) {
            number++;
            sum += number;
            if (sum >= 100)
                break;
        }
        System.out.println("The number is " + number);
        System.out.println("The sum is " + sum);
    }
}
```



```
public class TestContinue {
    public static void main(String[] args) {
        int sum = 0;
        int number = 0;

        while (number < 20) {
            number++;
            if (number == 10 || number == 11)
                continue;
            sum += number;
        }

        System.out.println("The sum is " + sum);
    }
}
```



(GUI) CONTROLLING A LOOP WITH A CONFIRMATION DIALOG (SELF READING)

A sentinel-controlled loop can be implemented using a confirmation dialog. The answers *Yes* or *No* to continue or terminate the loop. The template of the loop may look as follows:

```
int option = 0;
while (option == JOptionPane.YES_OPTION) {
    System.out.println("continue loop");
    option = JOptionPane.showConfirmDialog(null, "Continue?");
}
```