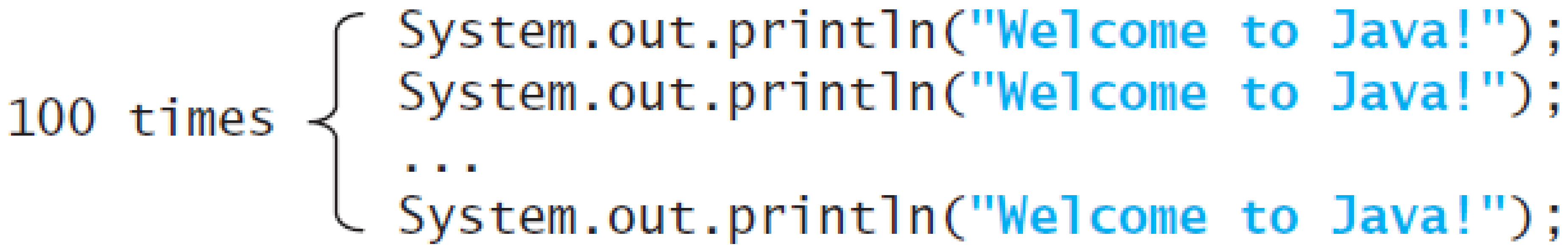
Loops

Introduction

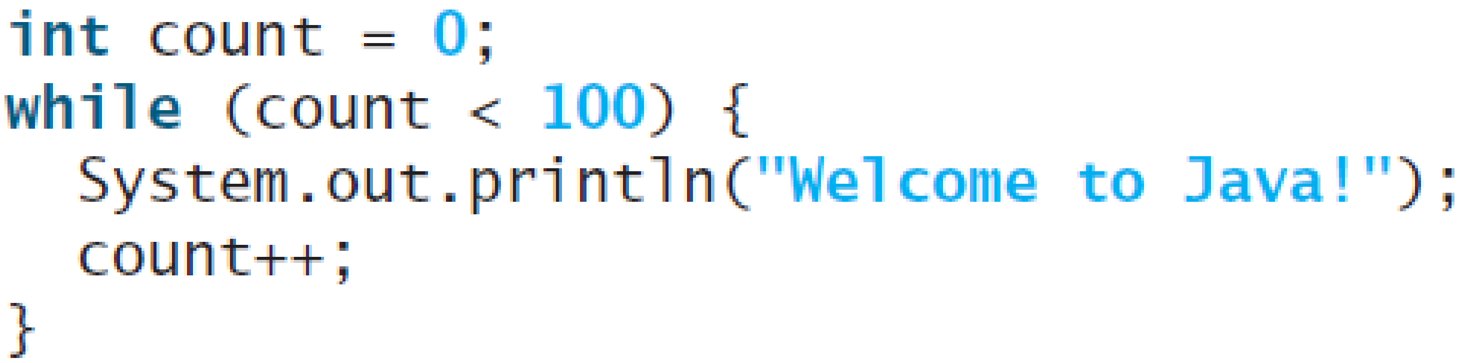
- A loop can be used to tell a program to execute statements repeatedly.
- It would be tedious to have to write the following statement a hundred times:



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

Introduction

- Java provides a powerful construct called a loop that controls how many times an operation or a sequence of operations is performed in succession.



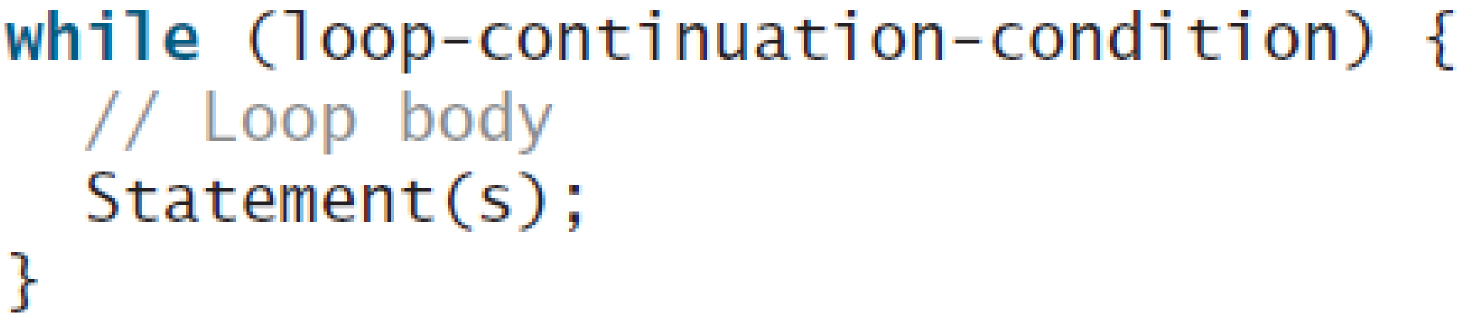
```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Introduction

- Loops are constructs that control repeated executions of a block of statements.
- The concept of looping is fundamental to programming.
- Java provides three types of loop statements: while loops, do-while loops, and for loops.

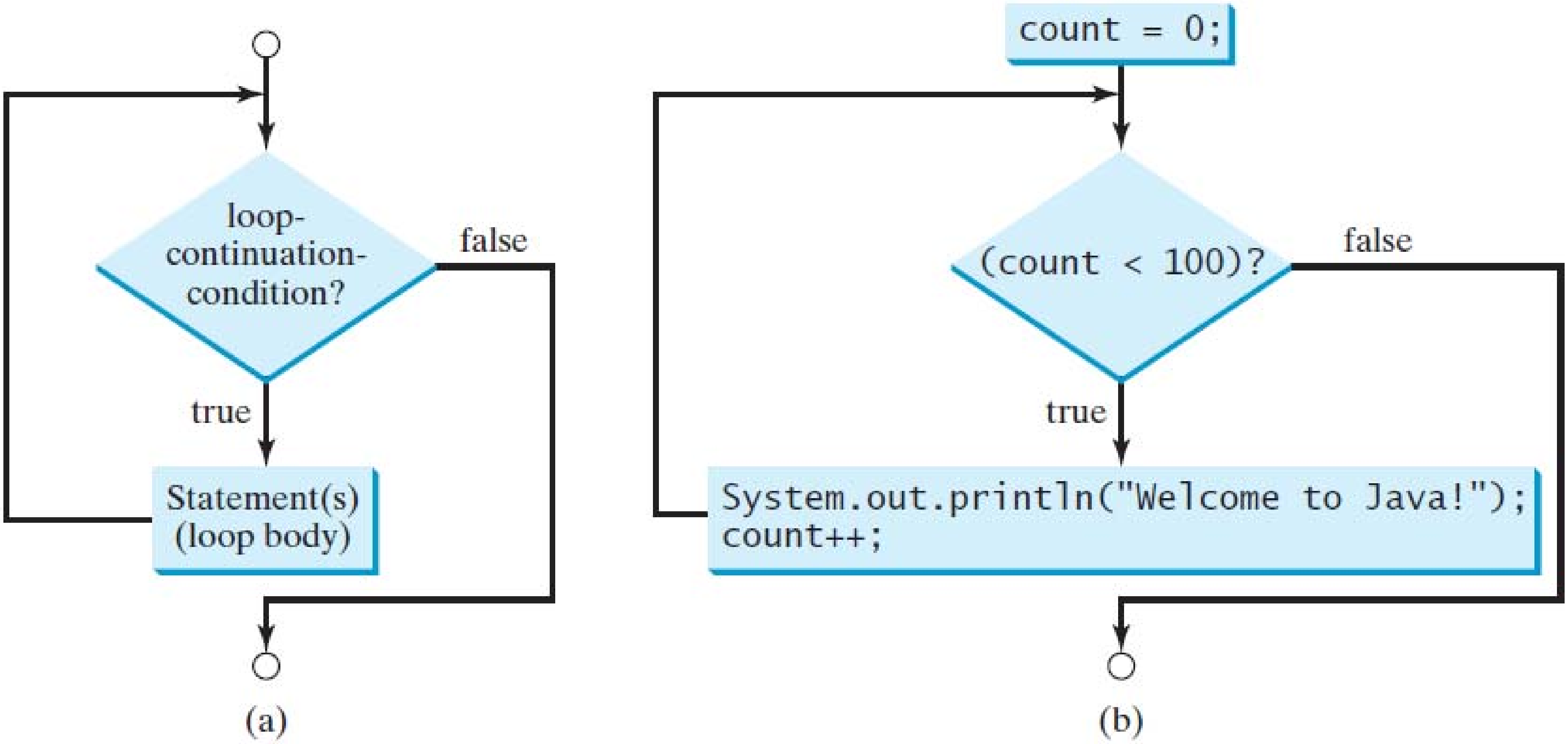
The while Loop

- A while loop executes statements repeatedly while the condition is true.
- The syntax for the while loop is:



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

The while Loop - Flowchart



loop-continuation-condition?

false

true

Statement(s)
(loop body)

(a)

count = 0;

(count < 100)?

false

true

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

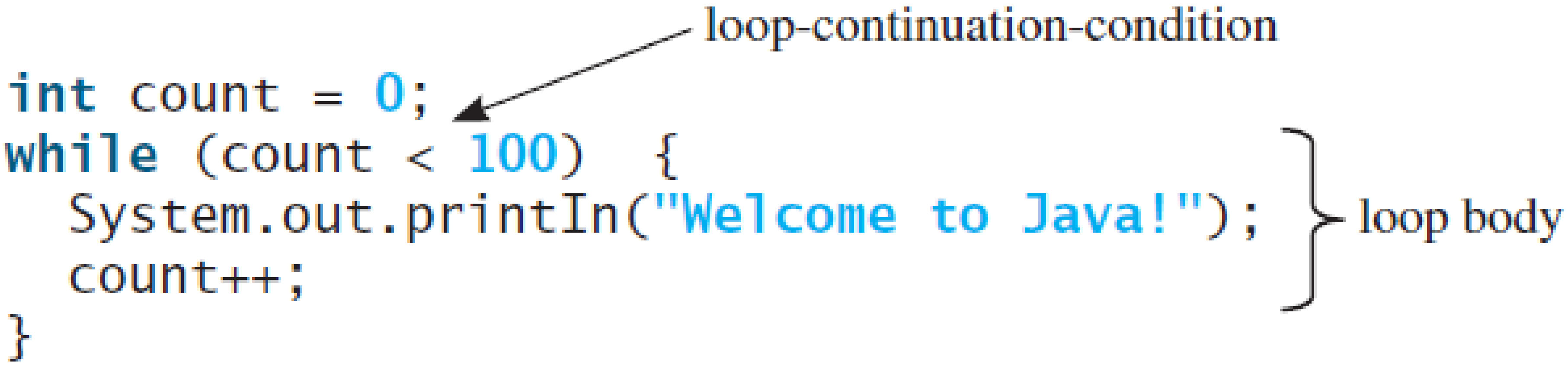
(b)

The while Loop - Mechanism

- The part of the loop that contains the statements to be repeated is called the loop body.
- A one-time execution of a loop body is referred to as an iteration (or repetition) of the loop.

The while Loop - Mechanism

- Each loop contains a loop-continuation-condition, a Boolean expression that controls the execution of the body.

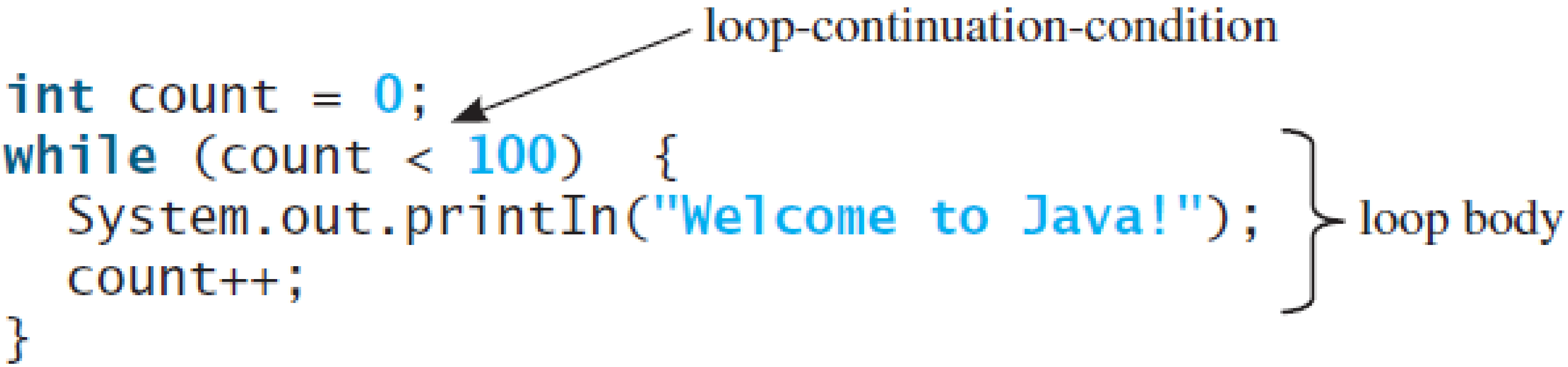


