Loop Statements (Chapter 4)

- The ***while*** statement
- The ***do-while*** statement
- The ***for*** statement

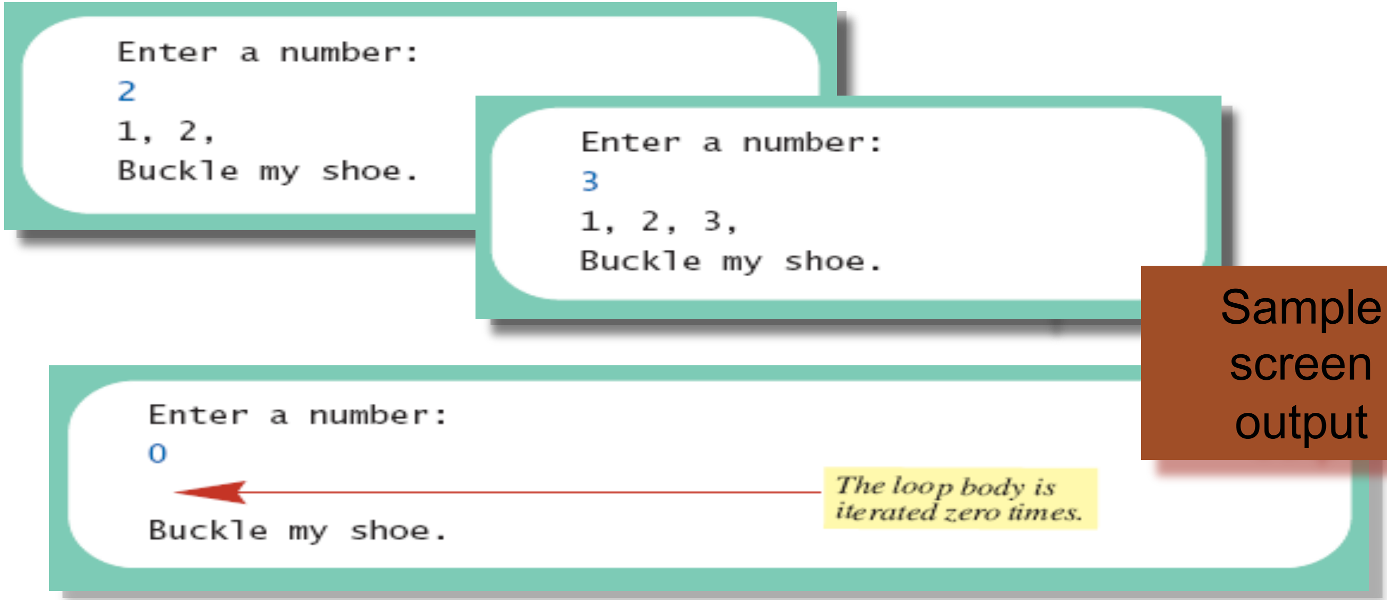
Java Loop Statements

- A portion of a program that repeats a statement or a group of statements is called a *loop.*
- The statement or group of statements to be repeated is called the *body* of the loop.
- There must be a means of exiting the loop.

The **while** Statement

- Also called a **while** loop
- A **while** statement repeats while a controlling boolean expression remains true
- The loop body typically contains an action that ultimately causes the controlling boolean expression to become false.

The while Statement – sample program



```
Enter a number:
2
1, 2,
Buckle my shoe.
```

```
Enter a number:
3
1, 2, 3,
Buckle my shoe.
```

