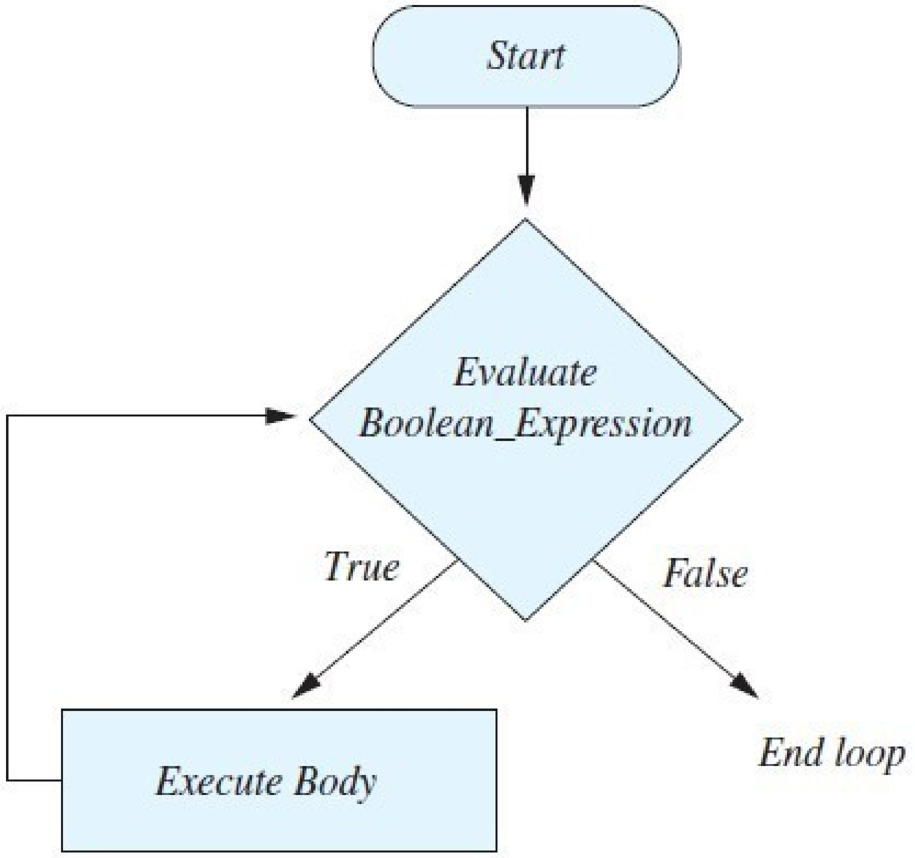
The `while` Statement

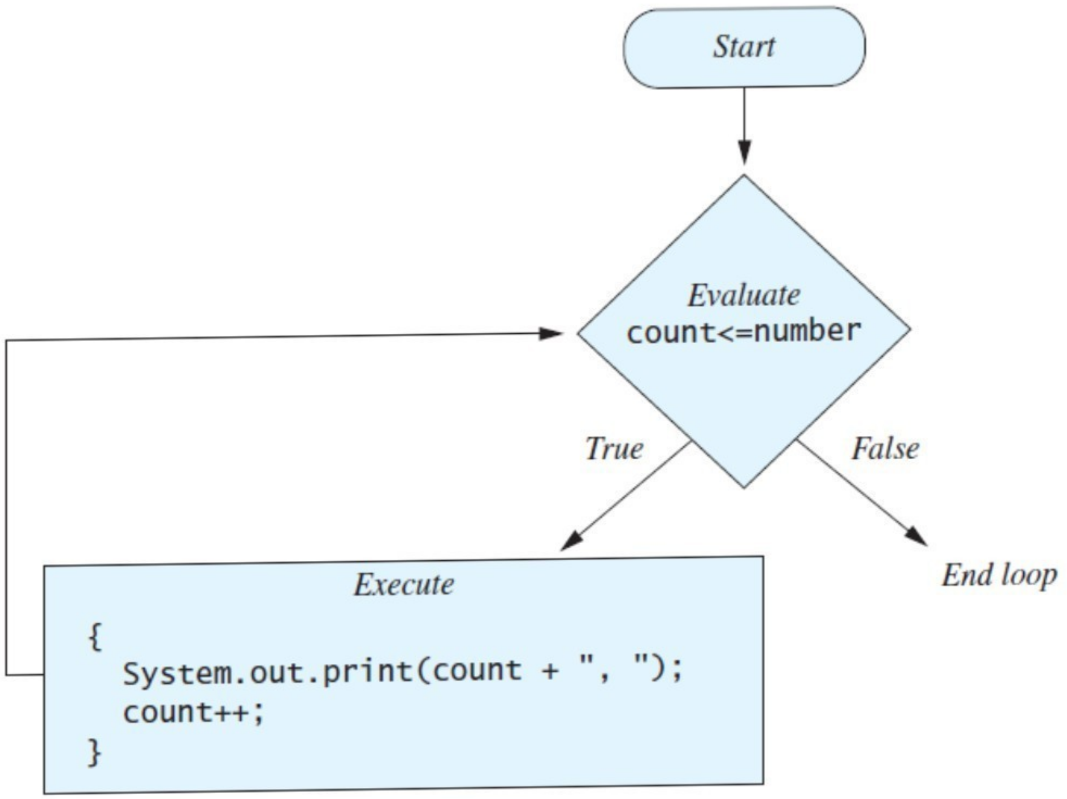
```
while (Boolean_Expression)
    Body
```