```
int count = 0;
while (count < 100)  {
   System.out.println("Welcome to Java!");
   count++;
}
```

The while Loop - Mechanism

- The loop-continuation-condition is evaluated each time to determine if the loop body is executed.
  - If its evaluation is true, the loop body is executed;
  - if its evaluation is false, the entire loop terminates and the program control turns to the statement that follows the while loop.

Counter-Controlled Loop



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

- In this example, you know exactly how many times the loop body needs to be executed because the control variable count is used to count the number of executions.
- This type of loop is known as a counter-controlled loop.

Exercise

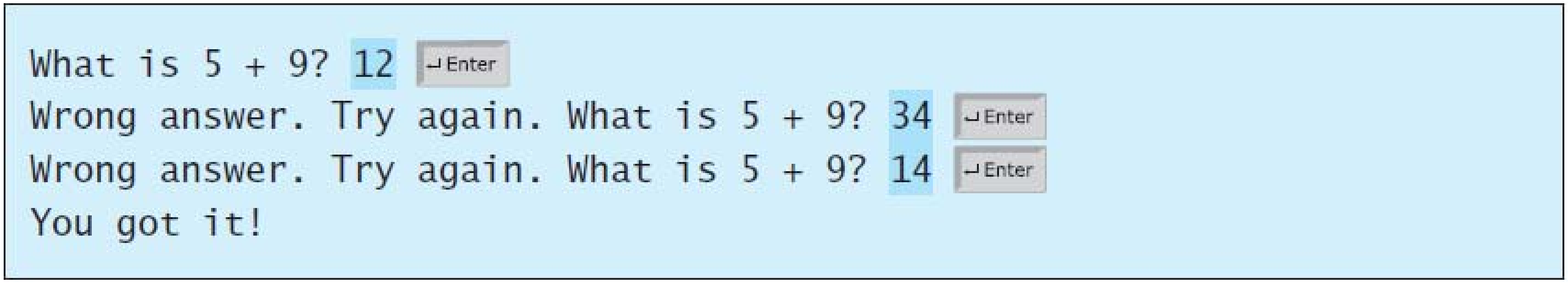
- Give a program that prompts the user to enter an answer for a question on addition of two single digits.
- Using a loop, you can write the program to let the user repeatedly enter a new answer until it is correct.

Exercise

```
import java.util.Scanner;

public class RepeatAdditionQuiz {
 public static void main(String[] args) {
  int number1 = (int)(Math.random() * 10);
  int number2 = (int)(Math.random() * 10);

  // Create a Scanner
  Scanner input = new Scanner(System.in);

  System.out.print("What is " + number1 + " + " + number2 + "? ");
  int answer = input.nextInt();
```

Exercise

```
  while (number1 + number2 != answer) {
   System.out.print("Wrong answer. Try again. What is "
   + number1 + " + " + number2 + "? ");
   answer = input.nextInt();
  }

  System.out.println("You got it!");
  }
}
```



```
What is 5 + 9? 12
Wrong answer. Try again. What is 5 + 9? 34
Wrong answer. Try again. What is 5 + 9? 14
You got it!
```

Sentinel-Controlled Loop

- Another common technique for controlling a loop is to designate a special value when reading and processing a set of values.
- This special input value, known as a sentinel value, signifies the end of the input.
- A loop that uses a sentinel value to control its execution is called a sentinel-controlled loop.

Example

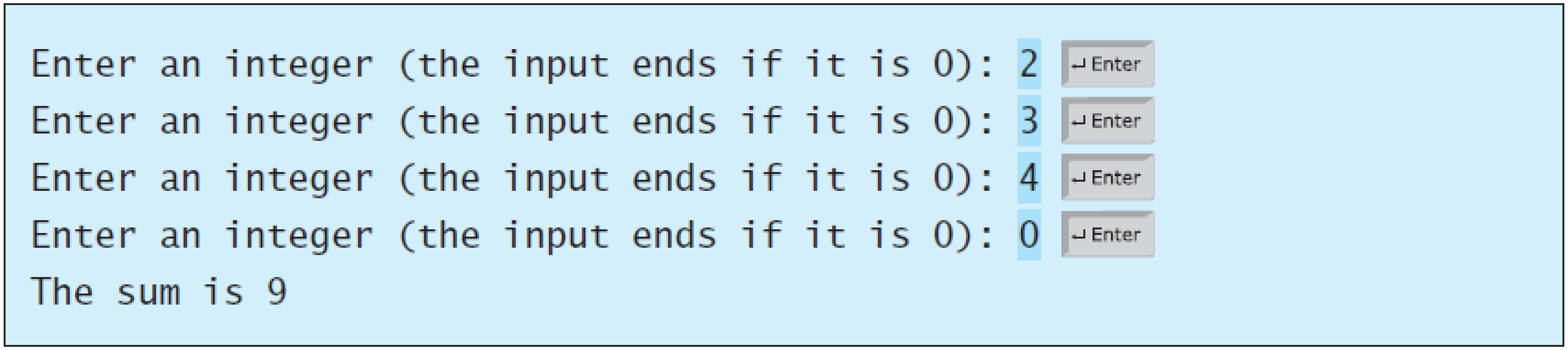
- Writes a program that reads and calculates the sum of an unspecified number of integers.
- The input 0 signifies the end of the input.

Example

```
import java.util.Scanner;

public class SentinelValue {
  public static void main(String[] args) {
  // Create a Scanner
  Scanner input = new Scanner(System.in);