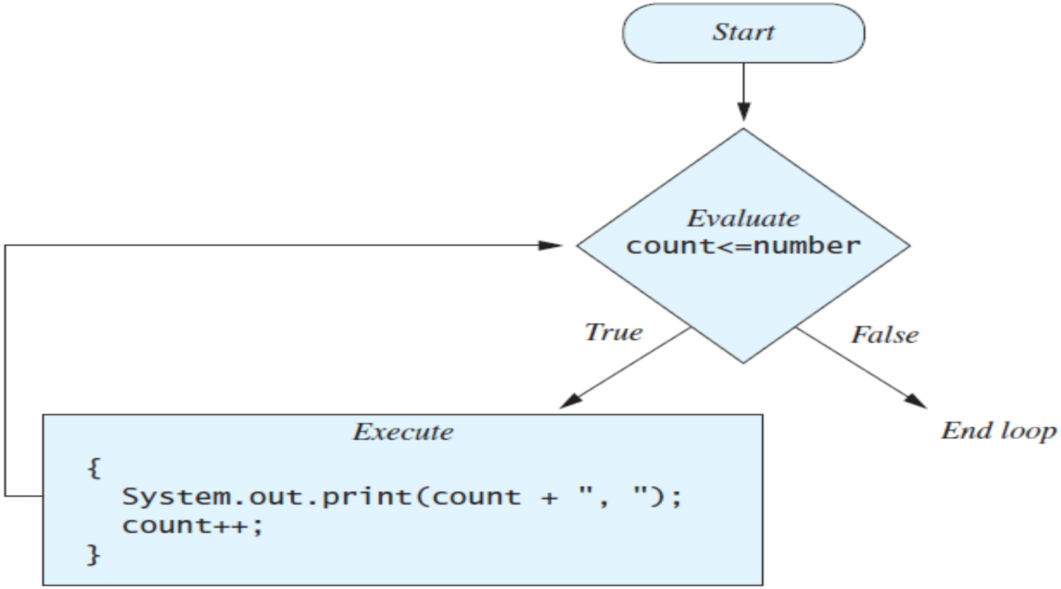
```
Enter a number:
0

Buckle my shoe.
```

The loop body is iterated zero times.

Sample screen output

The while Statement – code

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



Start

Evaluate
count<=number

True

False

End loop

Execute

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

The while Statement – syntax

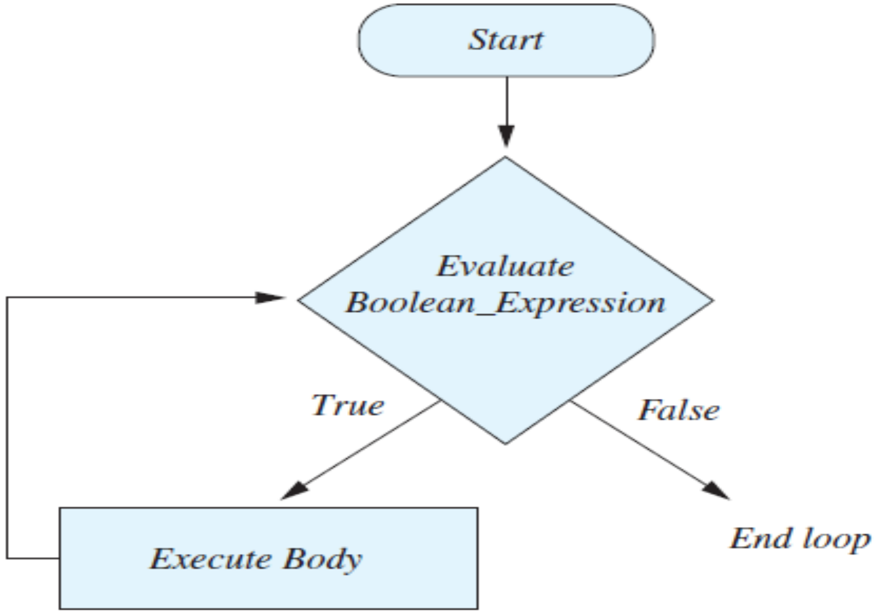
```
while (Boolean_Expression)
    Body_Statement;
```

or

```
while (Boolean_Expression)
{
    First_Statement;
    Second_Statement;
    …
}
```