The `while` Statement

- Figure 4.1

```
while (count <= number)
{
    System.out.print(count + ", ");
    count++;
}
```



```
{
  System.out.print(count + ", ");
  count++;
}
```

The **`do-while`** Statement

- Also called a **`do-while`** loop
- Similar to a **`while`** statement, except that the loop body is executed at least once
- Syntax

```
do
   Body_Statement
while (Boolean_Expression);
```

- Don’t forget the semicolon!

The `do-while` Statement

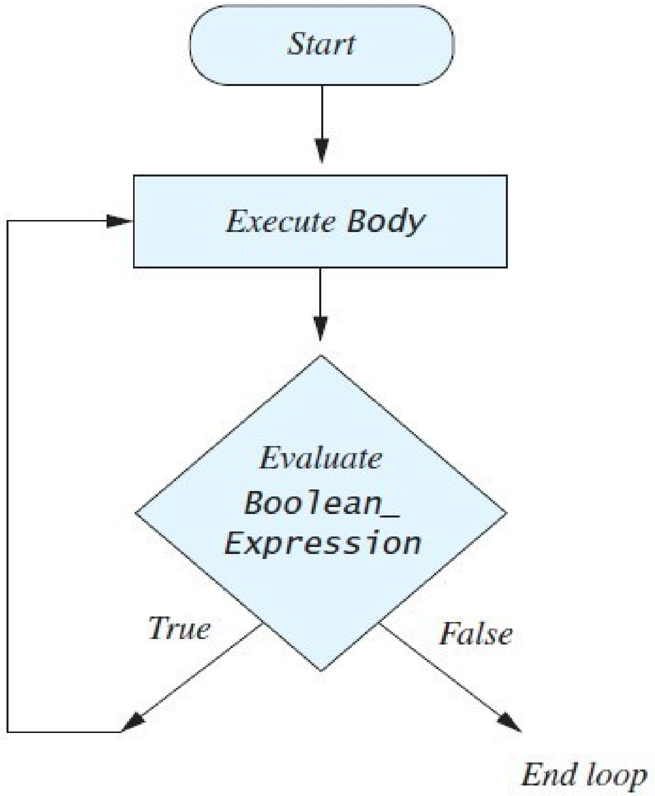
- First, the loop body is executed.
- Then the boolean expression is checked.
  - As long as it is true, the loop is executed again.
  - If it is false, the loop is exited.
- Equivalent `while` statement

```
Statement(s)_S1
while (Boolean_Condition)
    Statement(s)_S1
```

