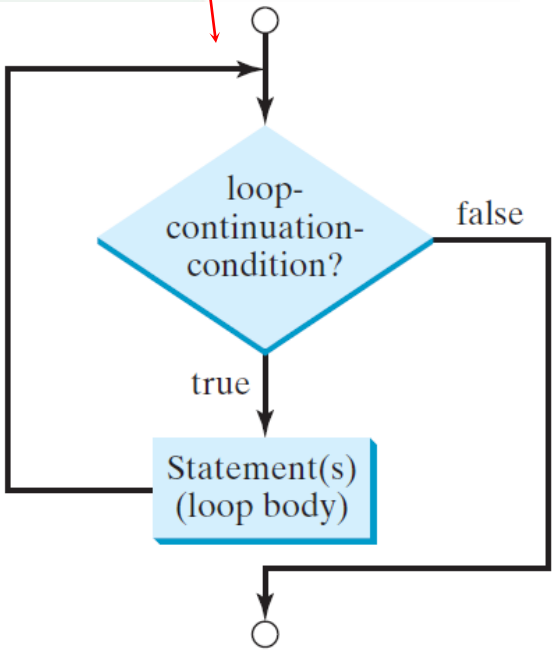
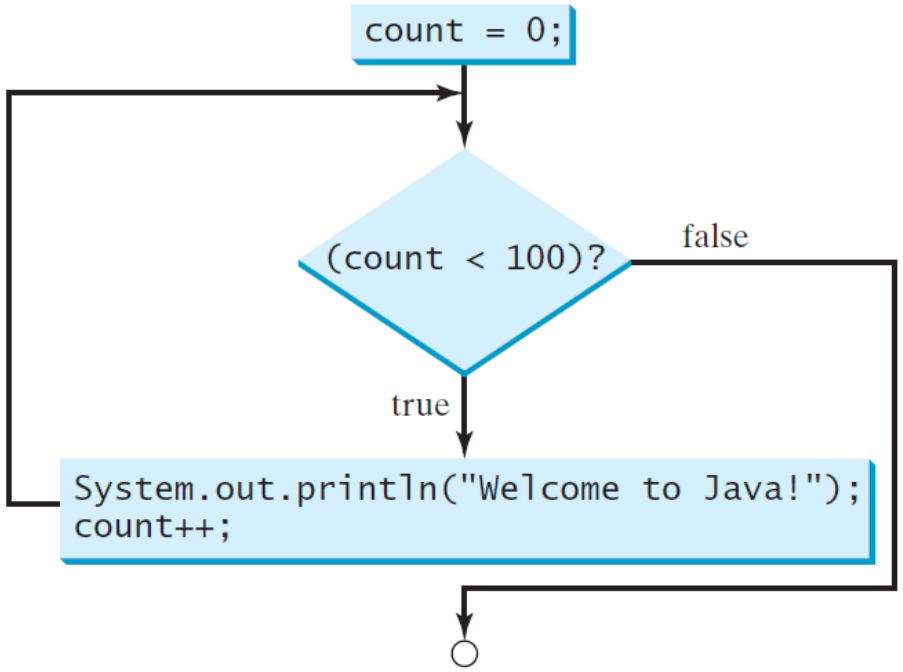
Chapter 5 Loops

`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++;
}
```



animation

Trace while Loop

Initialize count

```
int count = 0;

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

Problem: Repeat Addition Until Correct

Recall that Listing 3.1 AdditionQuiz.java gives a program that prompts the user to enter an answer for a question on addition of two single digits. Using a loop, you can now rewrite the program to let the user enter a new answer until it is correct.

RepeatAdditionQuiz

Run

```
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();

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

Problem: An Advanced Math Learning Tool

The Math subtraction learning tool program generates just one question for each run. You can use a loop to generate questions repeatedly. This example gives a program that generates five questions and reports the number of the correct answers after a student answers all five questions.

SubtractionQuizLoop Run

Ending a Loop with a Sentinel Value

You may use an input value to signify the end of the loop. Such a value is known as a *sentinel value*.

```
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();
// 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();
}
```

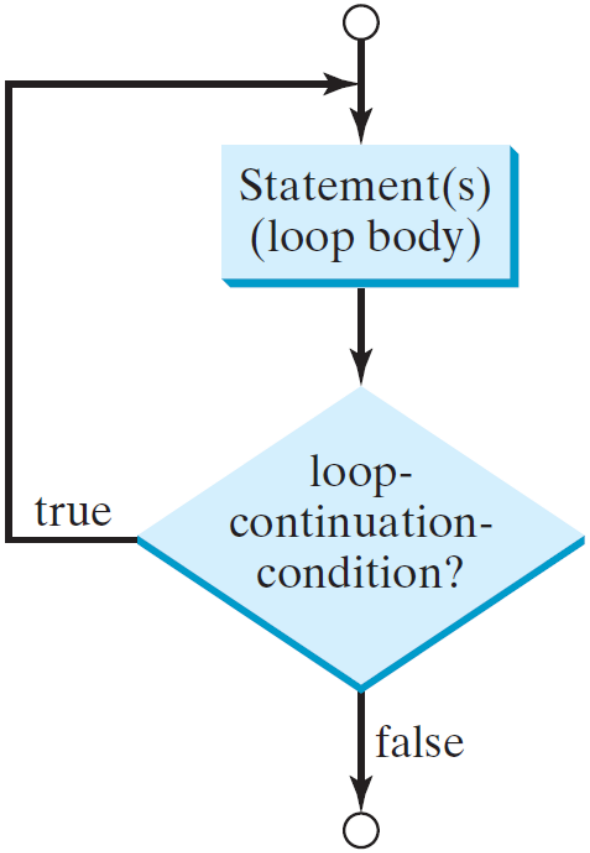
SentinelValue Run

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 + ... + 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);
```