  // Read an initial data
  System.out.print(
  "Enter an integer (the input ends if it is 0): ");
  int data = input.nextInt();
```

Example

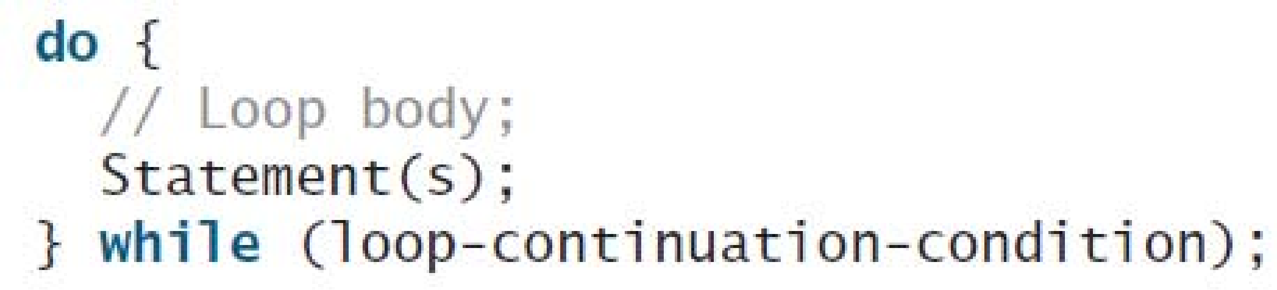
```
  // Keep reading data until the input is 0
  int sum = 0;
  while (data != 0) {
    sum += data;

    // Read the next data
    System.out.print(
    "Enter an integer (the input ends if it is 0): ");
    data = input.nextInt();
  }
  System.out.println("The sum is " + sum);
 }
}
```



```
Enter an integer (the input ends if it is 0): 2 ↵Enter
Enter an integer (the input ends if it is 0): 3 ↵Enter
Enter an integer (the input ends if it is 0): 4 ↵Enter
Enter an integer (the input ends if it is 0): 0 ↵Enter
The sum is 9
```

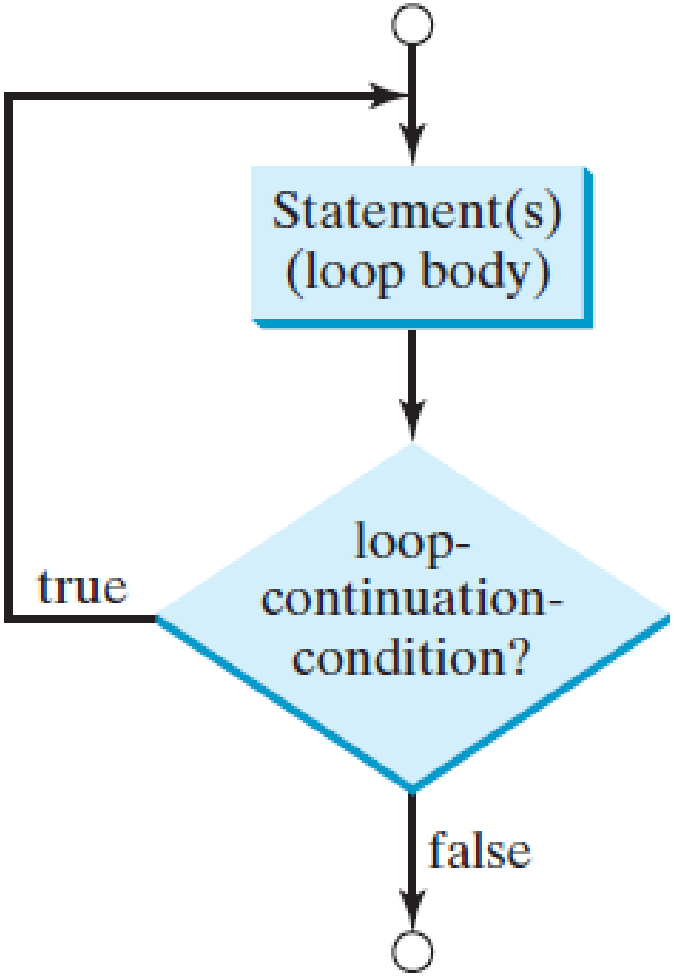
The do-while Loop

- A do-while loop is the same as a while loop except that it executes the loop body first and then checks the loop continuation condition.
- Syntax:



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

The do-while Loop - Flowchart



While-loop vs. Do-while loop

- You can write a loop using either the while loop or the do-while loop.
- Sometimes one is a more convenient choice than the other.
- The key point is to check the condition first (use while loop), or to do something and then check the condition (use do-while loop).

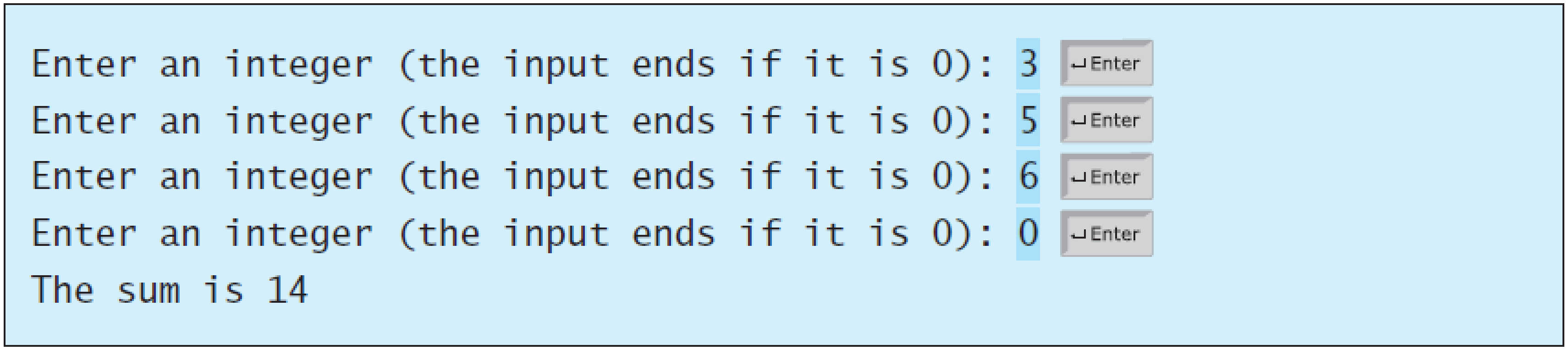
Example

```
import java.util.Scanner;

public class TestDoWhile {
  public static void main(String[] args) {
    int data;
    int sum = 0;

    // Create a Scanner
    Scanner input = new Scanner(System.in);
```

Example

```
    // Keep reading data until the input is 0
    do {
        // Read the next data
        System.out.print(
        "Enter an integer (the input ends if it is 0): ");
        data = input.nextInt();

        sum += data;
    } while (data != 0);

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

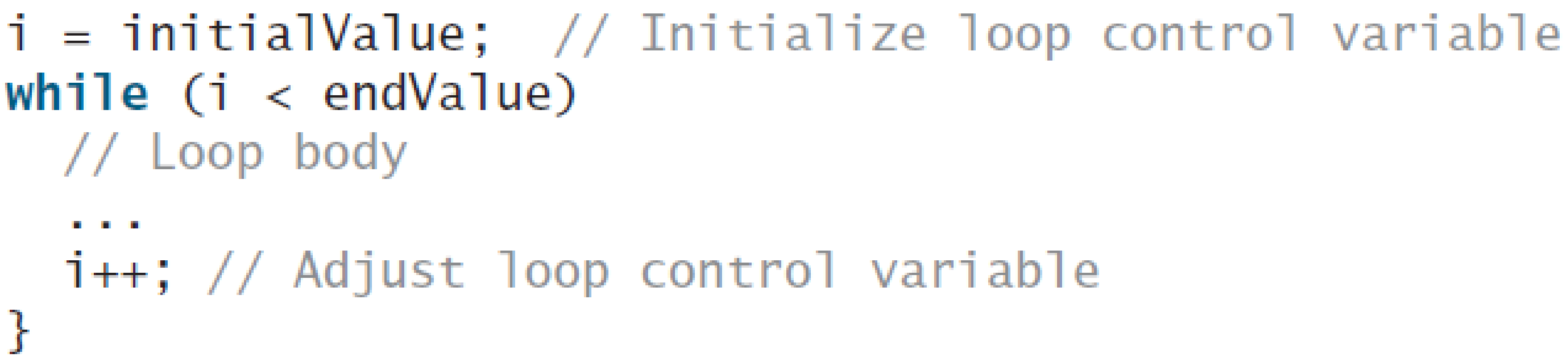
Example



```
Enter an integer (the input ends if it is 0): 3 ↵Enter
Enter an integer (the input ends if it is 0): 5 ↵Enter
Enter an integer (the input ends if it is 0): 6 ↵Enter
Enter an integer (the input ends if it is 0): 0 ↵Enter
The sum is 14
```

The for Loop

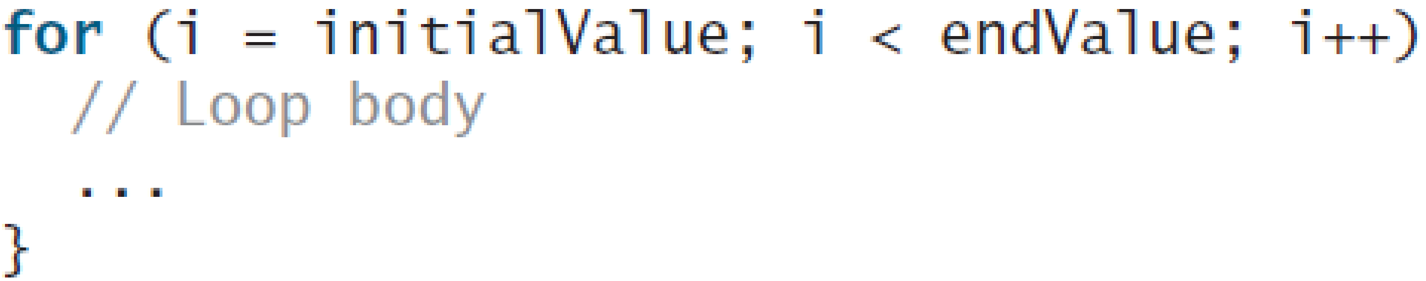
- A for loop has a concise syntax for writing loops.
- Often you write a loop in the following common form:



```
i = initialValue;  // Initialize loop control variable
while (i < endValue)
  // Loop body
  ...
  i++; // Adjust loop control variable
}
```

