4. Flow of Control: Loops

Harald Gall, Prof. Dr.

Institut für Informatik

Universität Zürich

http://seal.ifi.uzh.ch/info1

University of Zurich
Department of Informatics

s.e.a.l.
software evolution & architecture lab

Objectives

- Design a loop
- Use `while`, and `for` in a program

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.
- A loop could be used to compute grades for each student in a class.
- There must be a means of exiting the loop.

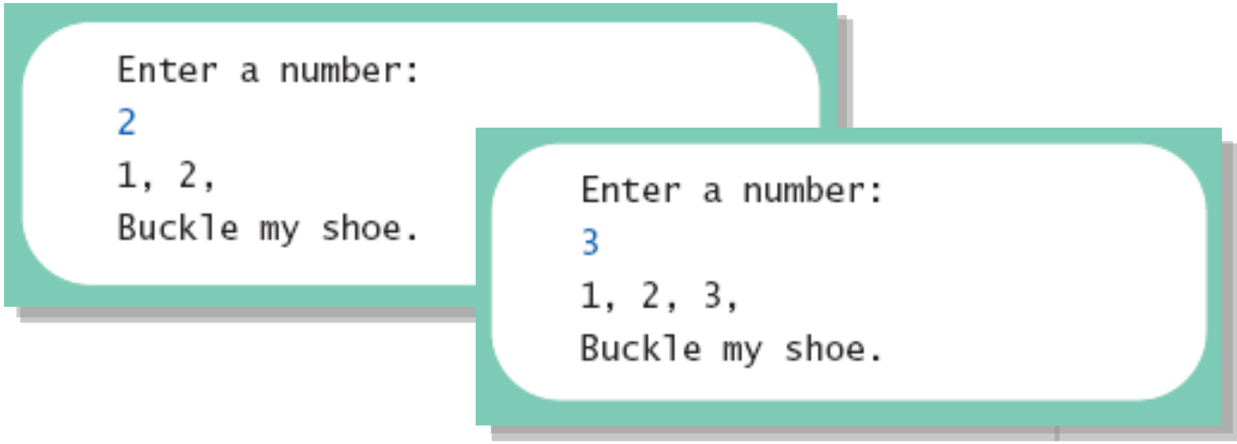
The `while` loop

- A `while` loop repeats while a controlling boolean expression remains true
  - If the controlling boolean expression is false initially, the while loop is not executed
- The loop body typically contains an action that ultimately causes the controlling boolean expression to become false.

The `while` loop

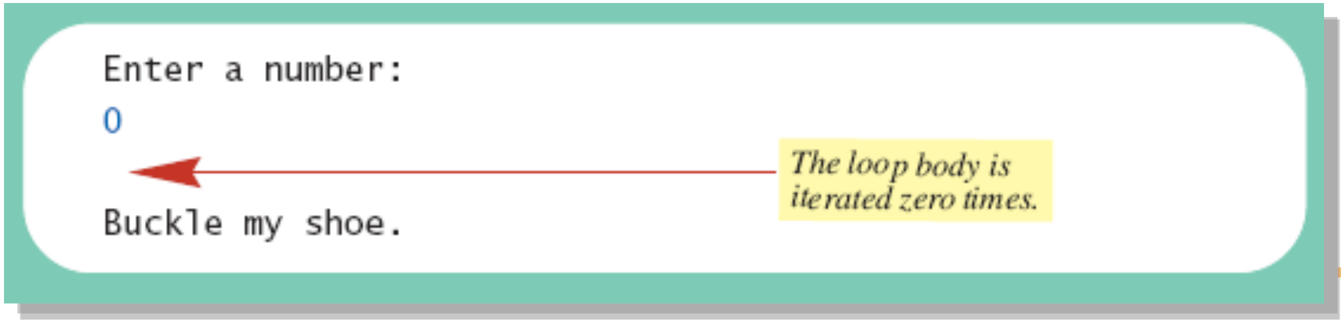
- Sample program

  `class WhileDemo`



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

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

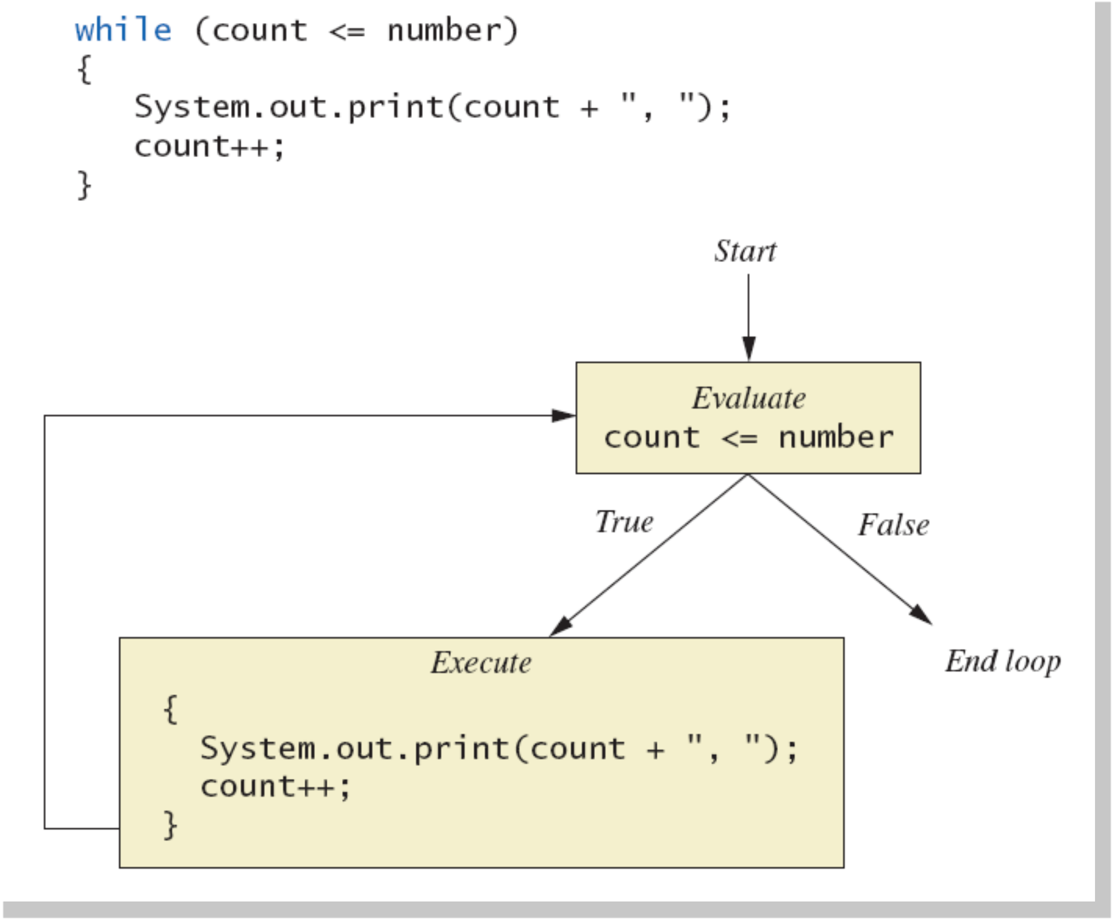


```
Enter a number:
0

Buckle my shoe.
```