The while Statement – semantics

```
while (Boolean_Expression)
      Body
```



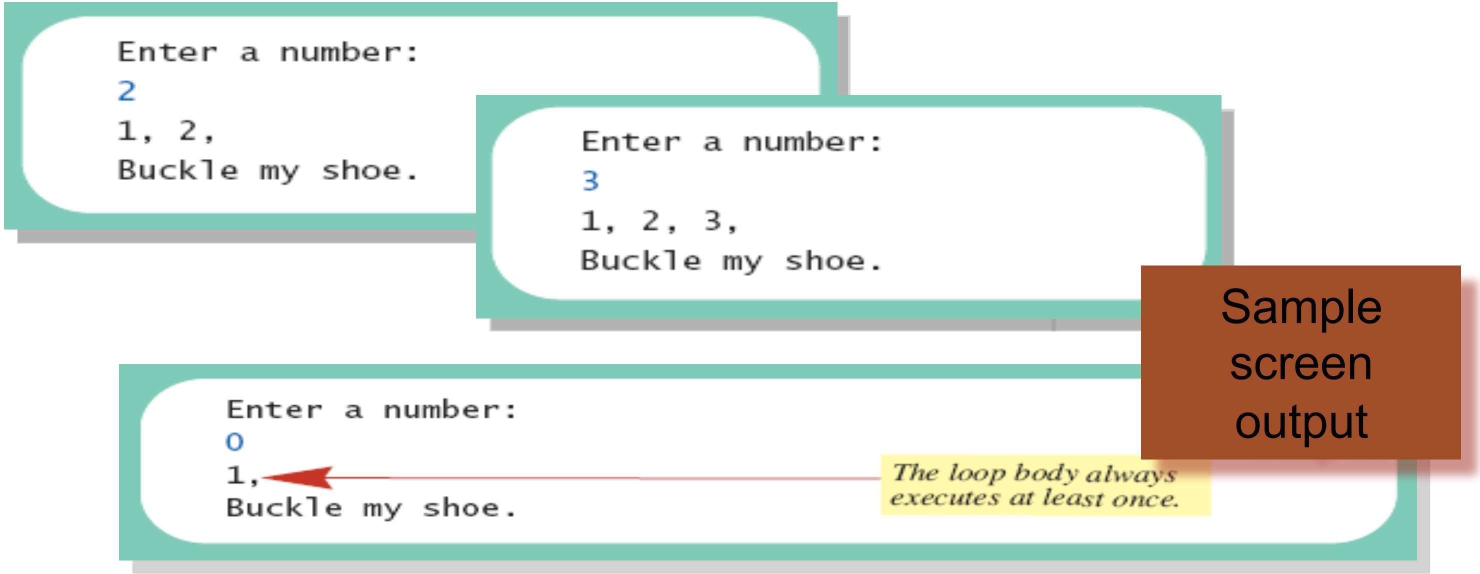
The do-while Statement

- 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 – sample program



```
Enter a number:
2
1, 2,
Buckle my shoe.
```

```
Enter a number:
3
1, 2, 3,
Buckle my shoe.
```

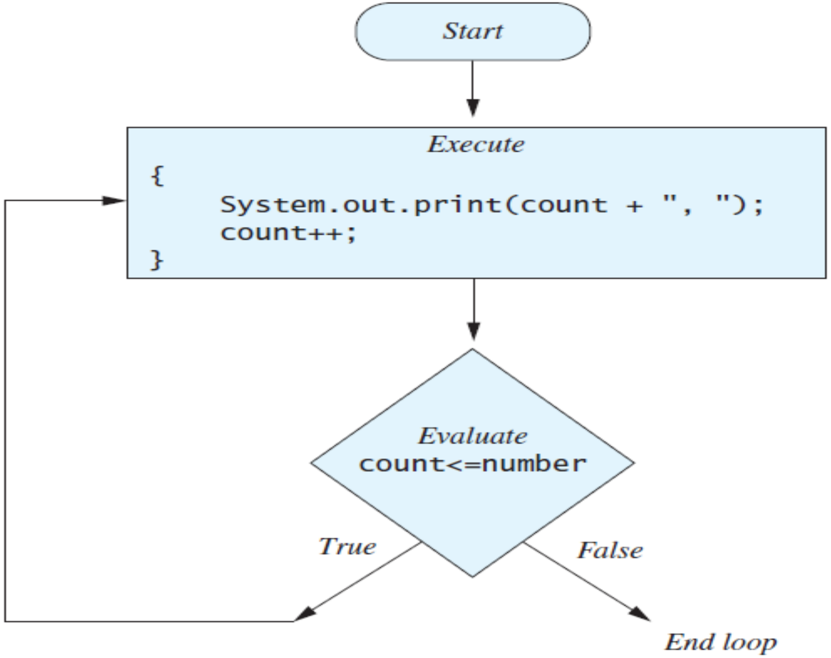
```
Enter a number:
0
1,
Buckle my shoe.
```

The loop body always executes at least once.