The for Loop

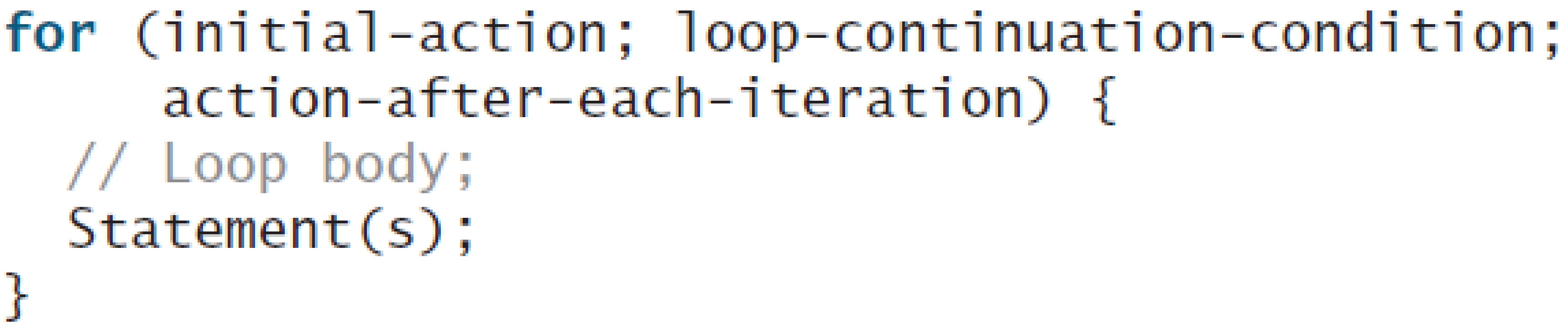
- A for loop can be used to simplify the preceding loop as:



```
for (i = initialValue; i < endValue; i++)
  // Loop body
  ...
}
```

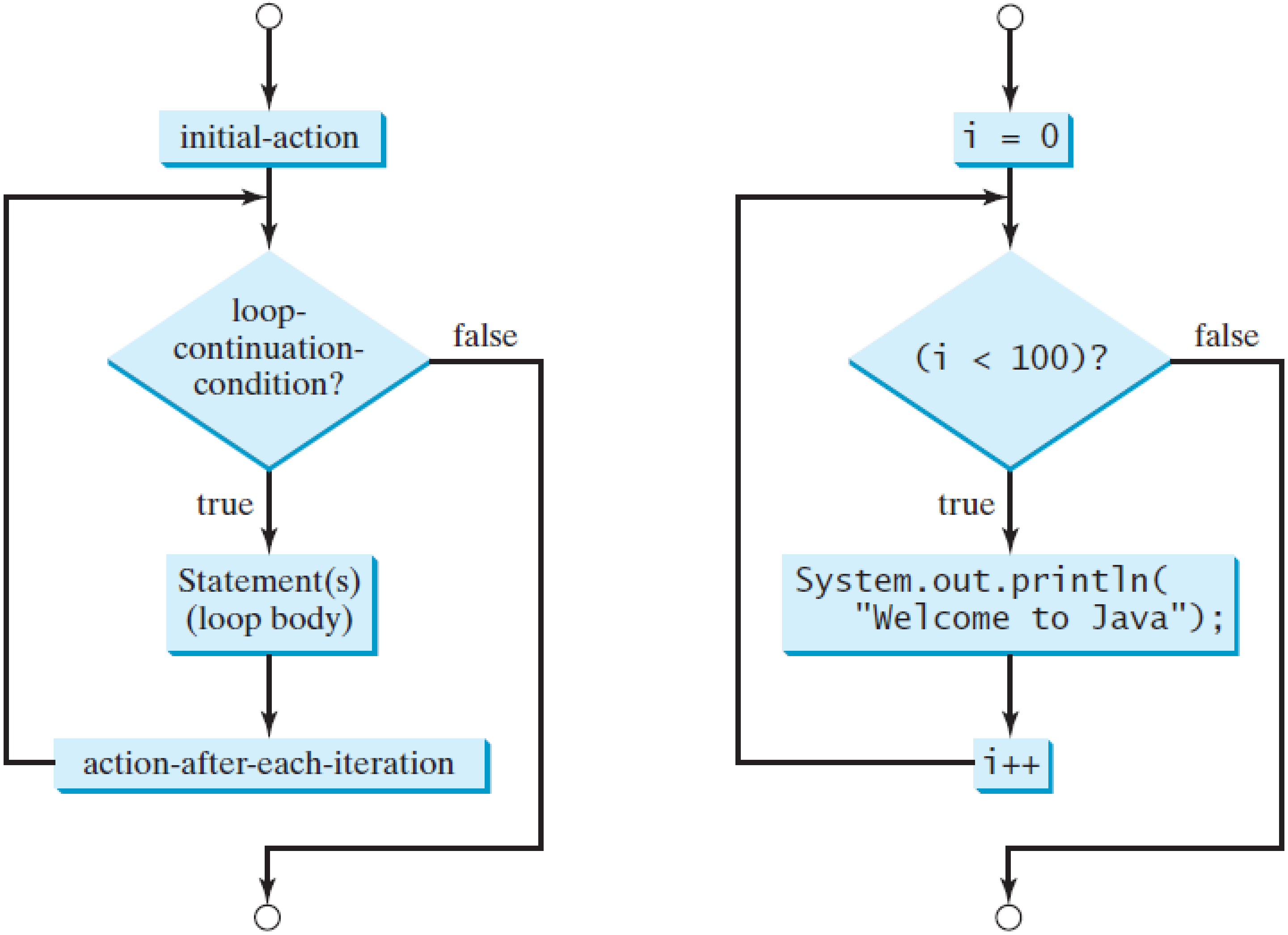
The for Loop

- In general, the syntax of a for loop is:



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

The for Loop - Flowchart


initial-action
loop-continuation-condition?
false
true
Statement(s) (loop body)
action-after-each-iteration
i = 0
(i < 100)?
false
true
System.out.println("Welcome to Java");
i++

(a)

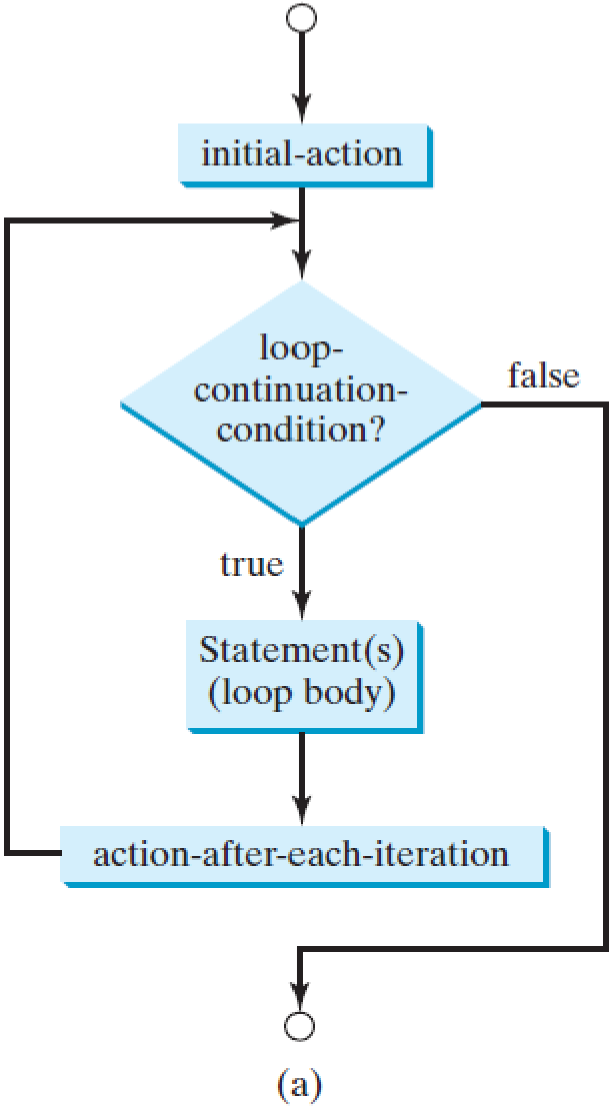
(b)

The for Loop - Mechanism

- A for loop generally uses a variable to control how many times the loop body is executed and when the loop terminates.
- This variable is referred to as a control variable.

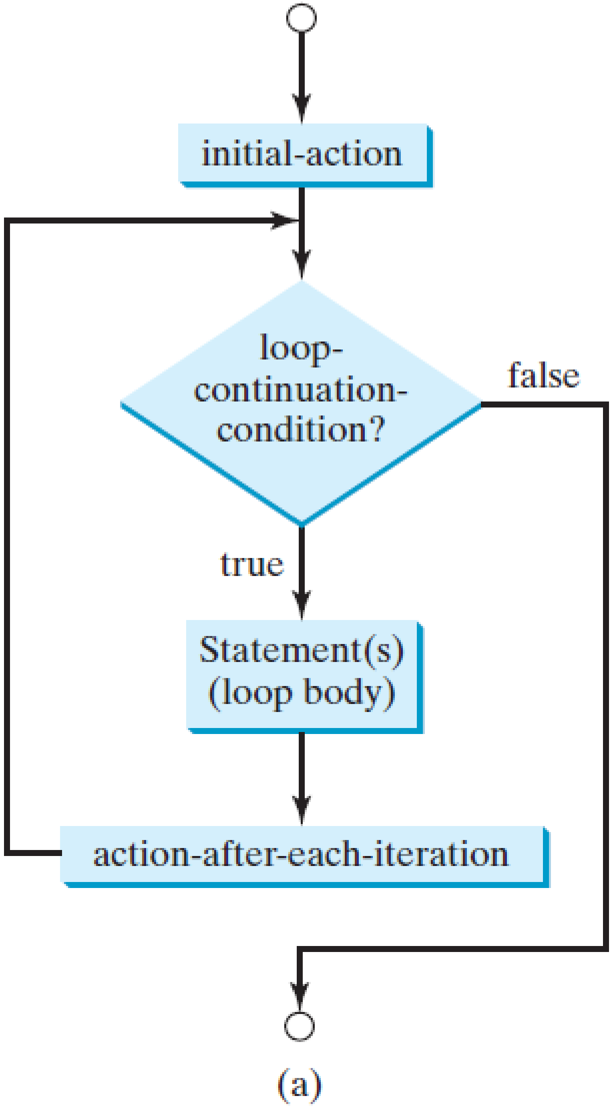
The for Loop - Mechanism

- The initial-action often initializes a control variable.