The do-while Statement

- The Semantics of the do-while Statement

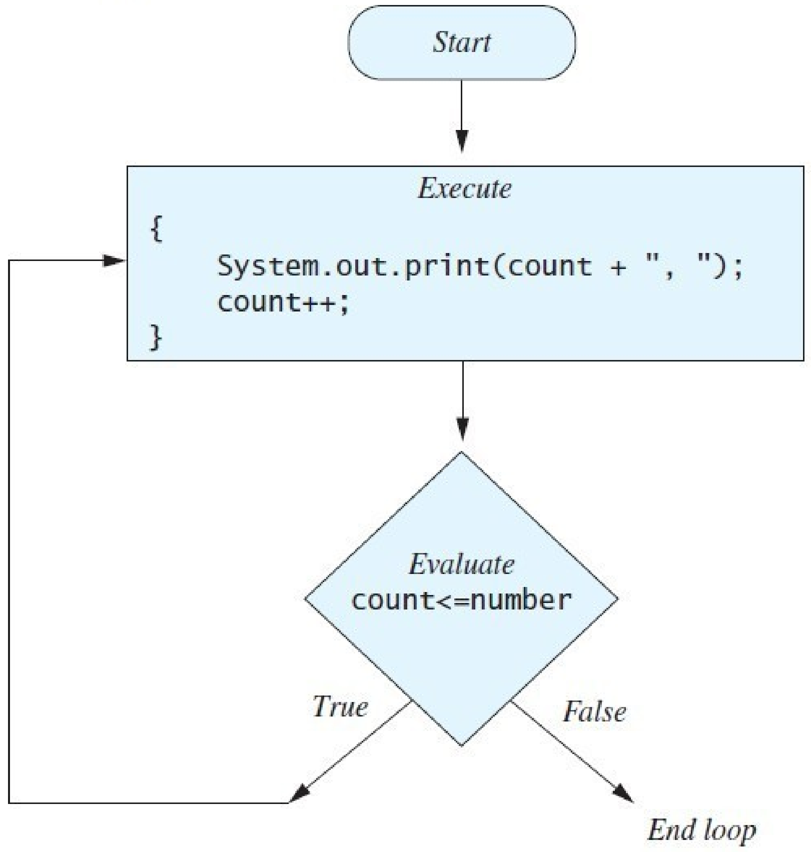
```
do
     Body
while (Boolean_Expression)
```



The **do-while** Statement

- The Action of the **do-while** Loop

```
do
{
   System.out.print(count + ", ");
   count++;
} while (count <= number);
```



```
{
   System.out.print(count + ", ");
   count++;
}
```

Infinite Loops

- A loop which repeats without ever ending is called an infinite loop.
- If the controlling boolean expression never becomes false, a **`while`** loop or a **`do-while`** loop will repeat without ending.

Nested Loops

- The body of a loop can contain any kind of statements, including another loop.

Negating a Boolean Expression

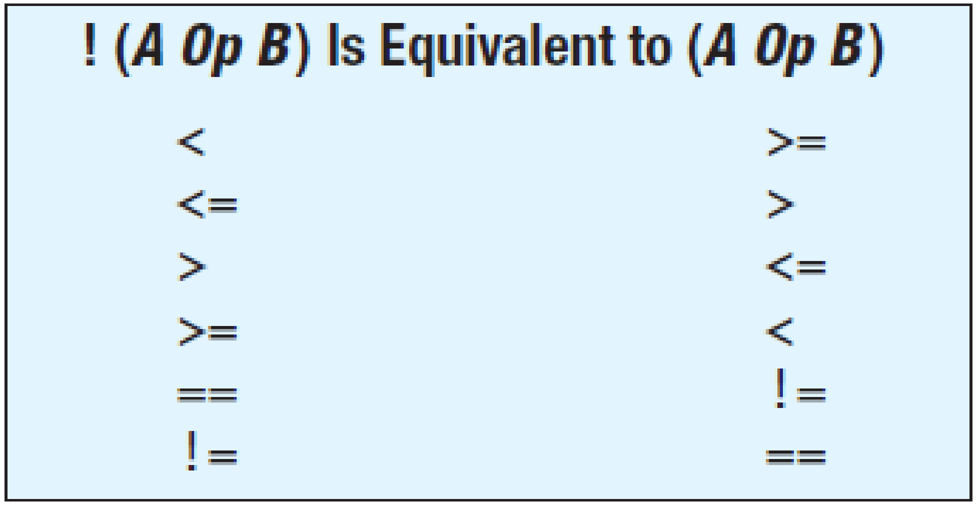
- A boolean expression can be negated using the "not" (`!`) operator.
- Syntax

  ```
  !(Boolean_Expression)
  ```
- Example

  ```
  (a || b) && !(a && b)
  ```
  which is the exclusive or

