05 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:

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

So, how do you solve this problem?

Opening Problem

Problem:

100 times

```
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

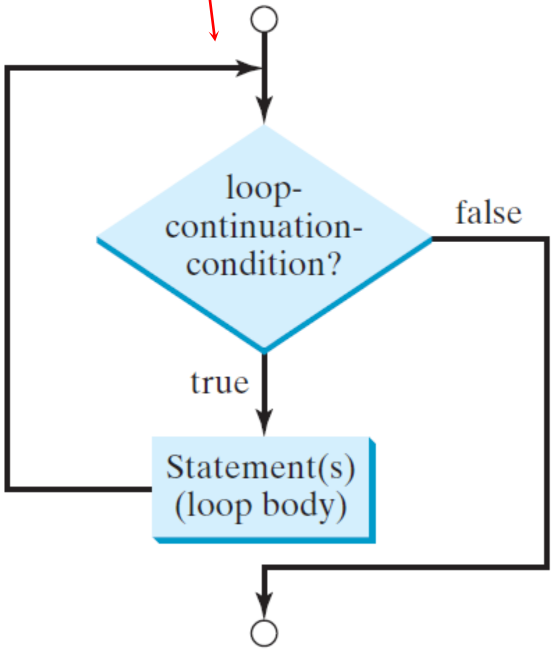
```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java");
  count++;
}
```

Objectives

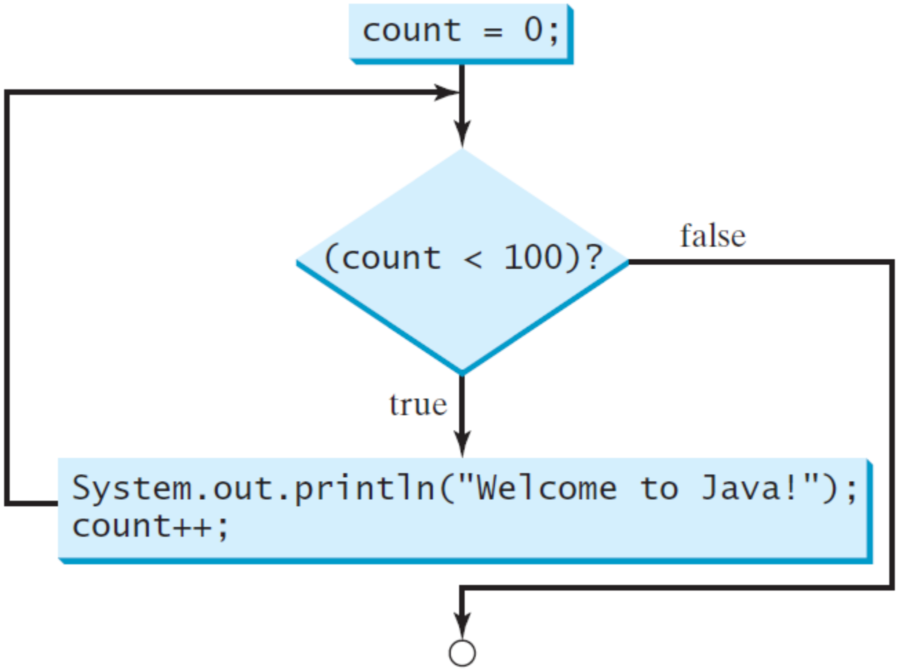
- To write programs for executing statements repeatedly using a **while** loop (§5.2).
- To follow the loop design strategy to develop loops (§§5.2.1–5.2.3).
- To control a loop with a sentinel value (§5.2.4).
- To obtain large input from a file using input redirection rather than typing from the keyboard (§5.2.5).
- To write loops using **do-while** statements (§5.3).
- To write loops using **for** statements (§5.4).
- To discover the similarities and differences of three types of loop statements (§5.5).
- To write nested loops (§5.6).
- To learn the techniques for minimizing numerical errors (§5.7).
- To learn loops from a variety of examples (**GCD**, **FutureTuition**, **Dec2Hex**) (§5.8).
- To implement program control with **break** and **continue** (§5.9).
- To write a program that displays prime numbers (§5.11).

`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