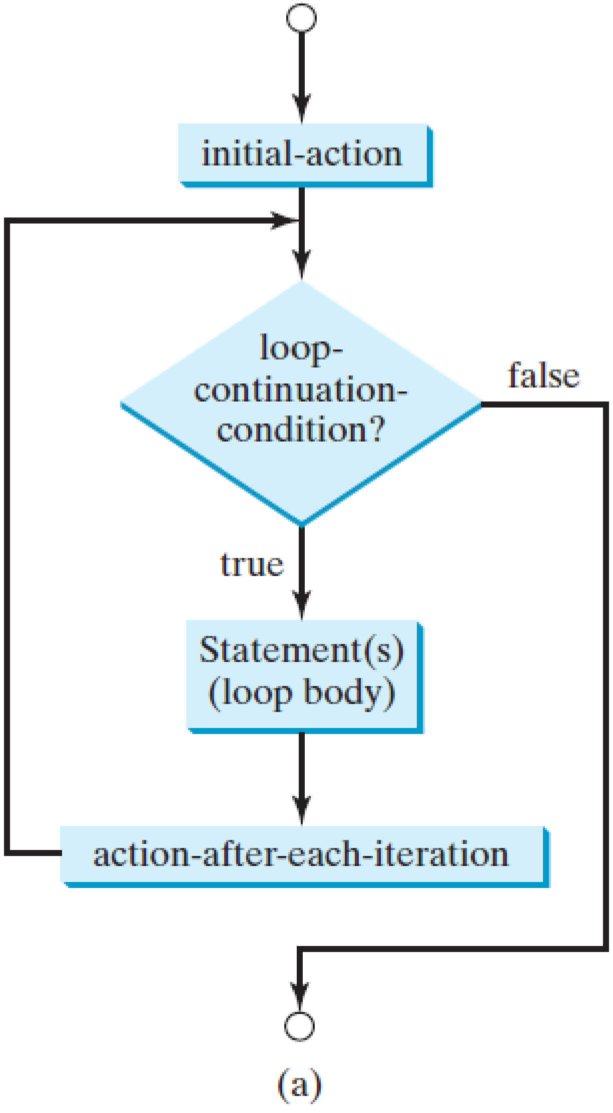
The for Loop - Mechanism

- The action-after-each-iteration usually increments or decrements the control variable.



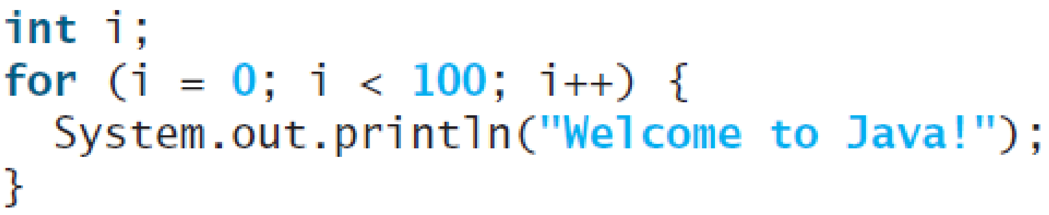
The for Loop - Mechanism

- The loop-continuation-condition tests whether the control variable has reached a termination value.

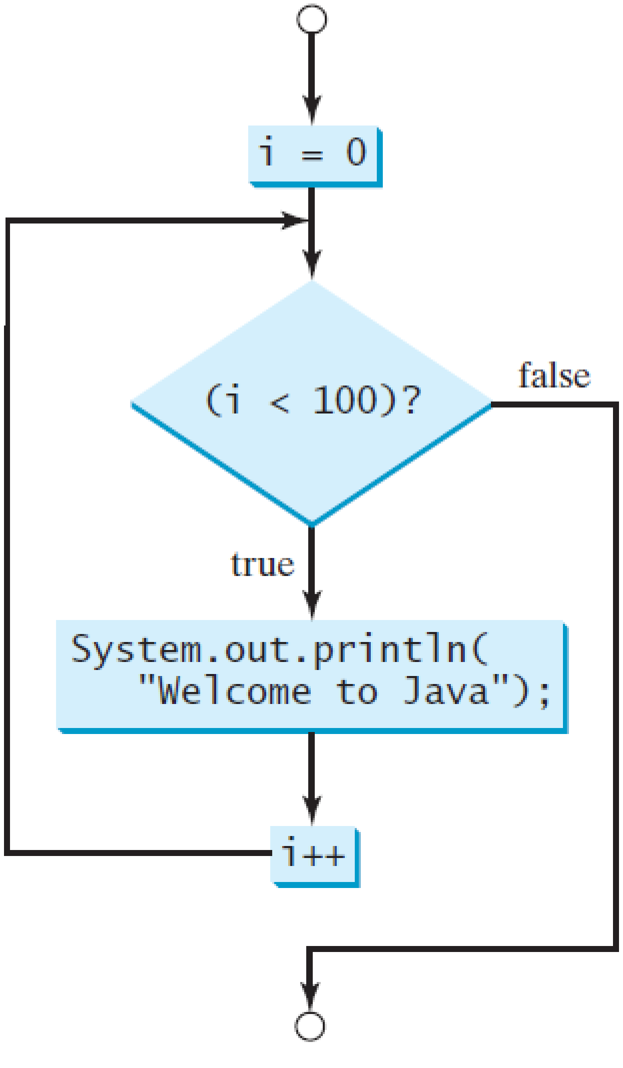


(a)

The for Loop - Mechanism



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

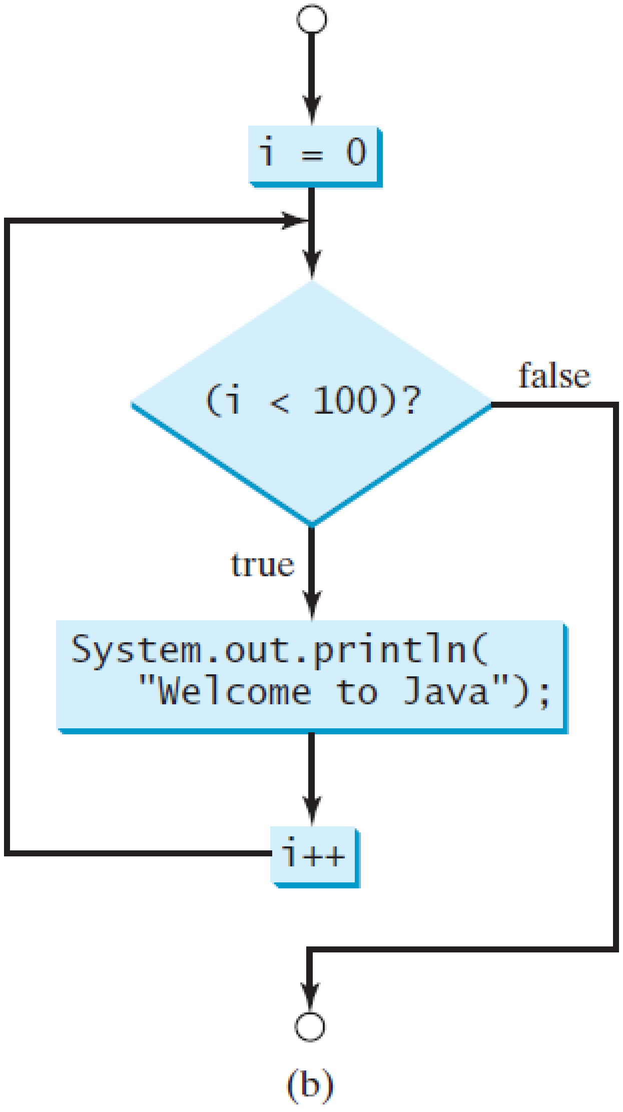


(b)

- The initial-action, i = 0, initializes the control variable, i.