The loop body is iterated zero times.

The `while` loop



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

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

University of Zurich
Department of Informatics

The `while` loop

- Syntax

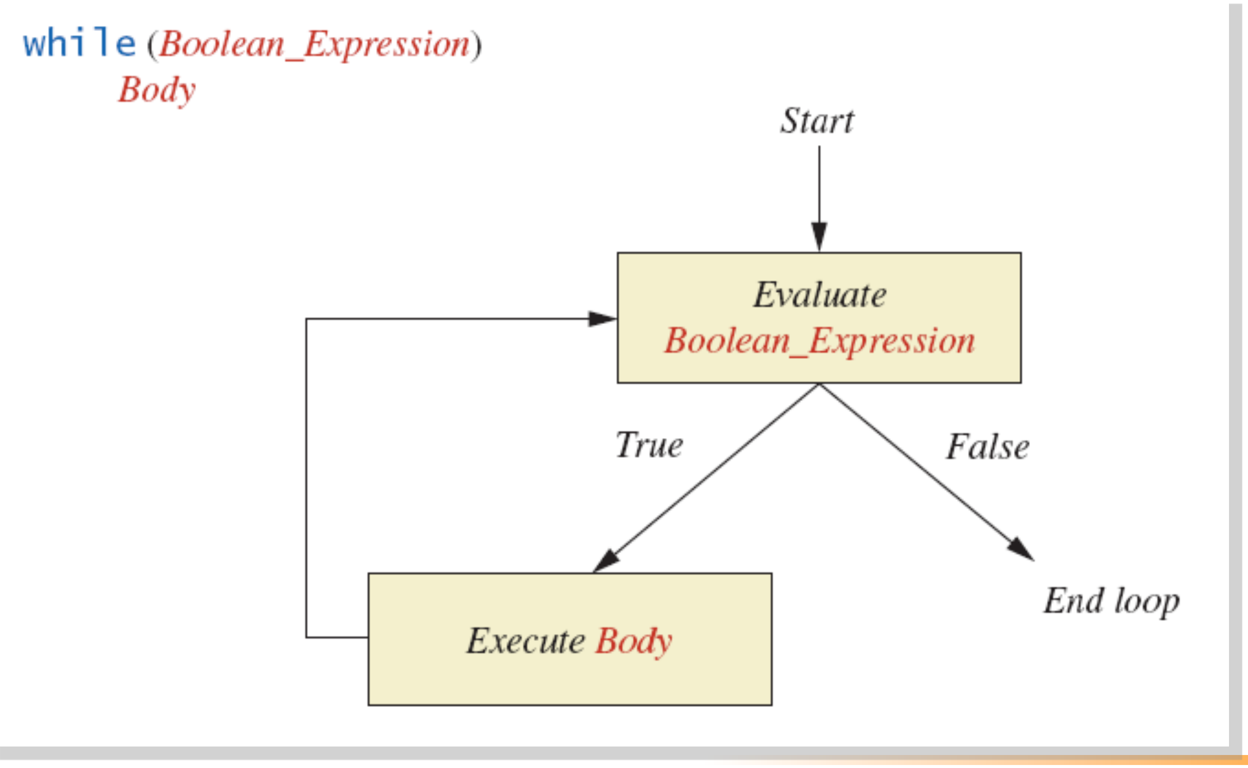
```
while (Boolean_Expression)
   Body_Statement;
```

or

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

The `while` loop

- Semantics of the `while` statement



```
while (Boolean_Expression)
      Body
```

The `do-while` loop

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

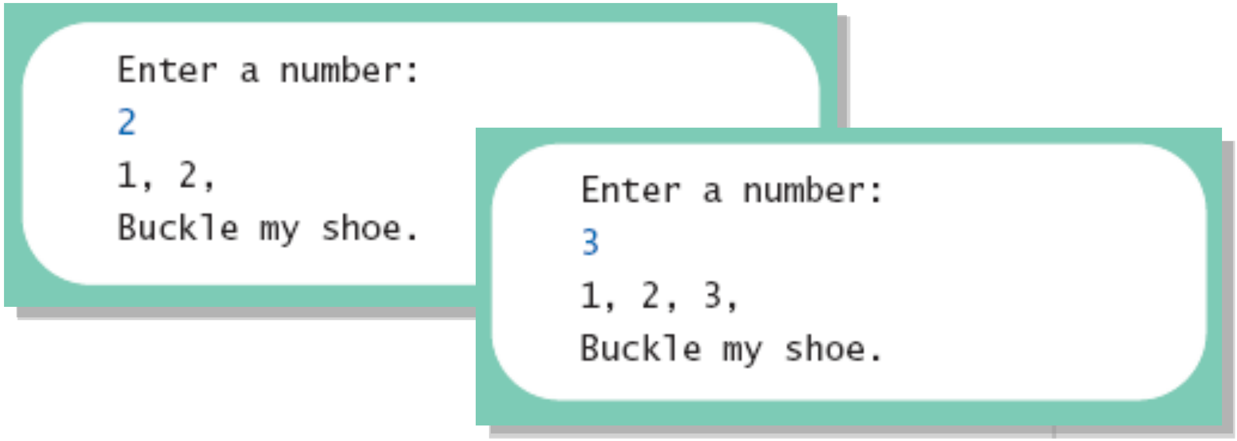
```
do
   Body_Statement
while (Boolean_Expression);
```

- Don’t forget the semicolon!