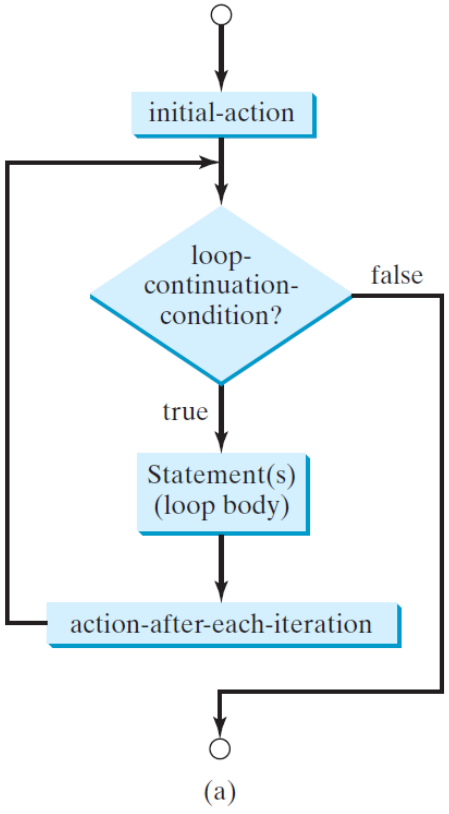
do-while Loop




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

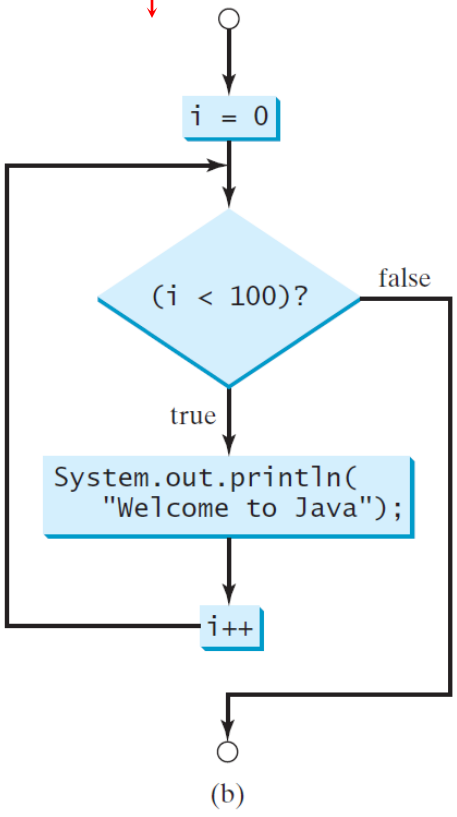
`for` Loops

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



(a)

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



(b)

Trace for Loop

Declare i

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

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);
}
```

Caution, cont.

Similarly, the following loop is also wrong:

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

Logic Error

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

Problem:
Finding the Greatest Common Divisor

Problem: Write a program that prompts the user to enter two positive integers and finds their greatest common divisor.

```
// Prompt the user to enter two integers
System.out.print("Enter first integer: ");
int n1 = input.nextInt();
System.out.print("Enter second integer: ");
int n2 = input.nextInt();
int gcd = 1;
int k = 2;
while (k <= n1 && k <= n2) {
        if (n1 % k == 0 && n2 % k == 0)
                gcd = k;
        k++;
}
```

GreatestCommonDivisor Run

Case Study: *Converting Decimals to Hexadecimals*

Hexadecimals are often used in computer systems programming (see Appendix F for an introduction to number systems). How do you convert a decimal number to a hexadecimal number? To convert a decimal number *d* to a hexadecimal number is to find the hexadecimal digits h_n, h_{n-1}, h_{n-2}, ... , h_2, h_1, and h_0 such that

$$d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + ... + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

These hexadecimal digits can be found by successively dividing *d* by 16 until the quotient is 0. The remainders are h_0, h_1, h_2, ... , h_{n-2}, h_{n-1}, and h_n.

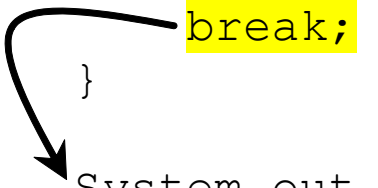
Dec2Hex Run

break

```
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);
  }
}
```



continue

```
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);
  }
}
```

Problem: Checking Palindrome

A string is a palindrome if it reads the same forward and backward. The words “mom,” “dad,” and “noon,” for instance, are all palindromes.

```
// Prompt the user to enter a string
System.out.print("Enter a string: ");
String s = input.nextLine();
// The index of the first character in the string
int low = 0;
// The index of the last character in the string
int high = s.length() - 1;
boolean isPalindrome = true;
while (low < high) {
        if (s.charAt(low) != s.charAt(high)) {
                isPalindrome = false;
                break;
        }
        low++;
        high--;
}
```

Palindrome

Run

Problem: Displaying Prime Numbers

Problem: Write a program that displays the first 50 prime numbers in five lines, each of which contains 10 numbers. An integer greater than 1 is *prime* if its only positive divisor is 1 or itself. For example, 2, 3, 5, and 7 are prime numbers, but 4, 6, 8, and 9 are not.

Solution: The problem can be broken into the following tasks:

- For number = 2, 3, 4, 5, 6, ..., test whether the number is prime.
- Determine whether a given number is prime.
- Count the prime numbers.
- Print each prime number, and print 10 numbers per line.

```
for (int divisor = 2; divisor <= number / 2; divisor++) {
        if (number % divisor == 0) {
                isPrime = false;
                break;
        }
}
```

PrimeNumber Run