```
int count = 0;

while (count < 2) {

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

  count++;

}
```

Initialize count

animation

Trace while Loop, cont.

(count < 2) is true

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

animation

Trace while Loop, cont.

Print Welcome to Java

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

animation

Trace while Loop, cont.

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

Increase count by 1
count is 1 now

animation

Trace while Loop, cont.

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

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

animation

Trace while Loop, cont.

Print Welcome to Java

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

animation

Trace while Loop, cont.

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

Increase count by 1
count is 2 now

animation

Trace while Loop, cont.

(count < 2) is false since count is 2 now

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

animation

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.

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 (报警阈值).*

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

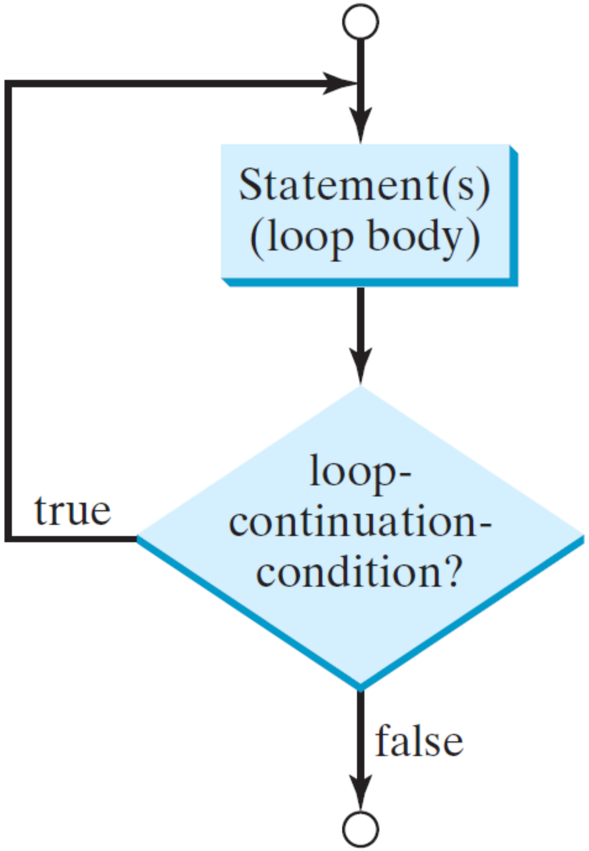
SentinelValue Run

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

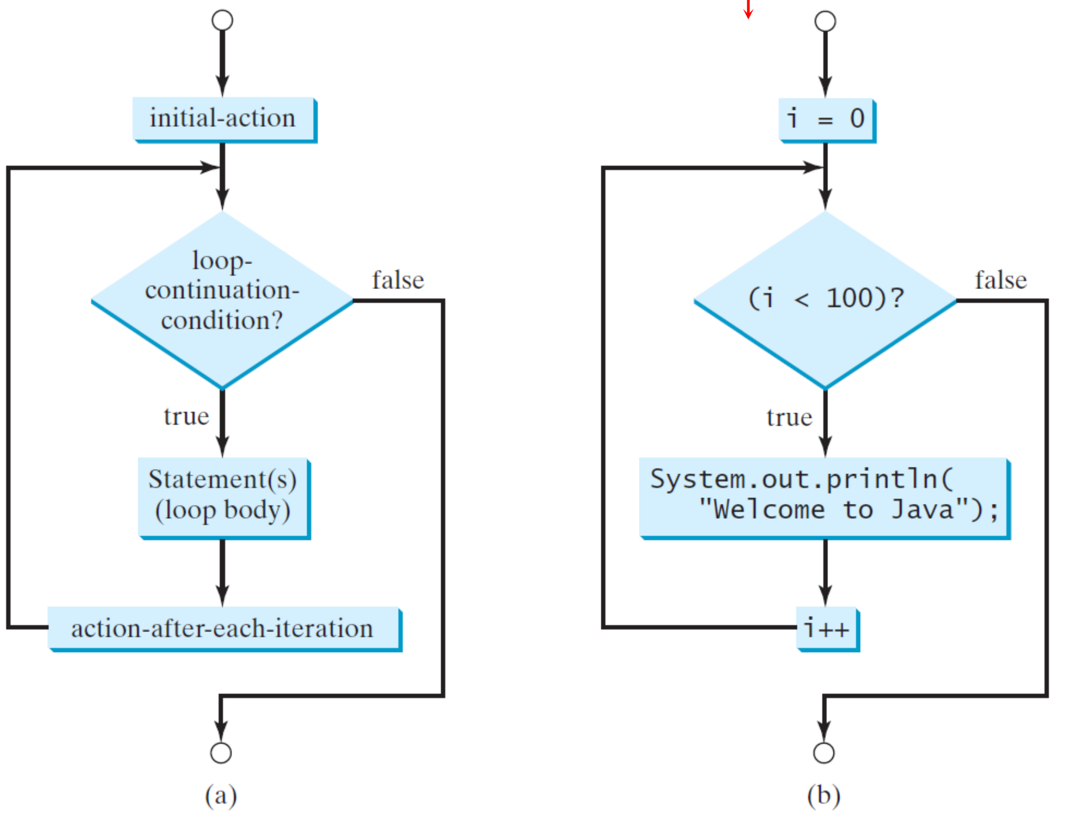