Sample screen output

The do-while Statement – code

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



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

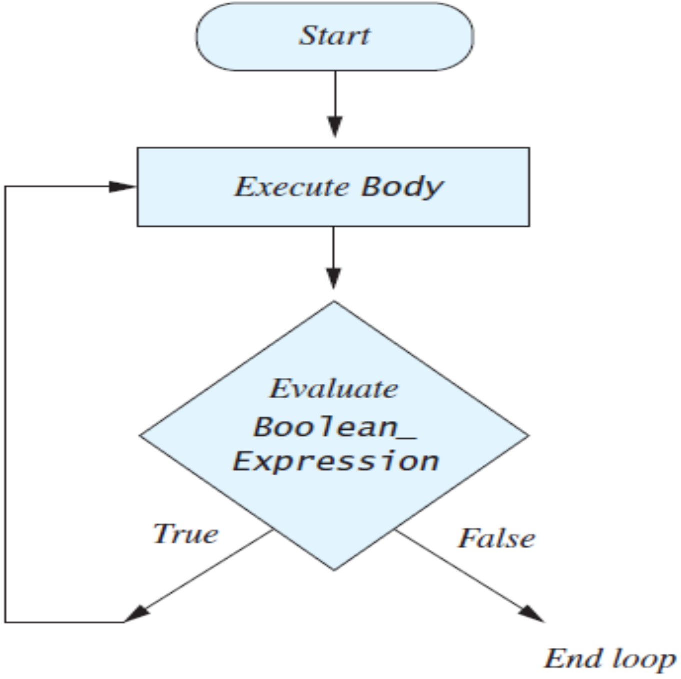
The do-while Statement

- 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 – semantics

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



Programming Exercise

- Given
  - 초파리 부피: 0.002 cubic feet
  - 초기 초파리 개체수
  - 초파리 개체 증가 속도: 95%/week
  - 집 크기

- Find
  - Number of hours to exceed the capacity of the house
  - Number and volume of fruit flies

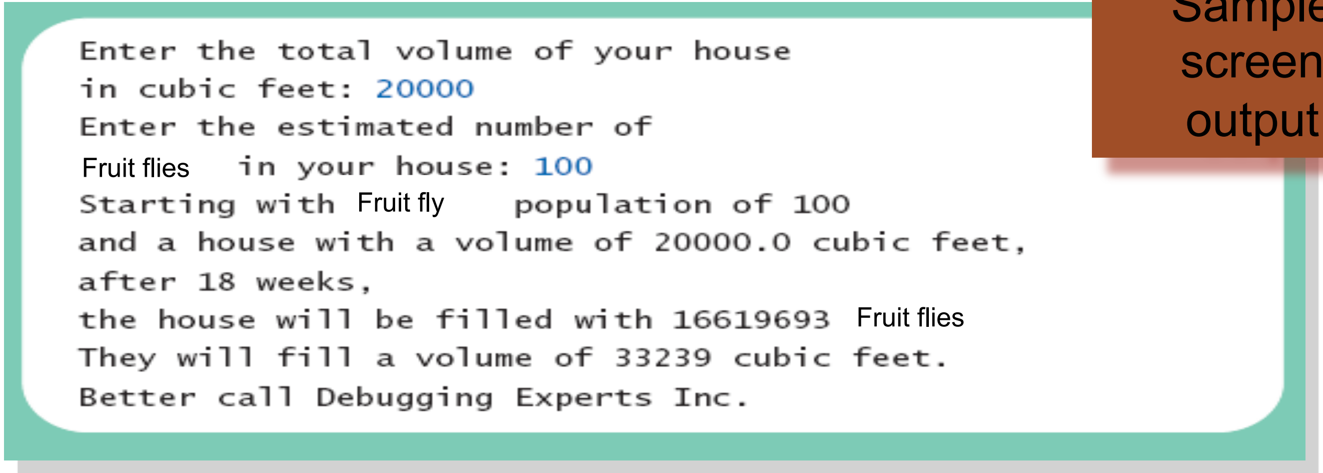
Programming Exercise – algorithm (rough draft)

- Get volume of fruit fly
- Get initial number of fruit flies in house
- Compute number of weeks until the house is full of fruit flies
- Display results

Programming Exercise – variables needed

- GROWTH_RATE — weekly growth rate of the fruit fly (a constant 0.95)
- ONE_FLY_VOLUME — volume of an average fly (a constant 0.002)
- houseVolume — volume of the house
- initNumFlies — initial number of fruit flies
- countWeeks — week counter
- Population — current number of fruit flies
- totalFlyVolume — total volume of all the flies
- newFlies — number of flies hatched this week
- newFlyVolume — volume of new flies

Programming Exercise – output example



```
Enter the total volume of your house
in cubic feet: 20000
Enter the estimated number of
Fruit flies   in your house: 100
Starting with Fruit fly   population of 100
and a house with a volume of 20000.0 cubic feet,
after 18 weeks,
the house will be filled with 16619693 Fruit flies
They will fill a volume of 33239 cubic feet.
Better call Debugging Experts Inc.
```