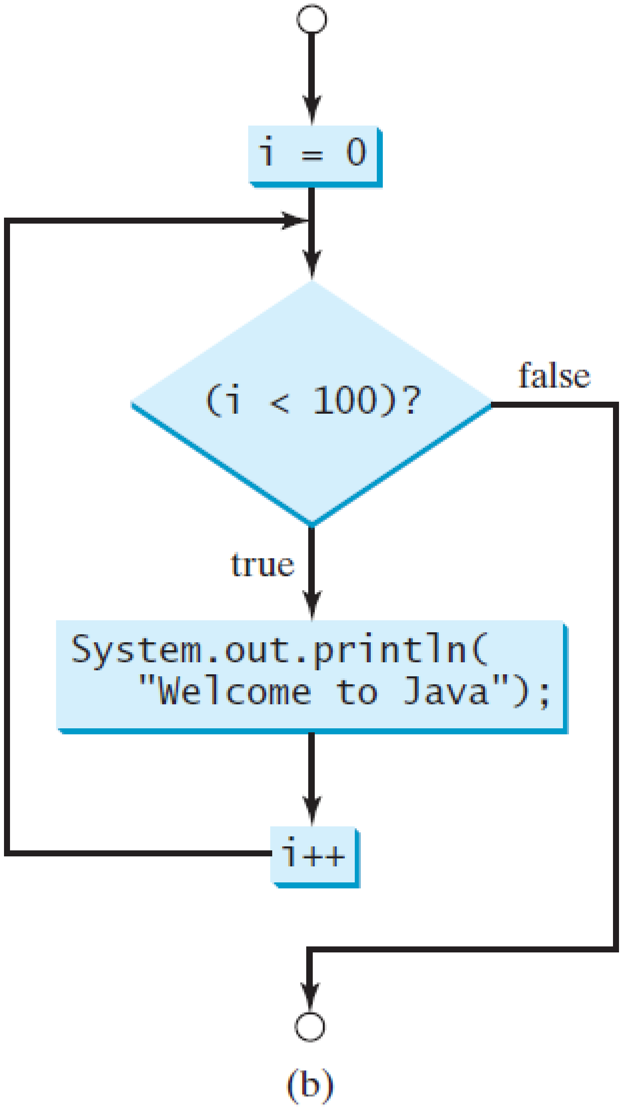
The for Loop - Mechanism

- The loop-continuation-condition, i < 100, is a Boolean expression.
- The expression is evaluated right after the initialization and at the beginning of each iteration.



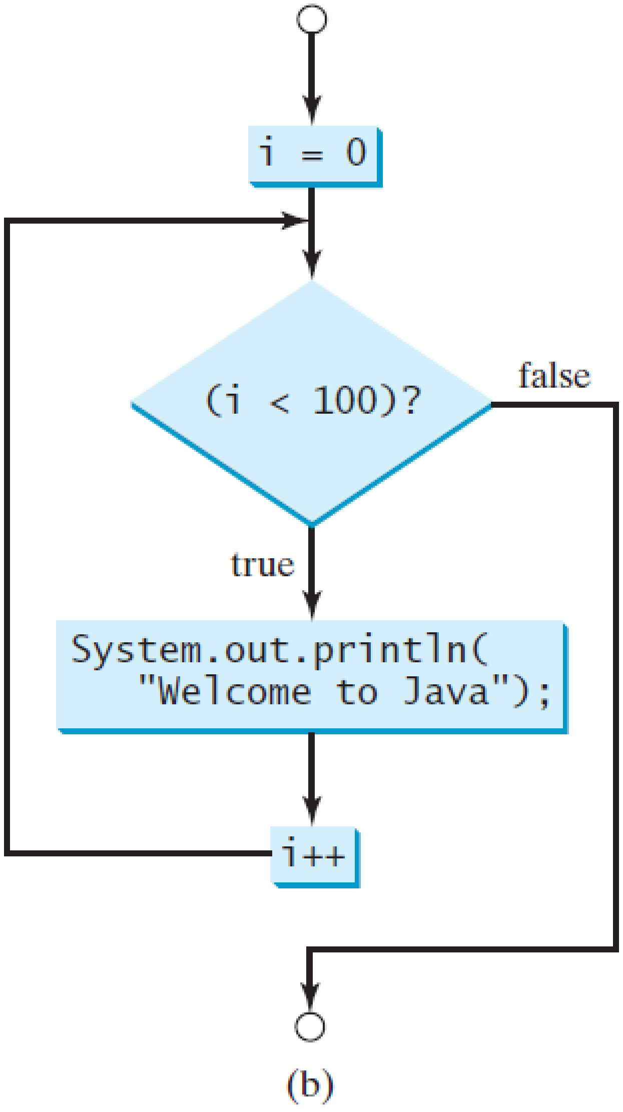
The for Loop - Mechanism

- If this condition is true, the loop body is executed.
- If it is false, the loop terminates and the program control turns to the line following the loop.



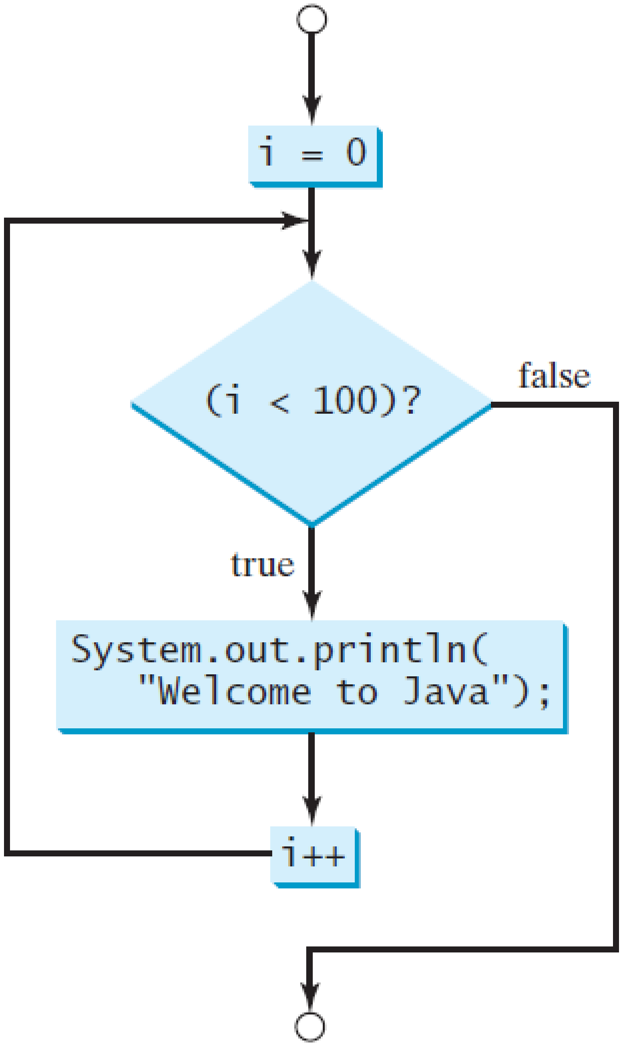
The for Loop - Mechanism

- The action-after-each-iteration, i++, is a statement that adjusts the control variable.
- This statement is executed after each iteration and increments the control variable.
- Eventually, the value of the control variable should force the loop-continuation-condition to become false; otherwise, the loop is infinite.



The for Loop - Mechanism

- Eventually, the value of the control variable should force the loop-continuation-condition to become false; otherwise, the loop is infinite.



(b)

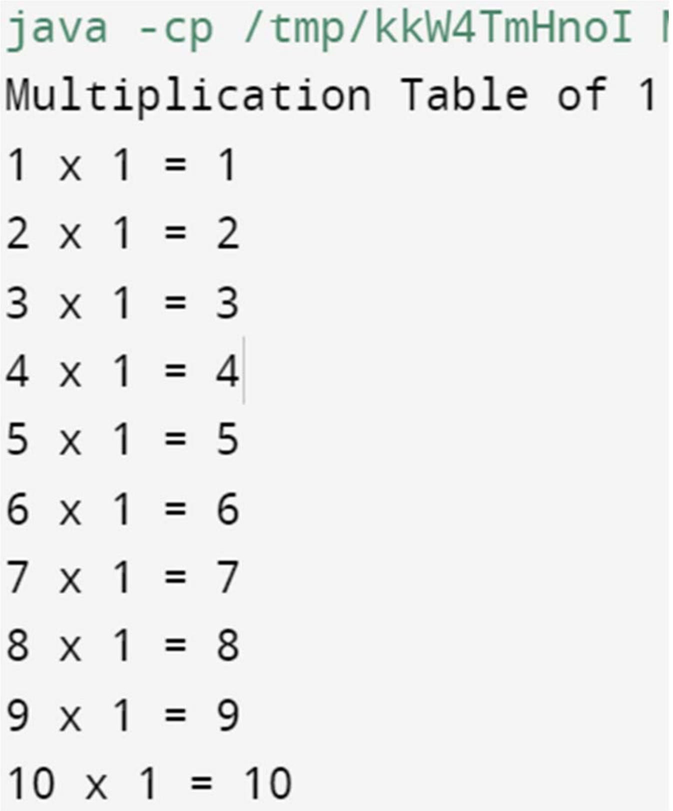
Nested Loops

- Nested loops consist of an outer loop and one or more inner loops.
- Each time the outer loop is repeated, the inner loops are reentered, and started anew.