The `do-while` loop

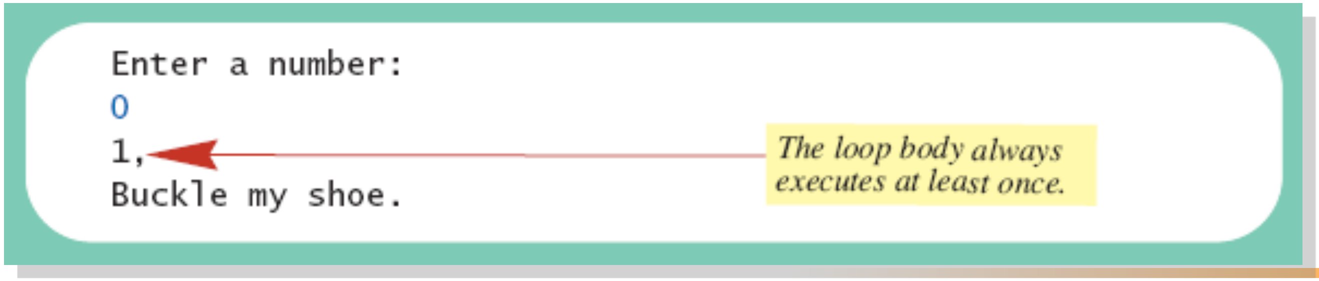
- View sample program, listing 4.2

`class DoWhileDemo`



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

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

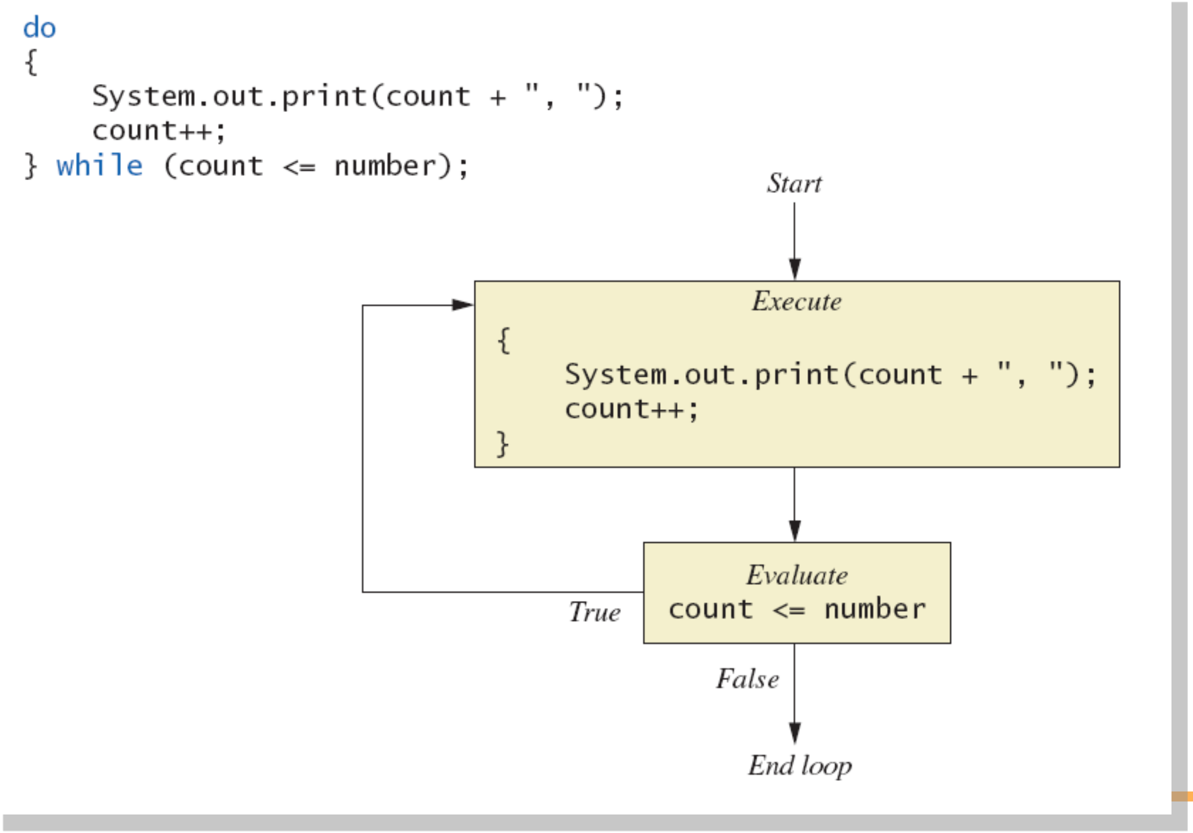


```
Enter a number:
0
1,
Buckle my shoe.
```

The loop body always executes at least once.