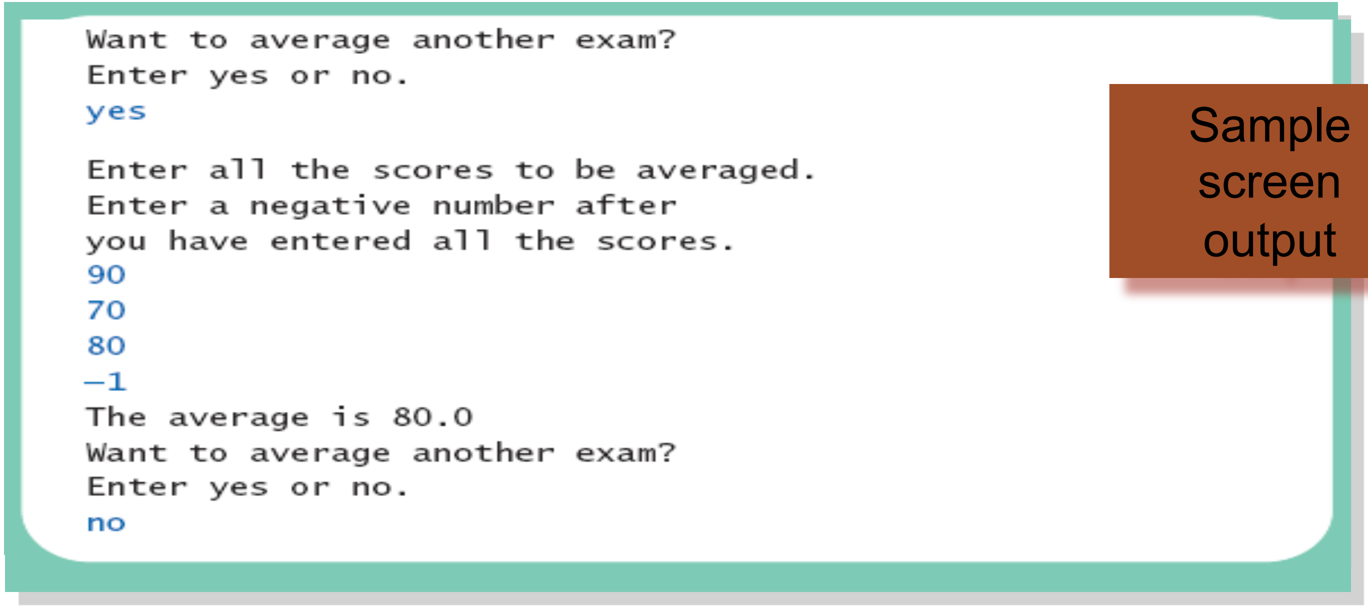
Sample screen output

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.
- A negative growth rate in the preceding problem causes **totalFlyVolume** always to be less than **houseVolume**, so that the loop never ends.

Nested Loops

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



```
Want to average another exam?
Enter yes or no.
yes

Enter all the scores to be averaged.
Enter a negative number after
you have entered all the scores.
90
70
80
-1
The average is 80.0
Want to average another exam?
Enter yes or no.
no
```

Sample screen output

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);
```

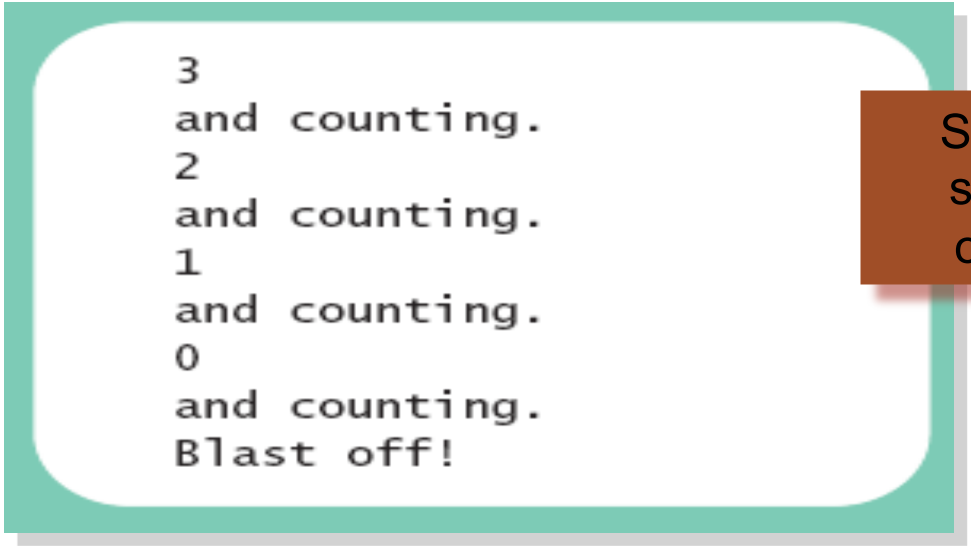
The for Statement – syntax