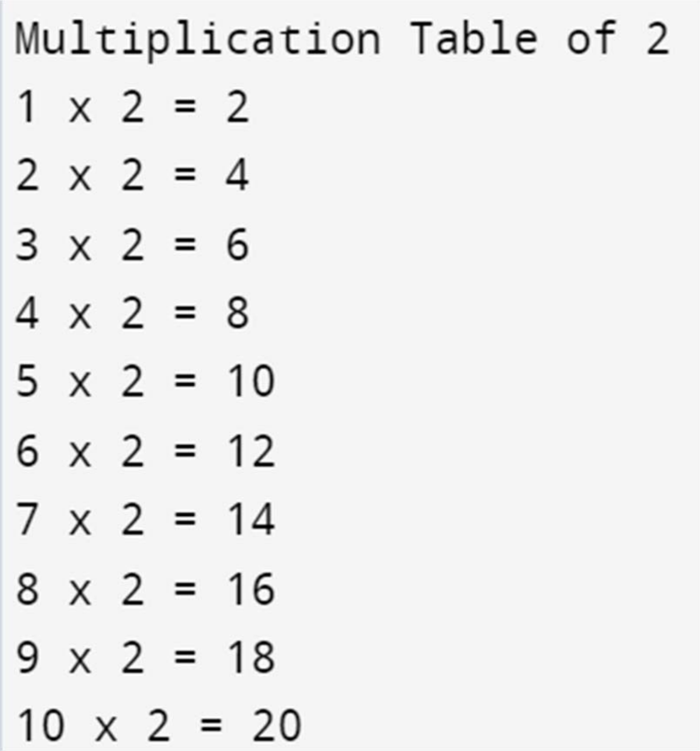
Example

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

        // Outer loop : fixing the base n
        for (int n = 1; n <= 10; n++) {
            System.out.println("Multiplication Table of " + n);

            // Inner loop : 10 lines of multiplication table
            for(int i = 1; i <= 10; i++) {
                System.out.println(i + " x " + n + " = " + (n*i));
            }
        }
    }
}
```

Example

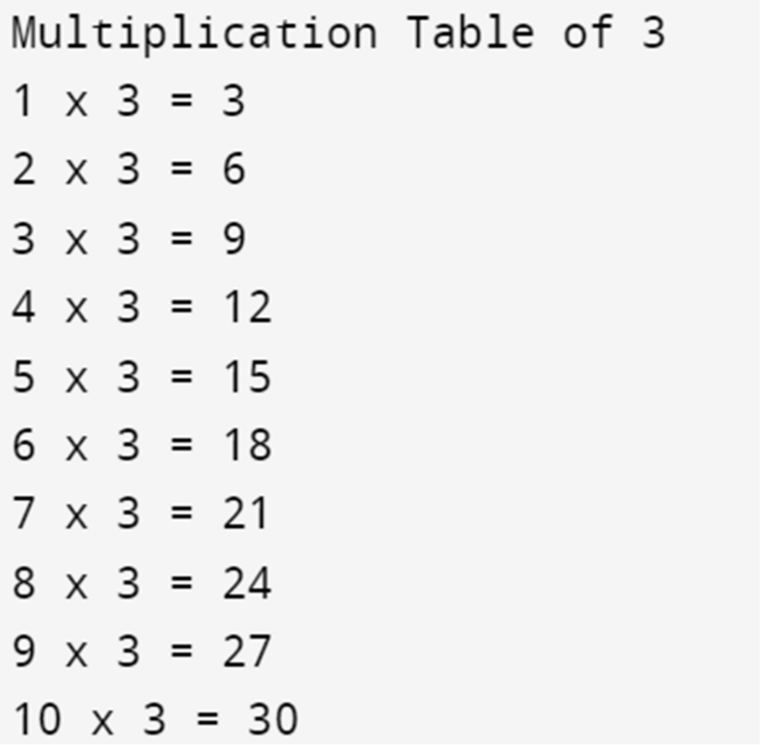


```
java -cp /tmp/kkW4TmHnoI
Multiplication Table of 1
1 x 1 = 1
2 x 1 = 2
3 x 1 = 3
4 x 1 = 4
5 x 1 = 5
6 x 1 = 6
7 x 1 = 7
8 x 1 = 8
9 x 1 = 9
10 x 1 = 10
```

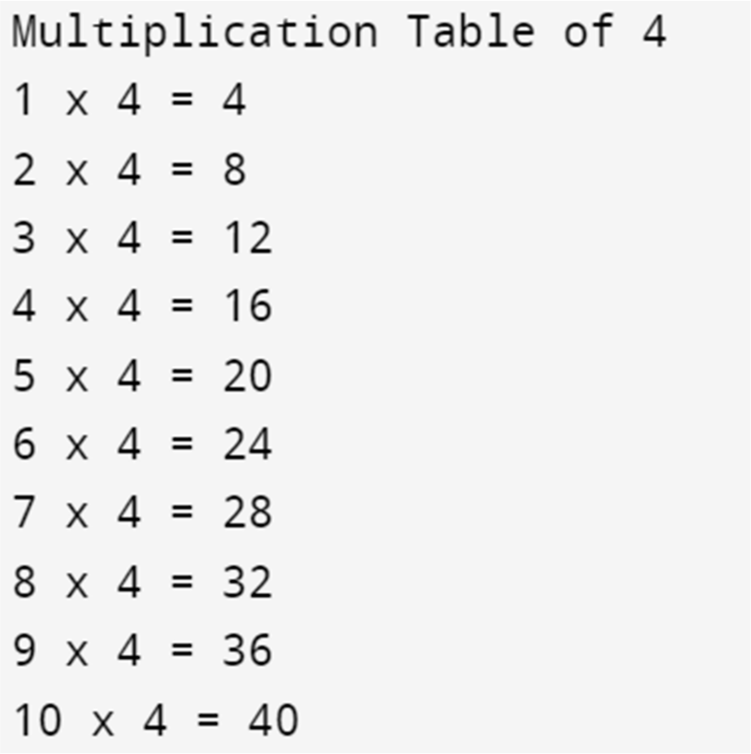


```
Multiplication Table of 2
1 x 2 = 2
2 x 2 = 4
3 x 2 = 6
4 x 2 = 8
5 x 2 = 10
6 x 2 = 12
7 x 2 = 14
8 x 2 = 16
9 x 2 = 18
10 x 2 = 20
```

Example

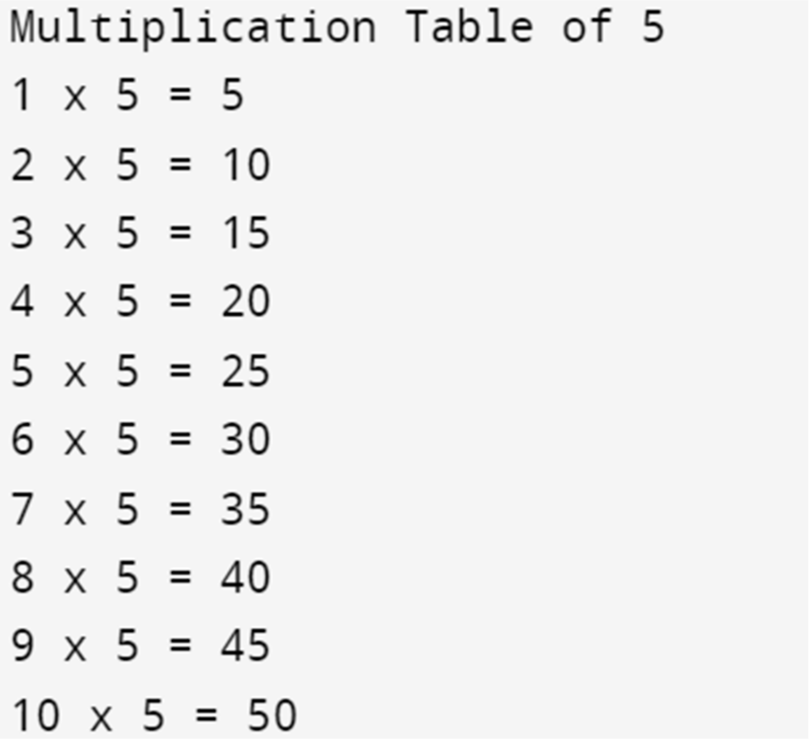


```
Multiplication Table of 3
1 x 3 = 3
2 x 3 = 6
3 x 3 = 9
4 x 3 = 12
5 x 3 = 15
6 x 3 = 18
7 x 3 = 21
8 x 3 = 24
9 x 3 = 27
10 x 3 = 30
```

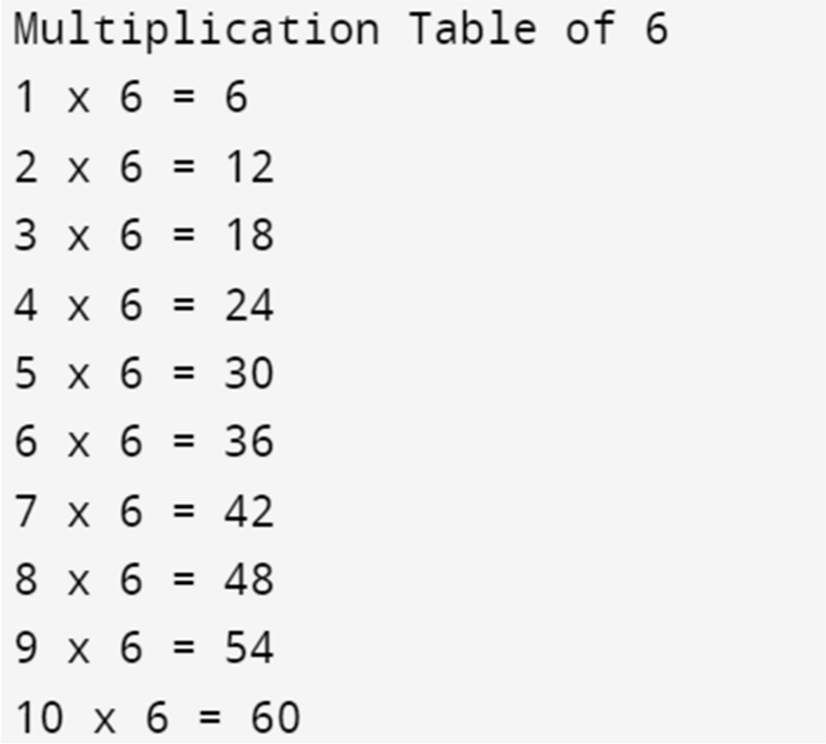