The `do-while` loop

- Figure 4.3 The Action of the `do-while` Loop in Listing 4.2



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

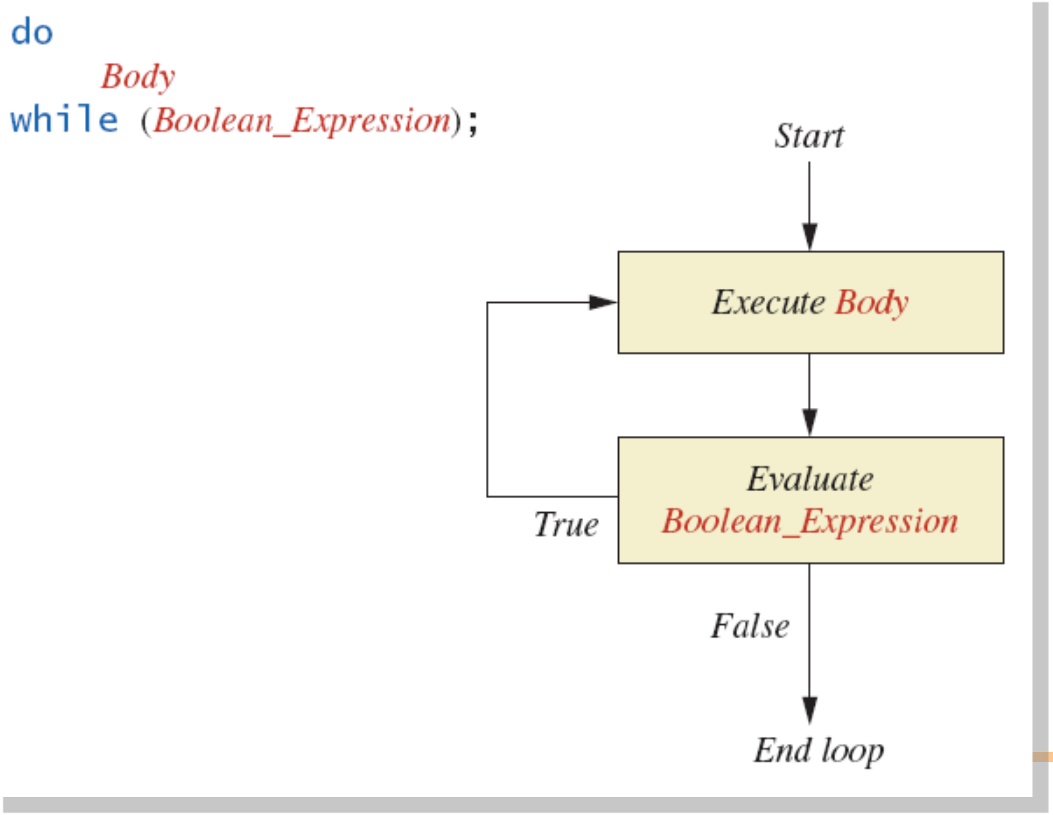
The `do-while` loop

- 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` loop

- The Semantics of the `do-while`



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

Example: Bug Infestation

- given
  - volume a roach: 0.002 cubic feet *(=0.6 mm3)*
  - starting roach population
  - rate of increase: 95% per week
  - volume of a house
- find
  - number of weeks to exceed the capacity of the house
  - number and volume of roaches

Example: Bug Infestation

Algorithm for roach population program (rough draft)

1. Get volume of house.
2. Get initial number of roaches in house.
3. Compute number of weeks until the house is full of roaches.
4. Display results.

Example: Bug Infestation

Variables Needed

GROWTH_RATE —weekly growth rate of the roach population (a constant 0.95)

ONE_BUG_VOLUME —volume of an average roach (a constant 0.002)

houseVolume — volume of the house

startPopulation —initial number of roaches

ctd. …

Example: Bug Infestation

Variables Needed

`countWeeks` —week counter

`Population` —current number of roaches

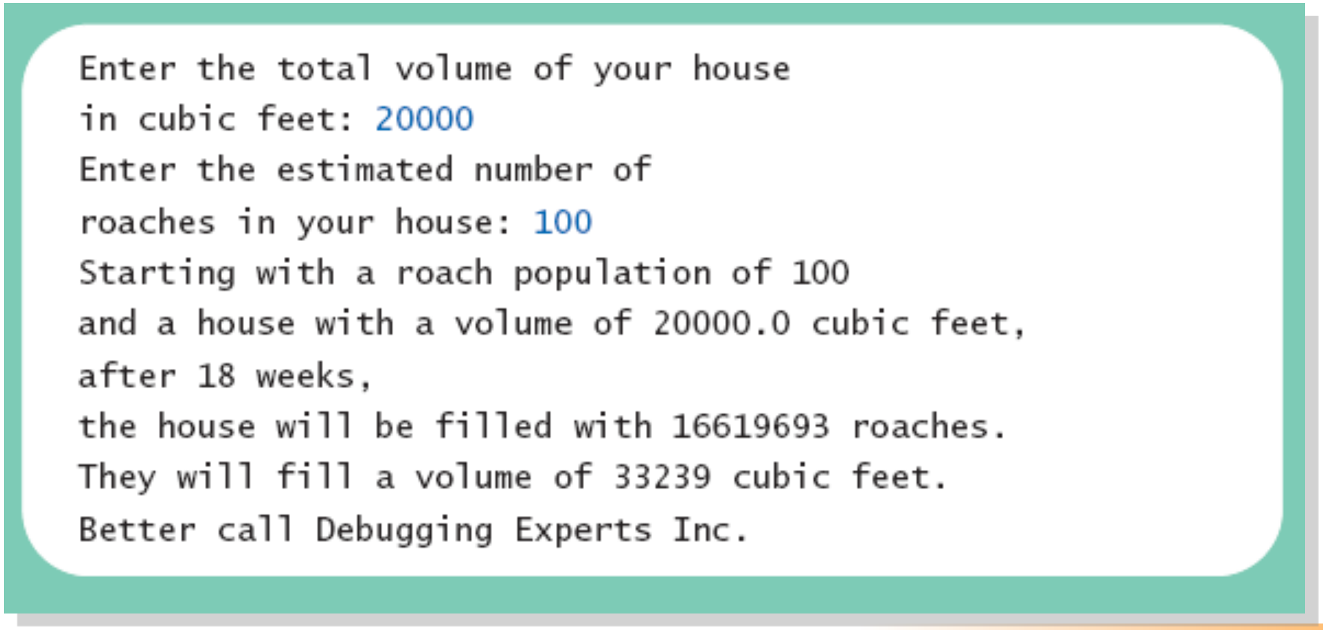
`totalBugVolume` —total volume of all the roaches