```
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
```

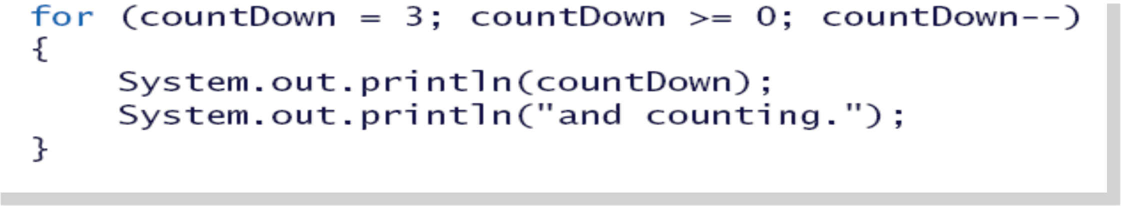
The for Statement – sample program



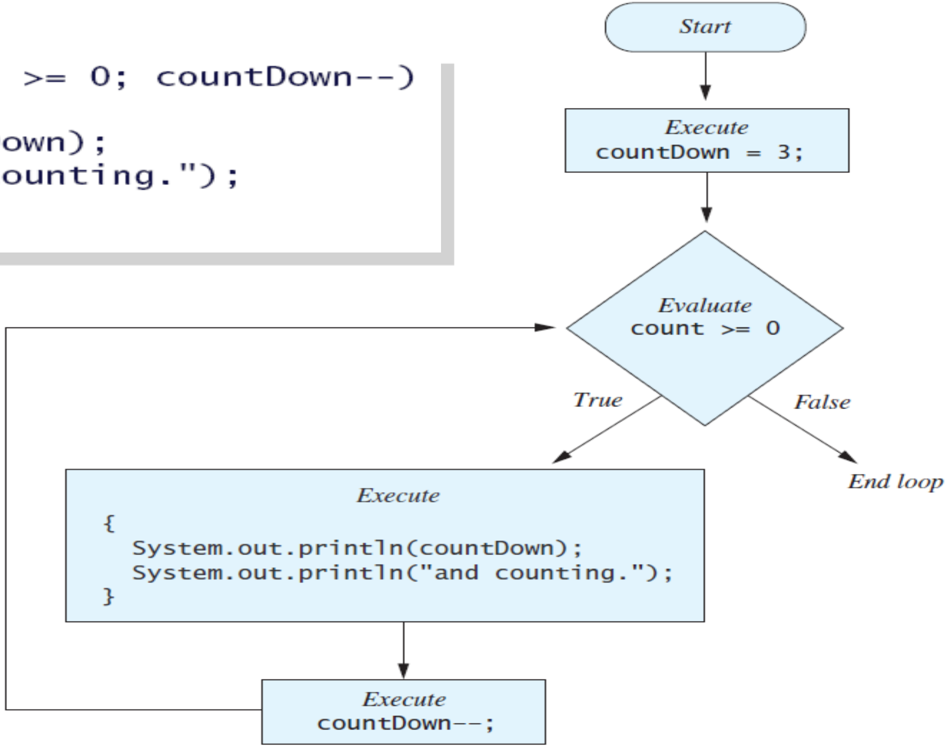
```
3
and counting.
2
and counting.
1
and counting.
0
and counting.
Blast off!
```

Sample screen output

The for Statement – action



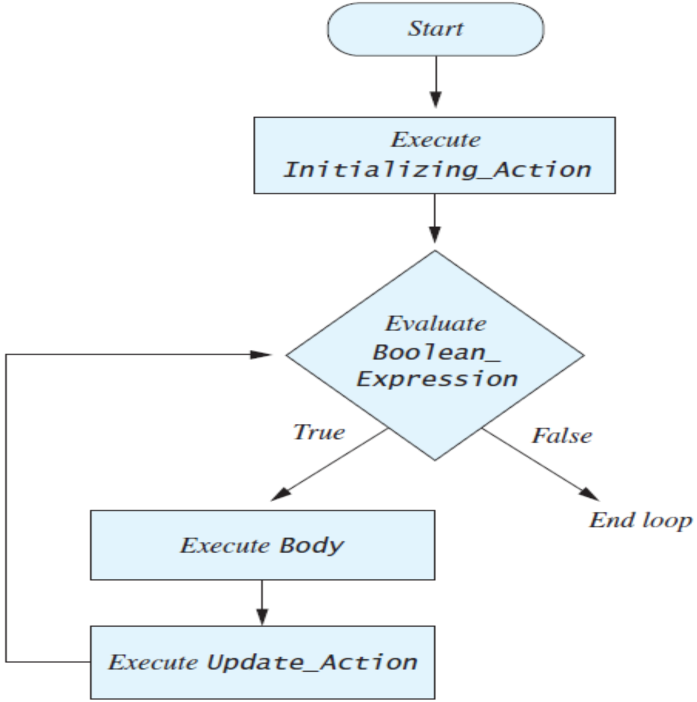
```
for (countDown = 3; countDown >= 0; countDown--)
{
    System.out.println(countDown);
    System.out.println("and counting.");
}
```



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

The for Statement – semantics

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



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++);
```