Negating a Boolean Expression

- Figure 3.5 Avoiding the Negation Operator



! (*A Op B*) Is Equivalent to	(*A Op B*)
<	>=
<=	>
>	<=
>=	<
==	!=
!=	==

The `for` Statement

- A `for` statement executes the body of a loop a fixed number of times.
- Example

```
for (count = 1; count < 3; count++)
    System.out.println(count);
```

The `for` Statement

- Syntax

```
for (Initialization, Condition, Update)
   Body_Statement
```

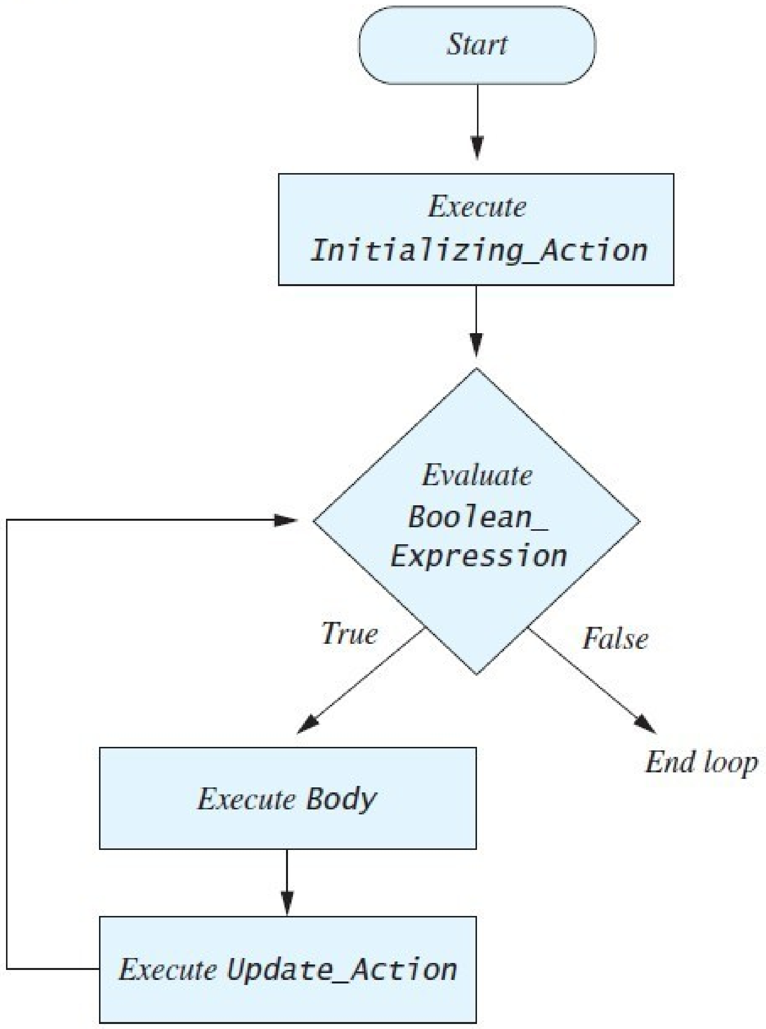
- `Body_Statement` can be either a simple statement or a compound statement in `{}`.
- Corresponding `while` statement

```
Initialization
while (Condition)
   Body_Statement_Including_Update
```