```
Multiplication Table of 4
1 x 4 = 4
2 x 4 = 8
3 x 4 = 12
4 x 4 = 16
5 x 4 = 20
6 x 4 = 24
7 x 4 = 28
8 x 4 = 32
9 x 4 = 36
10 x 4 = 40
```

Example

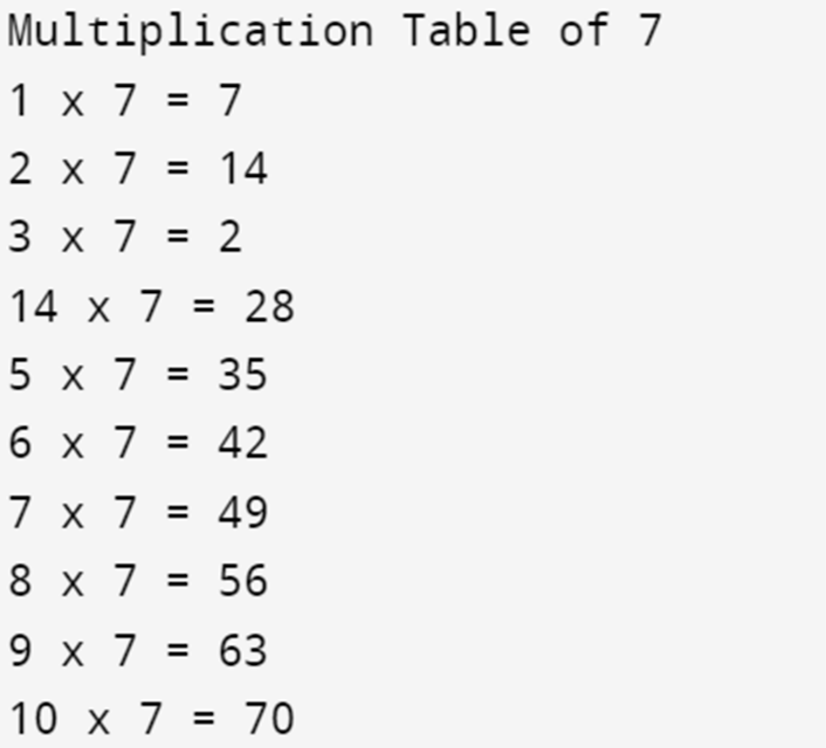


```
Multiplication Table of 5
1 x 5 = 5
2 x 5 = 10
3 x 5 = 15
4 x 5 = 20
5 x 5 = 25
6 x 5 = 30
7 x 5 = 35
8 x 5 = 40
9 x 5 = 45
10 x 5 = 50
```

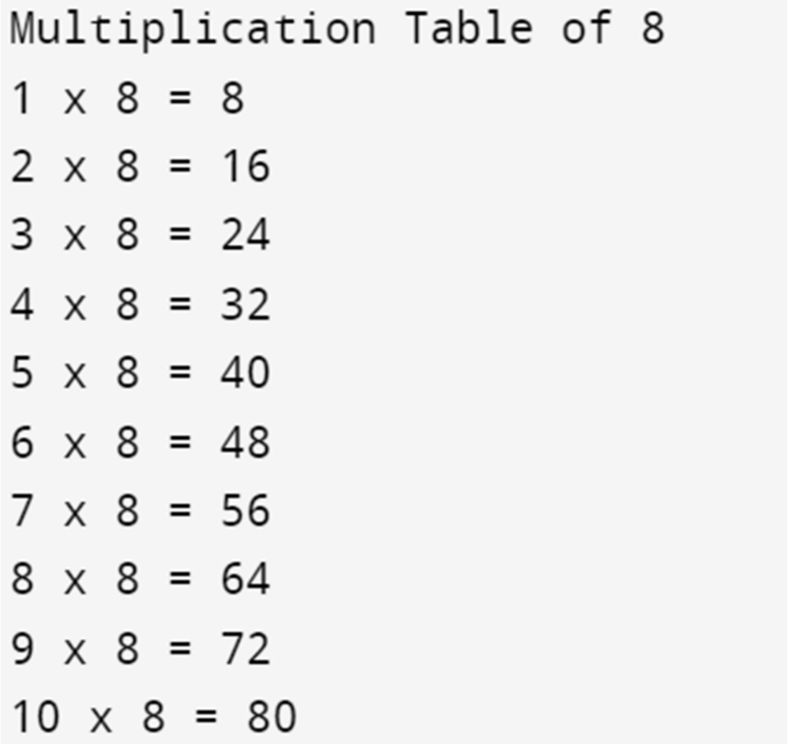


```
Multiplication Table of 6
1 x 6 = 6
2 x 6 = 12
3 x 6 = 18
4 x 6 = 24
5 x 6 = 30
6 x 6 = 36
7 x 6 = 42
8 x 6 = 48
9 x 6 = 54
10 x 6 = 60
```

Example

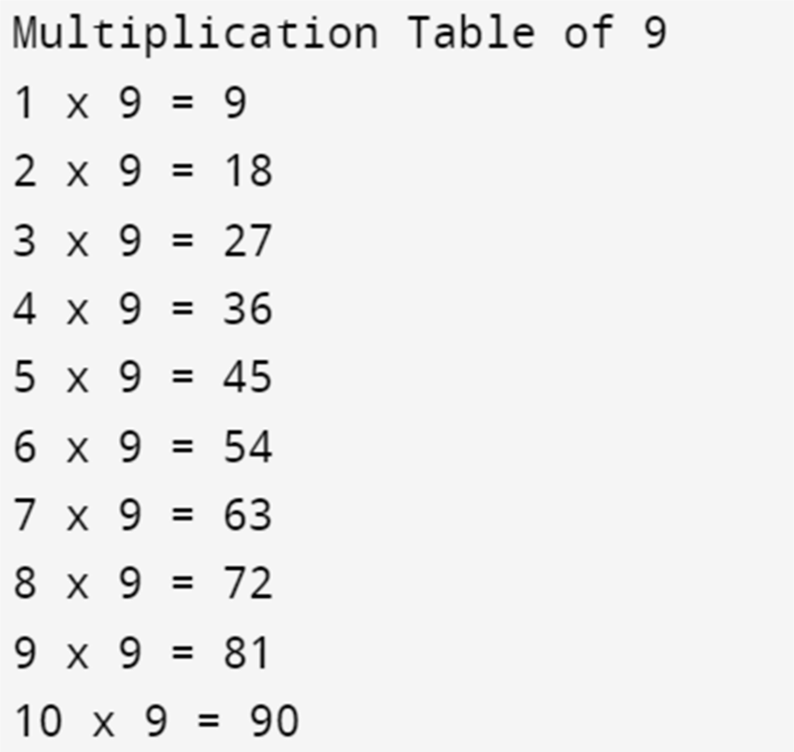


```
Multiplication Table of 7
1 x 7 = 7
2 x 7 = 14
3 x 7 = 2
14 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
```

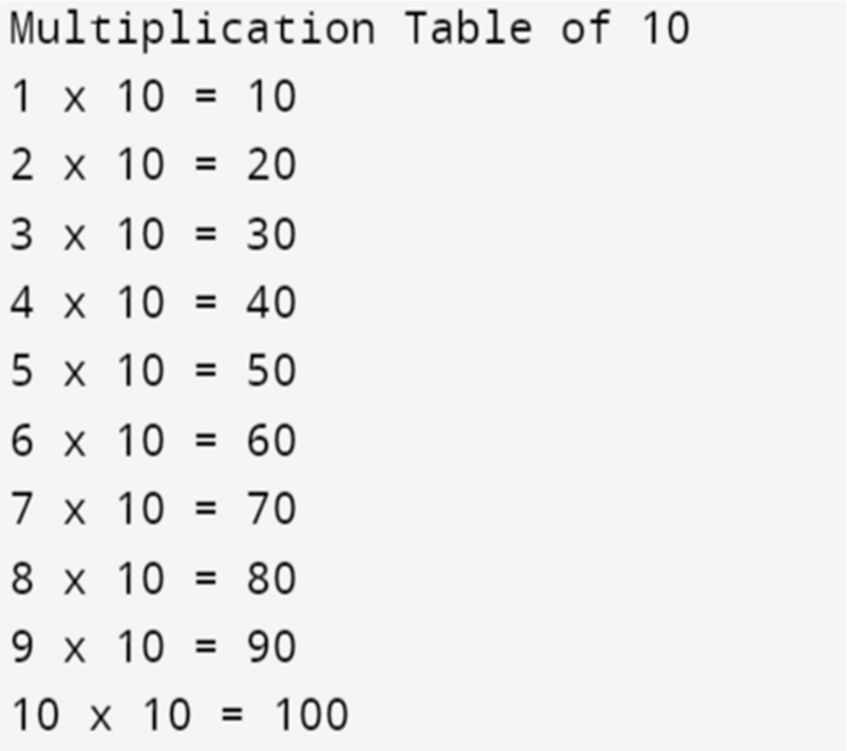


```
Multiplication Table of 8
1 x 8 = 8
2 x 8 = 16
3 x 8 = 24
4 x 8 = 32
5 x 8 = 40
6 x 8 = 48
7 x 8 = 56
8 x 8 = 64
9 x 8 = 72
10 x 8 = 80
```

Example



```
Multiplication Table of 9
1 x 9 = 9
2 x 9 = 18
3 x 9 = 27
4 x 9 = 36
5 x 9 = 45
6 x 9 = 54
7 x 9 = 63
8 x 9 = 72
9 x 9 = 81
10 x 9 = 90
```



```
Multiplication Table of 10
1 x 10 = 10
2 x 10 = 20
3 x 10 = 30
4 x 10 = 40
5 x 10 = 50
6 x 10 = 60
7 x 10 = 70
8 x 10 = 80
9 x 10 = 90
10 x 10 = 100
```