```
do {
  // Loop body;
  Statement(s);
} while (loop-continuation-condition);
```

for Loops

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

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


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

animation

Trace for Loop

Declare i

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

Trace for Loop, cont.

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

Execute initializer
i is now 0

animation

Trace for Loop, cont.

```
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.

Print Welcome to Java

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

animation

Trace for Loop, cont.

Execute adjustment statement
i now is 1

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

animation

Trace for Loop, cont.

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

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

animation

Trace for Loop, cont.

Print Welcome to Java

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

animation

Trace for Loop, cont.

Execute adjustment statement
i now is 2

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

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
}
```

(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:

Logic
Error

```
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);        Logic Error
{
  System.out.println("i is " + i);
  i++;
}
```

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
```

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
}
```

(a)

Equivalent

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

(b)

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;
}
```

(a)

Equivalent

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

(b)

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.

Using `break` and `continue`

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

□ TestBreak.java

TestBreak

Run

□ TestContinue.java

TestContinue

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

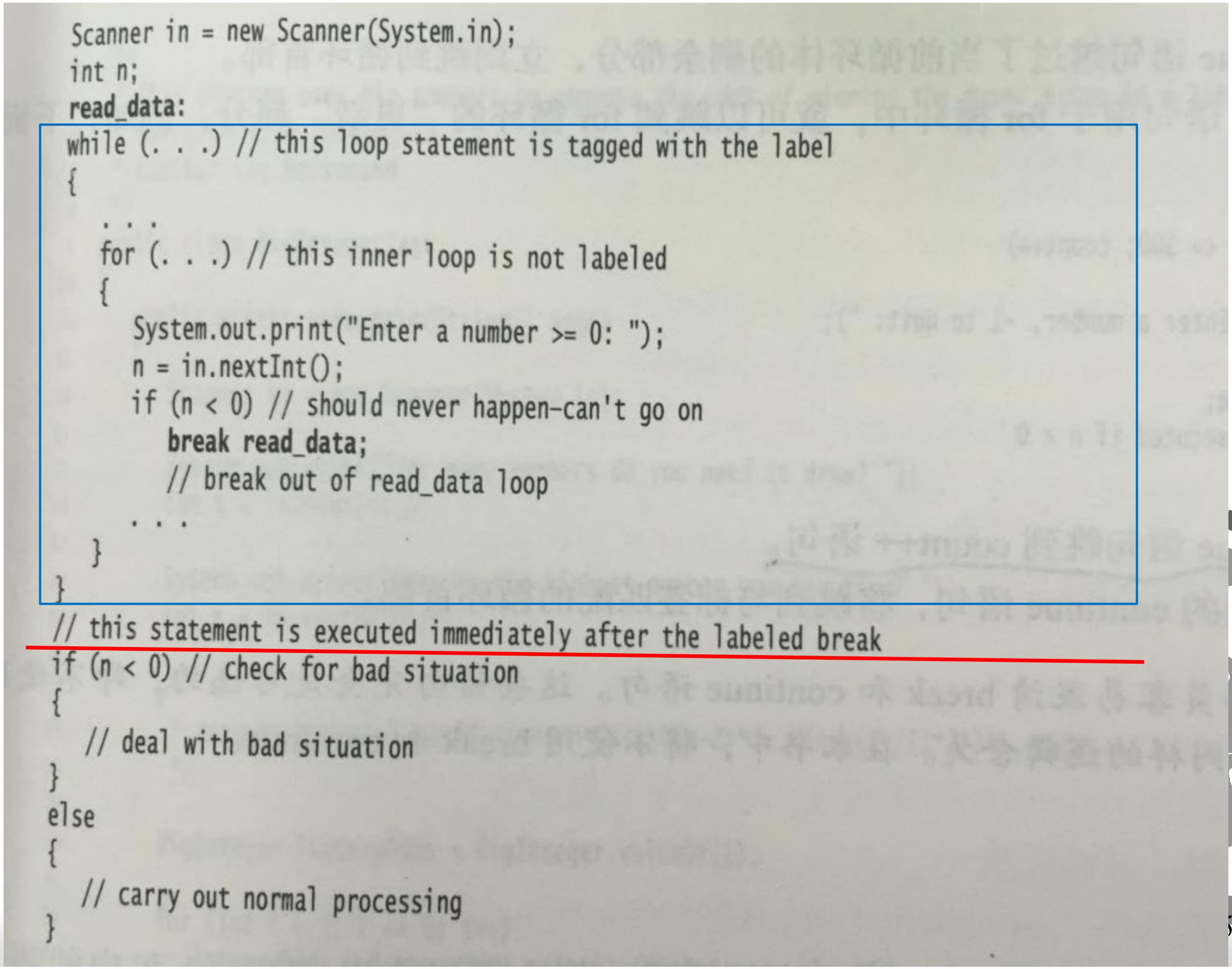
```
public class TestContinue {
  public static void main(String[] args) {
    int sum = 0;
    int number = 0;

    while (number < 20) {
      number++;
      if (number == 10 || number == 11)
        continue;
      sum += number;
    }

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

break

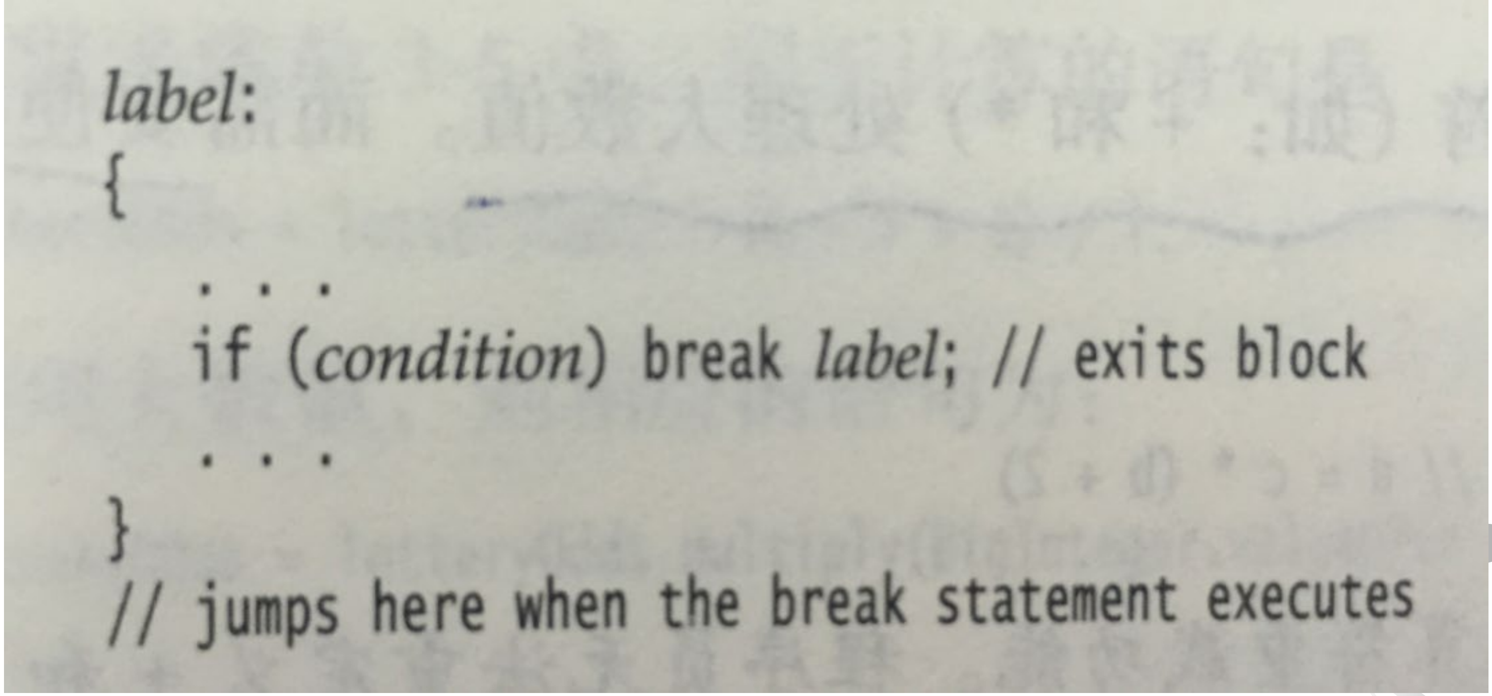
- 与C++不同，Java还提供了带标签的break语句