`newBugs` —number of roaches hatched this week

`newBugVolume` —volume of new roaches

Example: Bug Infestation

- View more detailed algorithm
- **`class BugTrouble`**



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

Infinite Loops

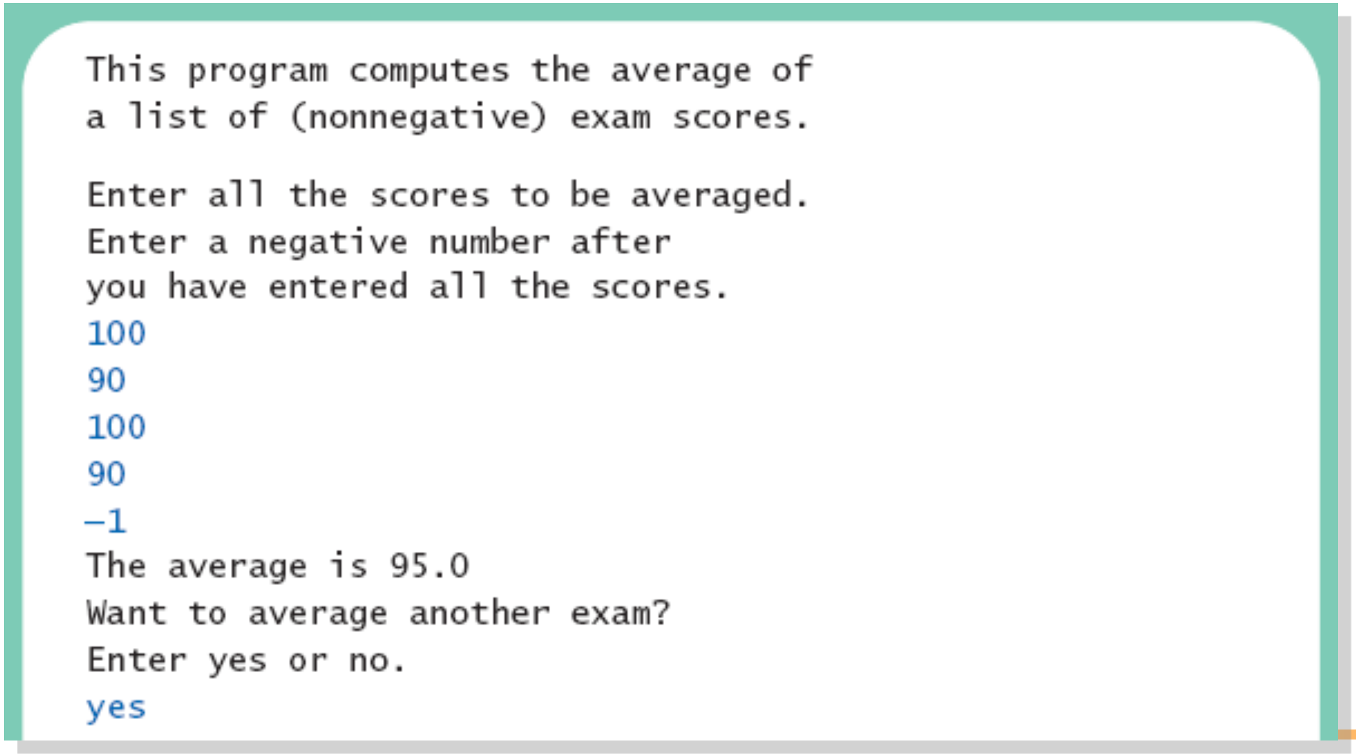
- A loop which repeats without ever ending is called an *infinite loop.*
- If the controlling boolean expression never becomes false, a **`while`** loop will repeat without ending.
- A negative growth rate in the preceding problem causes **`totalBugVolume`** always to be less than **`houseVolume`**, so that the loop never ends.

Nested Loops

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

Nested Loops

- Sample program
  `class ExamAverager`



```
This program computes the average of
a list of (nonnegative) exam scores.

Enter all the scores to be averaged.
Enter a negative number after
you have entered all the scores.
100
90
100
90
-1
The average is 95.0
Want to average another exam?
Enter yes or no.
yes
```

The `for` loop

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

```
for (int count = 100; count > 3; count--)
    System.out.println(count);

System.out.println(“Done”);
```

The `for` loop

- Syntax

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

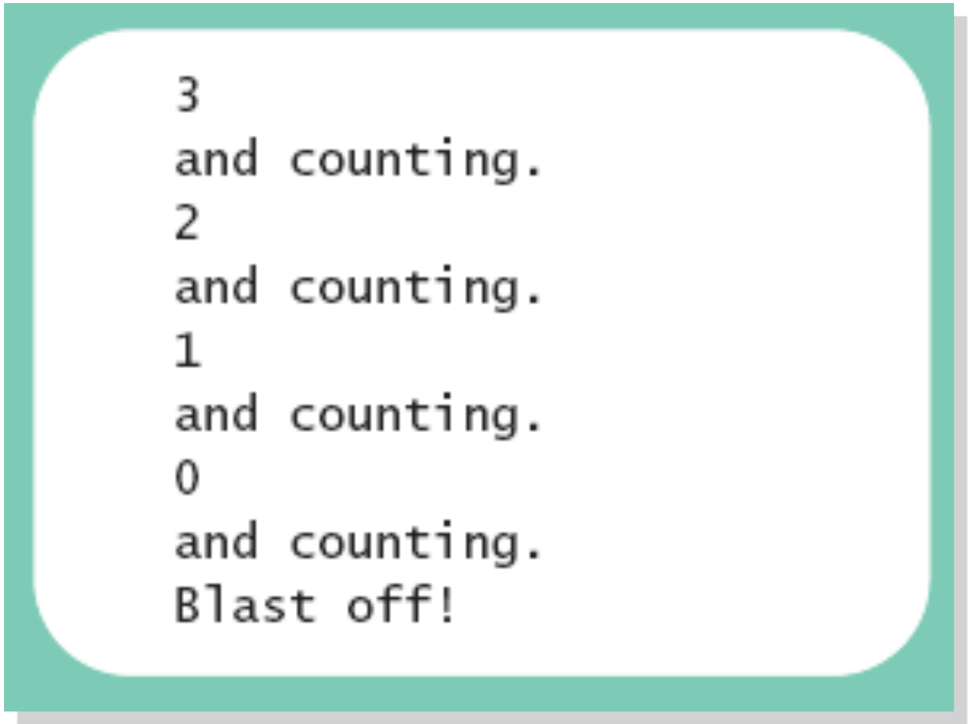
- `Body_Statement` can be either a simple statement or a compound statement in `{}`
- Corresponding `while` statement