The **for** Statement

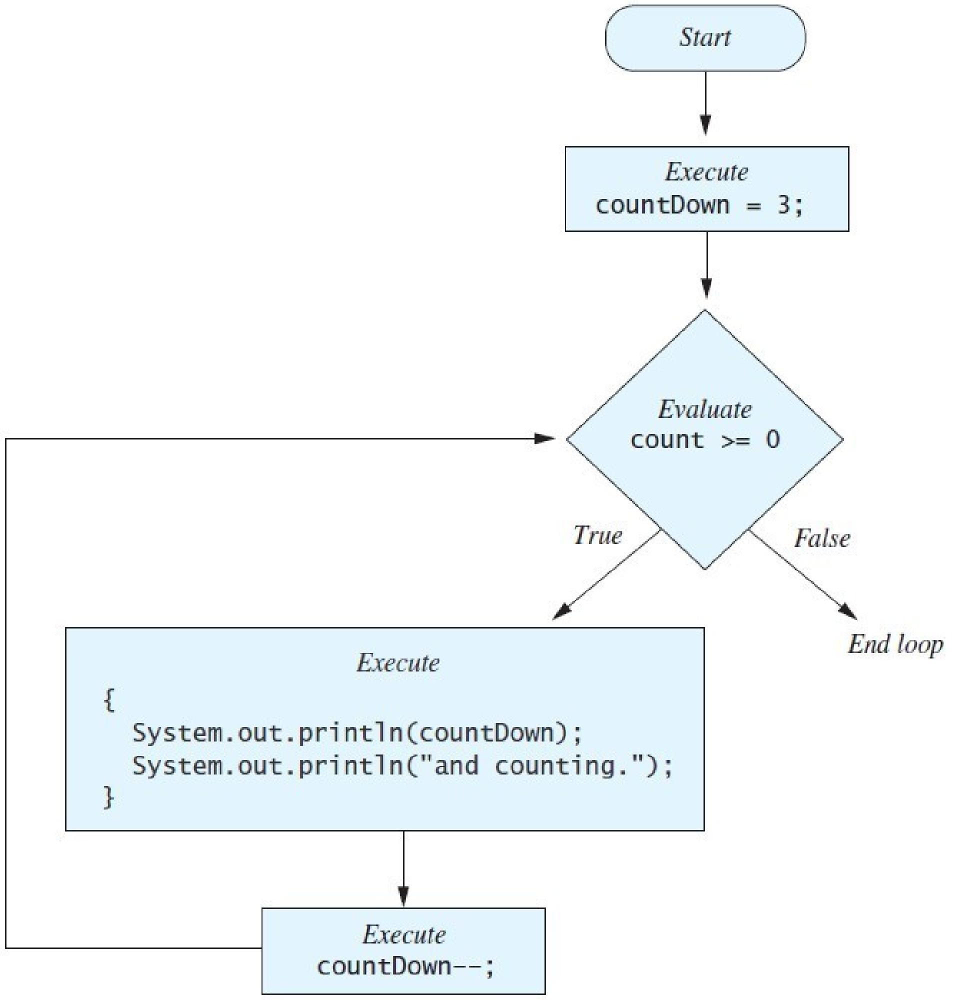
- The semantics of the **for** statement

```
for (Initializing_Action; Boolean_Expression; Update_Action)
    Body
```



The **for** Statement

- Figure 4.5



```
{
  System.out.println(countDown);
  System.out.println("and counting.");
}
```

```
for (countDown = 3; countDown >= 0; countDown--)
{
    System.out.println(countDown);
    System.out.println("and counting.");
}
```

The `for` Statement

- Possible to declare variables within a `for` statement

```
int sum = 0;
for (int n = 1 ; n <= 10 ; n++)
    sum = sum + n * n;
```

- Note that variable `n` is local to the loop

The `for` Statement

- A comma separates multiple initializations
- Example

```
for (n = 1, product = 1; n <= 10; n++)
    product = product * n;
```

- Only one boolean expression is allowed, but it can consist of `&&`s, `||`s, and `!`s.
- Multiple update actions are allowed, too.

```
for (n = 1, product = 1; n <= 10;
        product = product * n, n++);
```

Controlling Number of Loop Iterations

- If the number of iterations is known before the loop starts, the loop is called a count-controlled loop.
  - Use a `for` loop.
- Asking the user before each iteration if it is time to end the loop is called the ask-before-iterating technique.
  - Appropriate for a small number of iterations
  - Use a `while` loop or a `do-while` loop.