Chapter 4
Control structures and Loops

Motivations

Suppose that you need to print a string (e.g., "Welcome to Java!") a hundred times. It would be tedious to have to write the following statement a hundred times:

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

So, how do you solve this problem?
Opening Problem

Problem:

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

Introducing while Loops

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

- To write programs for executing statements repeatedly using a `while` loop (§4.2).
- To follow the loop design strategy to develop loops (§§4.2.1–4.2.3).
- To control a loop with a sentinel value (§4.2.4).
- To obtain large input from a file using input redirection rather than typing from the keyboard (§4.2.5).
- To write loops using `do-while` statements (§4.3).
- To write loops using `for` statements (§4.4).
- To discover the similarities and differences of three types of loop statements (§4.5).
- To write nested loops (§4.6).
- To learn the techniques for minimizing numerical errors (§4.7).
- To learn loops from a variety of examples (`GCD`, `FutureTuition`, `MonteCarloSimulation`) (§4.8).
- To implement program control with `break` and `continue` (§4.9).
- To write a program that displays prime numbers (§4.10).
- To control a loop with a confirmation dialog (§4.11).

**while Loop Flow Chart**

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

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

Trace while Loop, cont.

```java
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
```
Trace while Loop, cont.

int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
Trace while Loop, cont.

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

(count < 2) is still true since count is 1

Trace while Loop, cont.

int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
Trace while Loop, cont.

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

Increase count by 1
count is 2 now

(count < 2) is false since count is 2 now
Trace while Loop

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

The loop exits. Execute the next statement after the loop.

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.
Ending a Loop with a Sentinel Value

Often the number of times a loop is executed is not predetermined. You may use an input value to signify the end of the loop. Such a value is known as a sentinel value.

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

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 $1 + 0.9 + 0.8 + \ldots + 0.1$:

```java
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);
}

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

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

Trace for Loop, cont.

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

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

(i < 2) is true since i is 0

Trace for Loop, cont.

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

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

(i < 2) is still true since i is 1

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

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

Print Welcome to Java

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

Execute adjustment statement
i now is 2
Trace for Loop, cont.

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

(i < 2) is false since i is 2

Trace for Loop, cont.

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

Exit the loop. Execute the next statement after the loop
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
}
```

Equivalent
```
while (true) {
    // Do something
}
```

(a)  
(b)
Caution

Adding a semicolon at the end of the for clause before the loop body is a common mistake, as shown below:

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

Logic Error

Caution, cont.

Similarly, the following loop is also wrong:

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

In the case of the do loop, the following ; is needed to end the loop.

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

Correct
Which Loop to Use?

The three forms of loop statements, while, do-while, and for, are expressively equivalent; that is, you can write a loop in any of these three forms. For example, a while loop in (a) in the following figure can always be converted into the following for loop in (b):

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

```
for ( ; loop-continuation-condition; ) {
    // Loop body
}
```

A for loop in (a) in the following figure can generally be converted into the following while loop in (b) except in certain special cases (see Review Question 3.19 for one of them):

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

```
while (loop-continuation-condition) {
    // Loop body;
    action-after-each-iteration;
}
```

Recommendations

Use the one that is most intuitive and comfortable for you.

In general, a for loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times.

A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0.

A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.
Nested Loops

Problem: Write a program that uses nested for loops to print a multiplication table.

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.

Solution: Suppose you enter two integers 4 and 2, their greatest common divisor is 2. Suppose you enter two integers 16 and 24, their greatest common divisor is 8. So, how do you find the greatest common divisor? Let the two input integers be \( n_1 \) and \( n_2 \). You know number 1 is a common divisor, but it may not be the greatest common divisor. So you can check whether \( k \) (for \( k = 2, 3, 4, \) and so on) is a common divisor for \( n_1 \) and \( n_2 \), until \( k \) is greater than \( n_1 \) or \( n_2 \).
Problem: Predicting the Future Tuition

Problem: Suppose that the tuition for a university is $10,000 this year and tuition increases 7% every year. In how many years will the tuition be doubled?

double tuition = 10000;  // Year 0
int year = 0  // Year 0
tuition = tuition * 1.07; year++;  // Year 1
tuition = tuition * 1.07; year++;  // Year 2
tuition = tuition * 1.07; year++;  // Year 3
...
Using `break` and `continue`

Examples for using the `break` and `continue` keywords:

- **TestBreak.java**

```java
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);
    }
}
```

- **TestContinue.java**

```java
public class TestContinue {
    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);
    }
}