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

The `for` loop

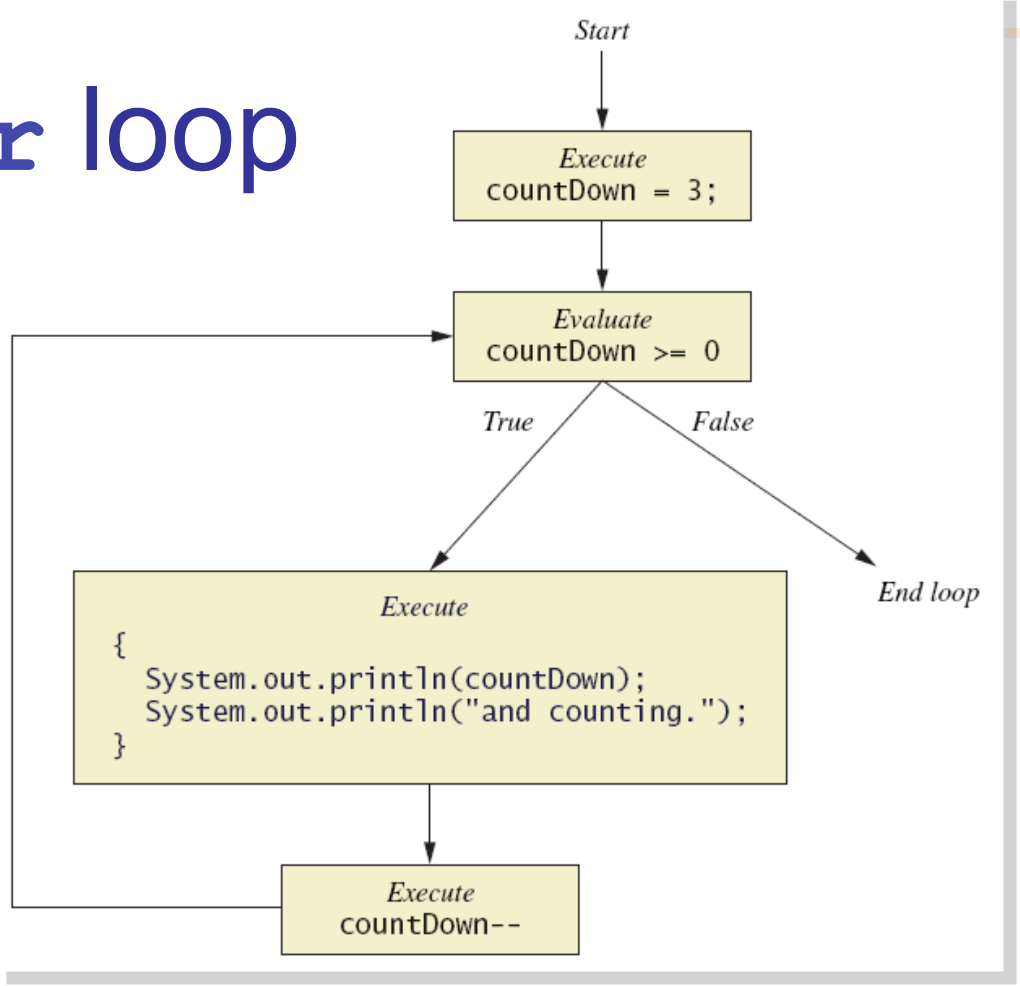
- Sample program

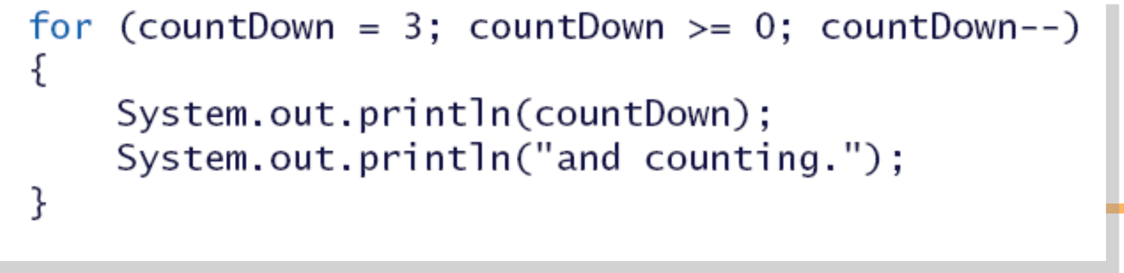
  `class CountDown`



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

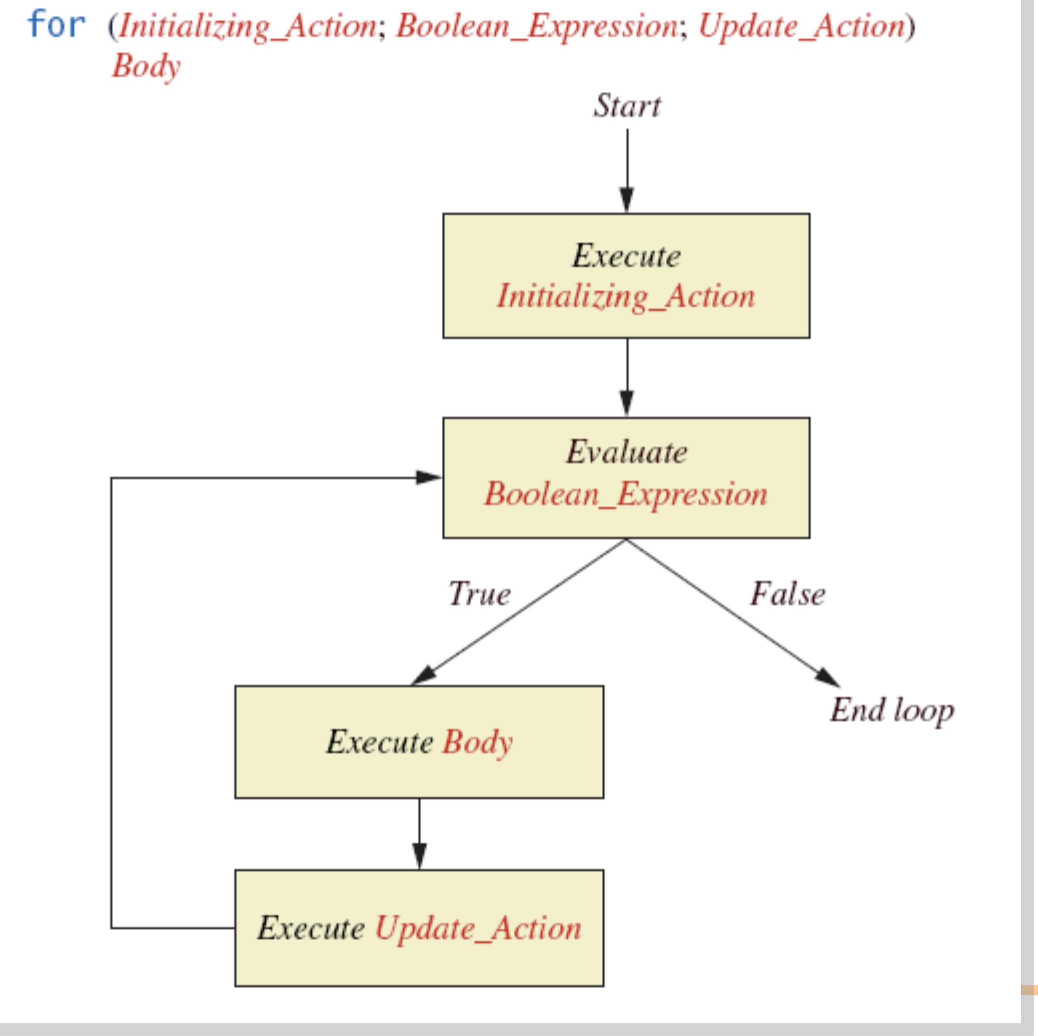
The `for` loop





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

The `for` loop



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

The `for` loop

- Possible to declare variables within a `for` statement

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

- Note that `n` is local to the loop

The `for` loop