```
Scanner in = new Scanner(System.in);
int n;
read_data:
while (. . .) // this loop statement is tagged with the label
{
   . . .
   for (. . .) // this inner loop is not labeled
   {
      System.out.print("Enter a number >= 0: ");
      n = in.nextInt();
      if (n < 0) // should never happen-can't go on
         break read_data;
         // break out of read_data loop
      . . .
   }
}
// this statement is executed immediately after the labeled break
if (n < 0) // check for bad situation
{
   // deal with bad situation
}
else
{
   // carry out normal processing
}
```

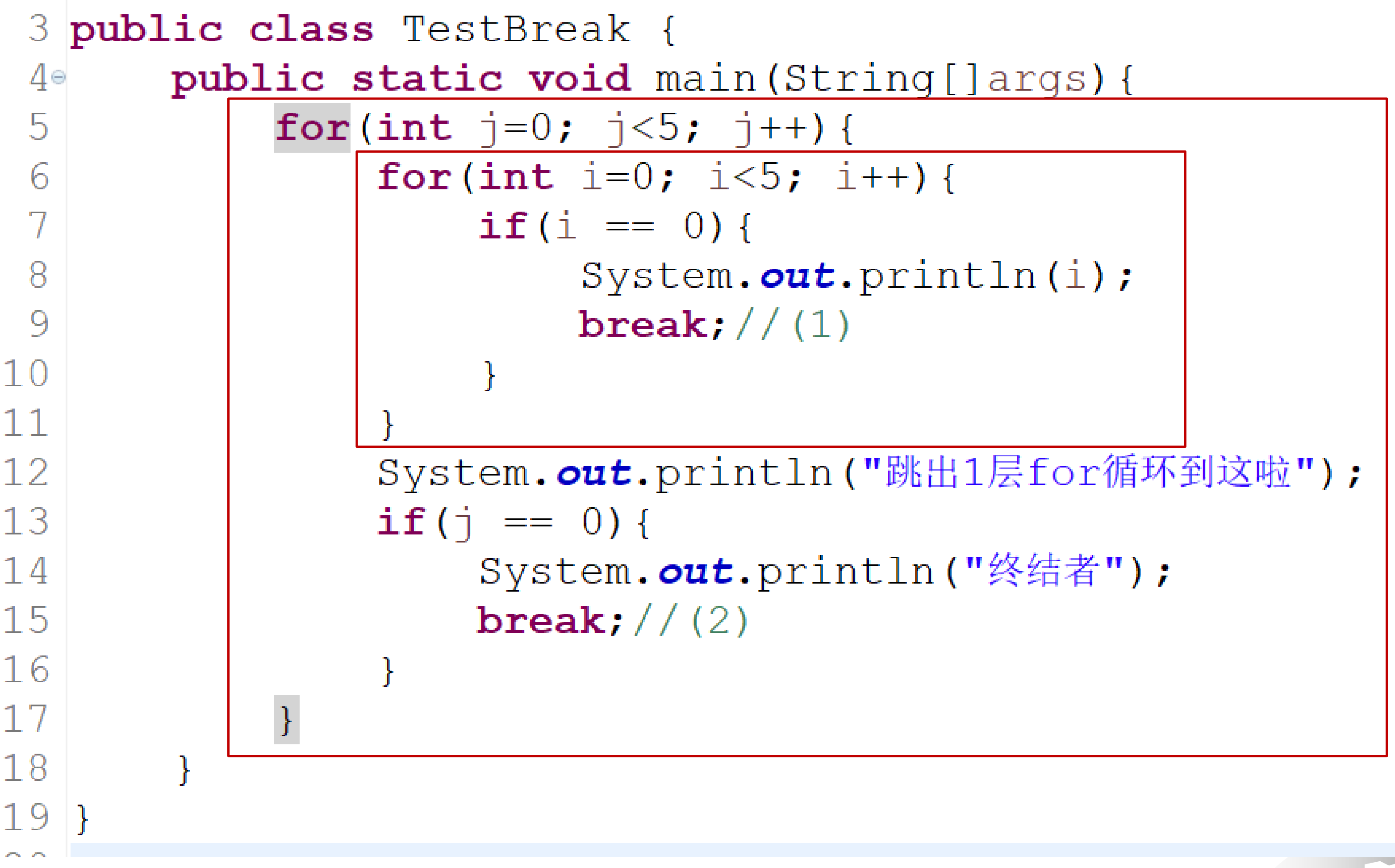
break

- 可以将标签用到任何语句中，甚至在if或块语句中

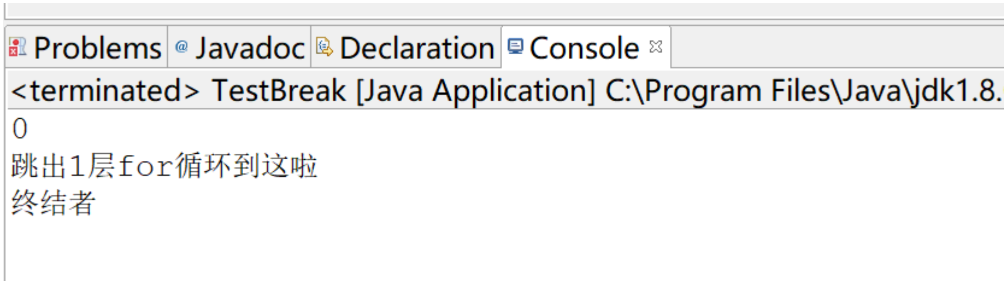