The for-each Statement

- Possible to step through values of an enumeration type
- Example

```
enum Suit {CLUBS, DIAMONDS, HEARTS, SPADES}
for (Suit nextSuit : Suit.values())
System.out.print(nextSuit + " ");
System.out.println();
```

Programming with Loops: Outline

The Loop Body

Initializing Statements

Controlling Loop Iterations

break and **continue** statements

Loop Bugs

Tracing Variables

Assertion checks

The Loop Body

- To design the loop body, write out the actions the code must accomplish.
- Then look for a repeated pattern.
  - The pattern need not start with the first action.
  - The repeated pattern will form the body of the loop.
  - Some actions may need to be done after the pattern stops repeating.

Initializing Statements

Some variables need to have a value before the loop begins.

- Sometimes this is determined by what is supposed to happen after one loop iteration.
- Often variables have an initial value of zero or one, but not always.

Other variables get values only while the loop is iterating.

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.

Controlling Number of Loop Iterations

- For large input lists, a ***sentinel value*** can be used to signal the end of the list.
  - The sentinel value must be different from all the other possible inputs.
  - A negative number following a long list of nonnegative exam scores could be suitable.

90

0

10

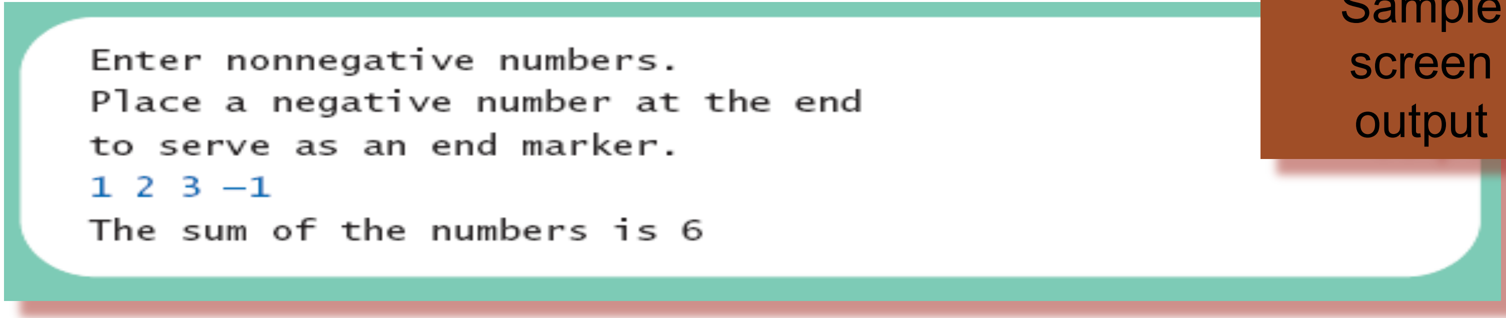
-1 ← Sentinel value

Controlling Number of Loop Iterations

- Example - reading a list of scores followed by a sentinel value

```
int next = keyboard.nextInt();
while (next  >= 0)
{
        Process_The_Score
        next = keyboard.nextInt();
}
```