- 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 `&&`, `||`, and `!`

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` statements
- Loop Bugs
- Tracing Variables

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.

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
```

Controlling Number of Loop Iterations

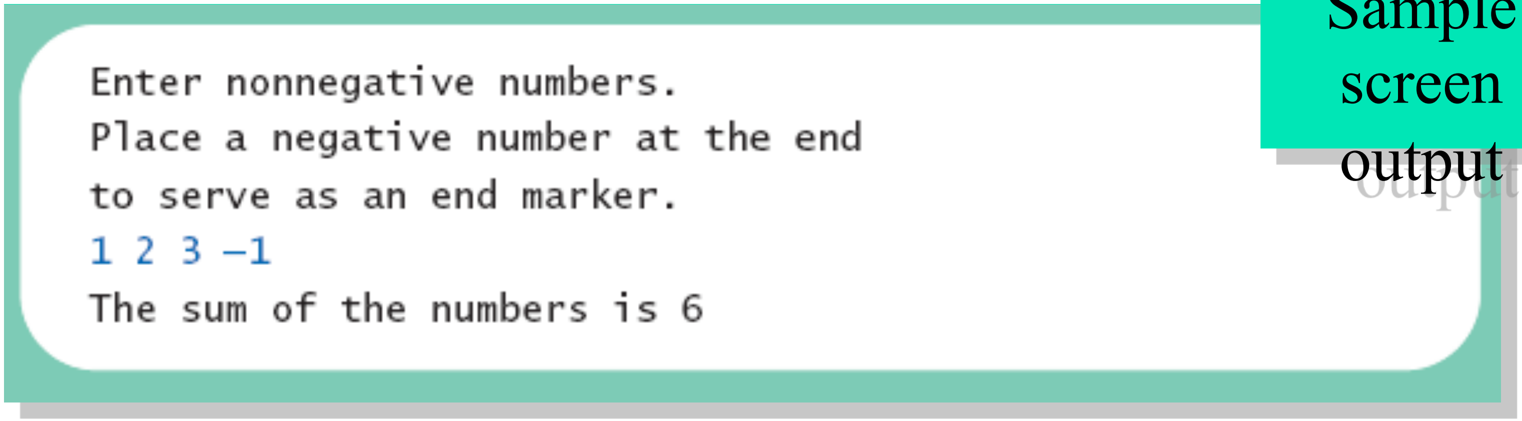
- Example - reading a list of scores followed by a sentinel value

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

Controlling Number of Loop Iterations

- Using a boolean variable to end the loop
- View sample program, listing 4.6

```
class BooleanDemo
```



```
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
```

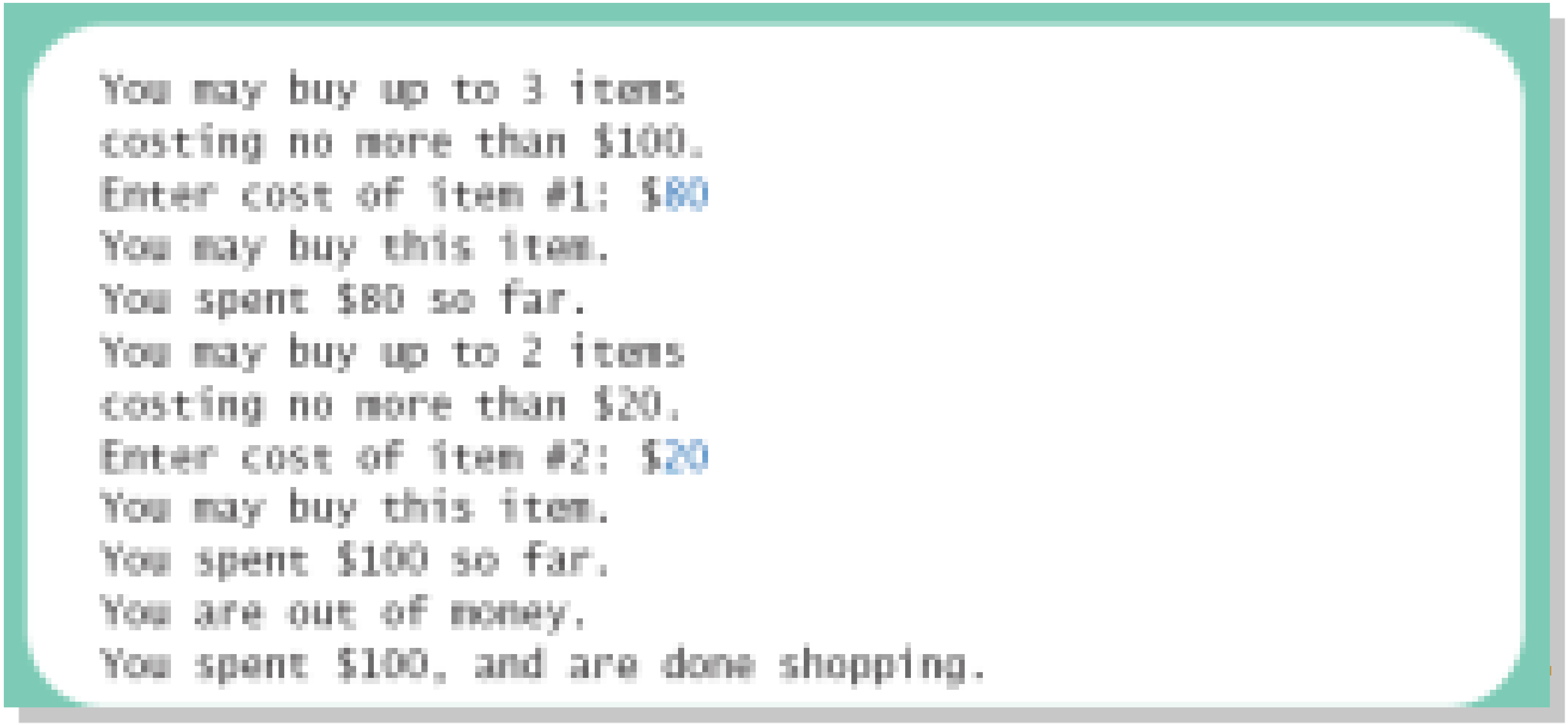
Sample screen output

Programming Example

- Spending Spree
  - You have $100 to spend in a store
  - Maximum 3 items
  - Computer tracks spending and item count
  - When item chosen, computer tells you whether or not you can buy it
- Client wants adaptable program
  - Able to change amount and maximum number of items

Programming Example

- Sample program
  `class SpendingSpree`

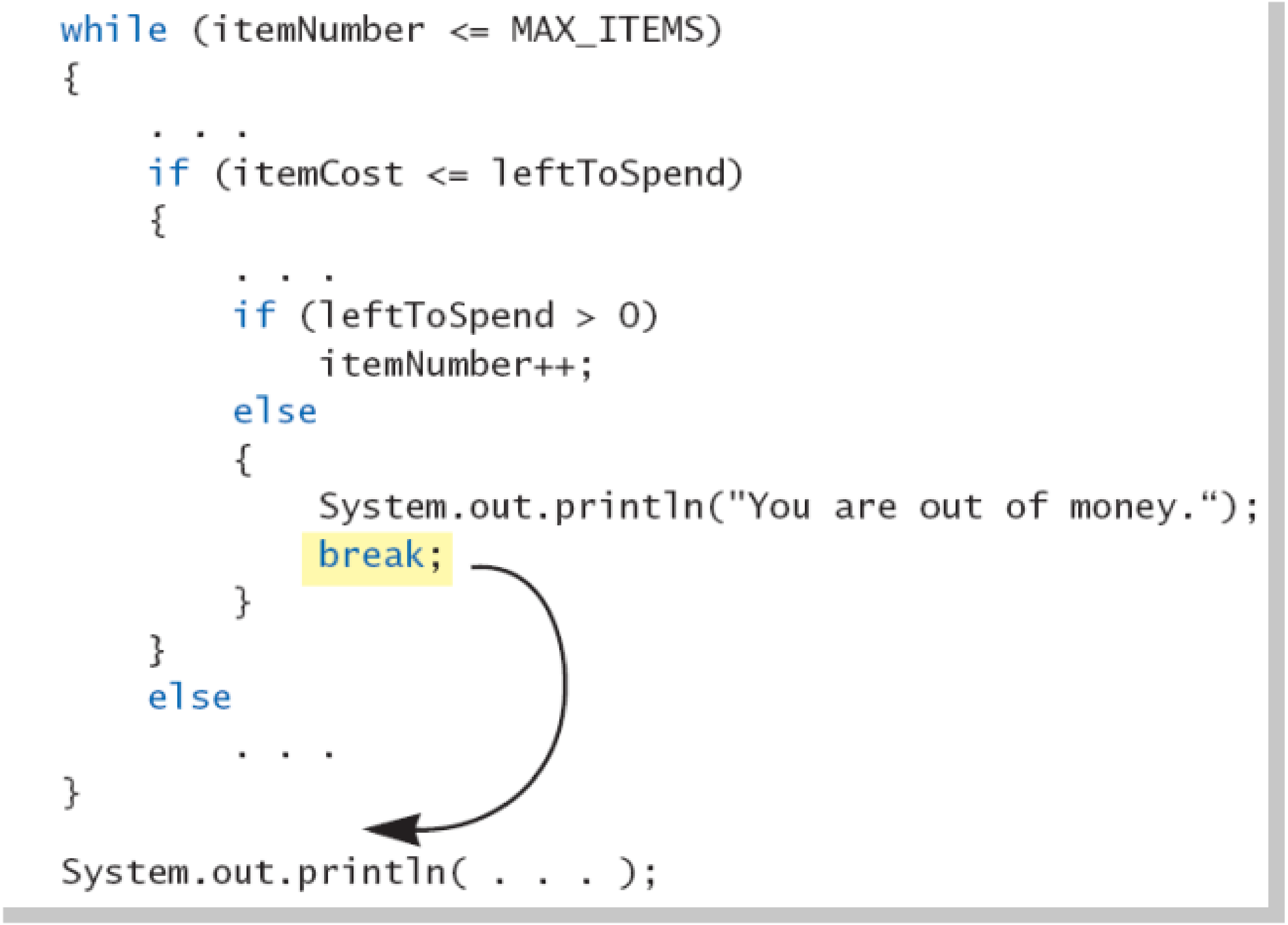


```
You may buy up to 3 items
costing no more than $100.
Enter cost of item #1: $80
You may buy this item.
You spent $80 so far.
You may buy up to 2 items
costing no more than $20.
Enter cost of item #2: $20
You may buy this item.
You spent $100 so far.
You are out of money.
You spent $100, and are done shopping.
```

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 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( . . . );
```

Assertion Checks

- Assertion : something that says something about the state of the program
  - Can be true or false
  - Should be true when no mistakes in running program

Assertion Checks

- Example found in comments

```
// n == 1
while (n < limit)
{
    n = 2 * n;
}
// n >= limit
// n is the smallest power of 2 >= limit
```

- Syntax for assertion check

```
Assert Boolean_Expression;
```

Assertion Checks

- Equivalent example using **`assert`**

```
assert n == 1;
while (n < limit)
{
    n = 2 * n;
}
assert n >= limit;
// n is the smallest power of 2 >= limit.
```

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.

Summary

- A loop is a programming construct that repeats an action
- Java has the `while`, (the `do-while)`, and the `for` statements
- The `while` repeat the loop while a condition is true
- The logic of a `for` statement is identical to the while