Controlling Number of Loop Iterations



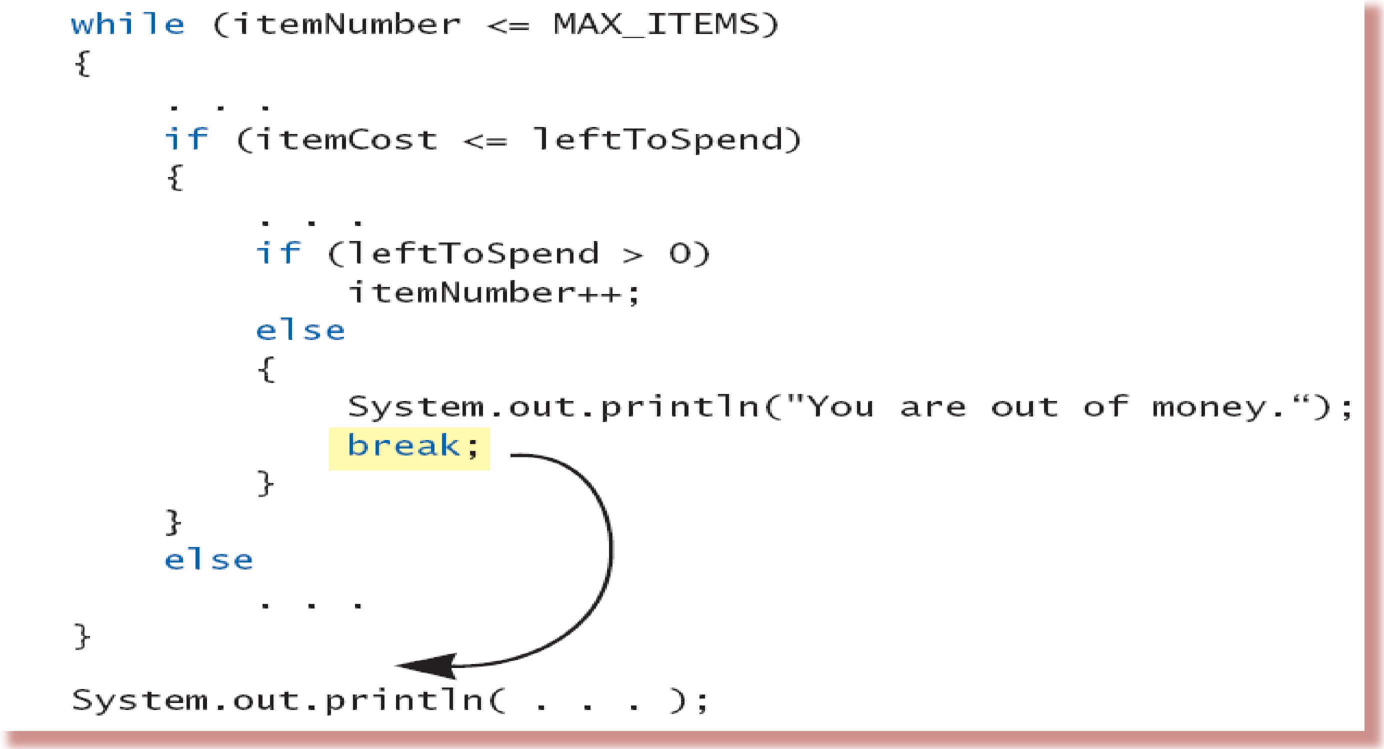
```
Enter nonnegative numbers.
Place a negative number at the end
to serve as an end marker.
1 2 3 -1
The sum of the numbers is 6
```

Sample screen output

The break Statement in Loops

- A **break** statement can be used to end a loop immediately.
- The **break** statement ends only the **innermost** loop or switch statement that contains the **break** statement.
- **break** statements make loops more difficult to understand.
- Use **break** statements sparingly (if ever).

The break Statement in Loops



```
while (itemNumber <= MAX_ITEMS)
{
     . . .
     if (itemCost <= leftToSpend)
     {
          . . .
          if (leftToSpend > 0)
               itemNumber++;
          else
          {
               System.out.println("You are out of money.“);
               break;
          }
     }
     else
          . . .
}

System.out.println( . . . );
```

The continue Statement in Loops

- A **continue** statement
  - Ends current loop iteration
  - Begins the next one
- Text recommends avoiding use
  - Introduce unneeded complications

Tracing Variables

- *Tracing variables* means watching the variables change while the program is running.
  - Simply insert temporary output statements in your program to print of the values of variables of interest
  - Or, learn to use the debugging facility that may be provided by your system.

Loop Bugs

- Common loop bugs
  - Unintended infinite loops
  - Off-by-one errors
  - Testing equality of floating-point numbers

- Subtle infinite loops
  - The loop may terminate for some input values, but not for others.
  - For example, you can't get out of debt when the monthly penalty exceeds the monthly payment.