```
label:
{
    . . .
    if (condition) break label; // exits block
    . . .
}
// jumps here when the break statement executes
```

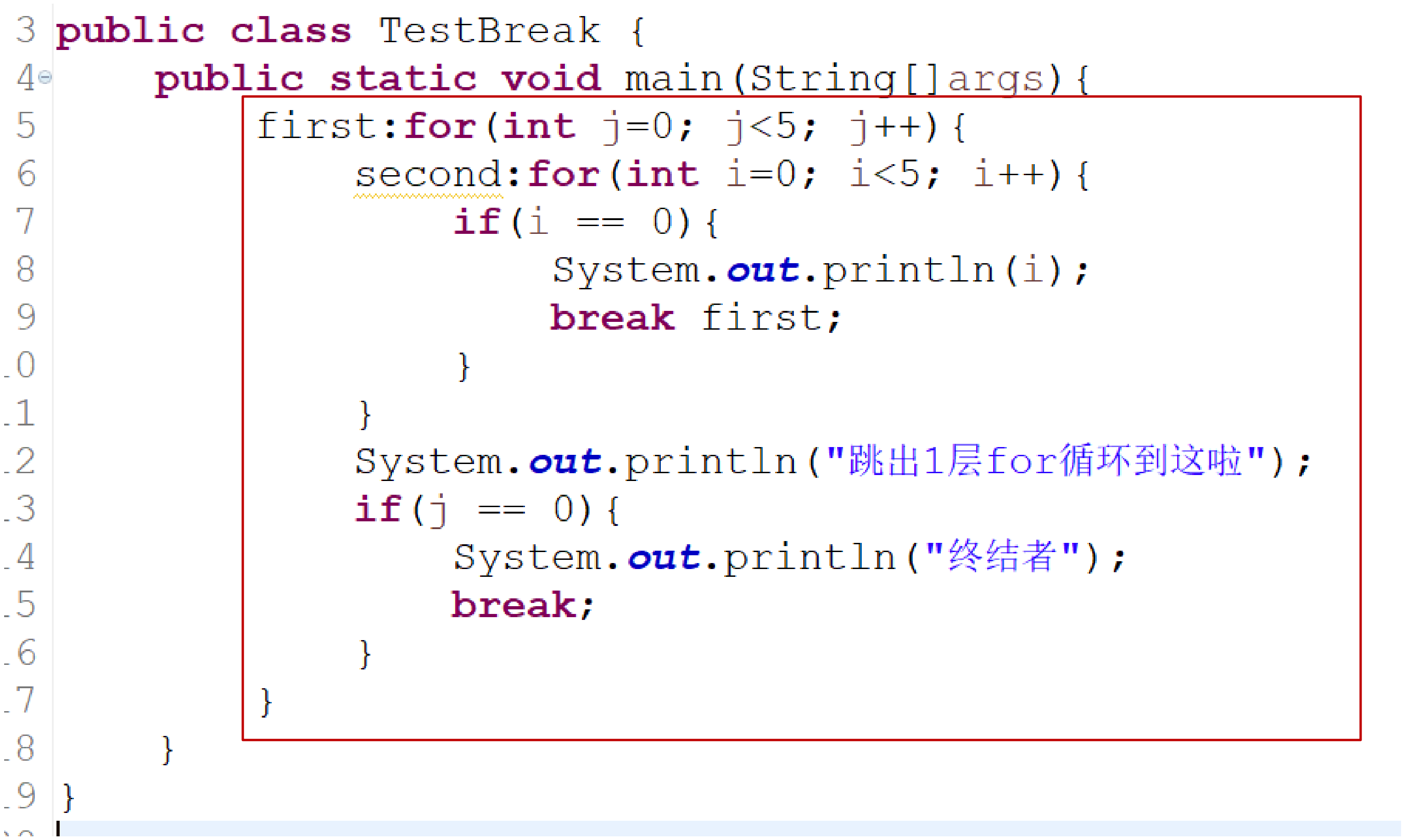
- 需要注意的是：智能跳出语句块，而不能跳入语句块。



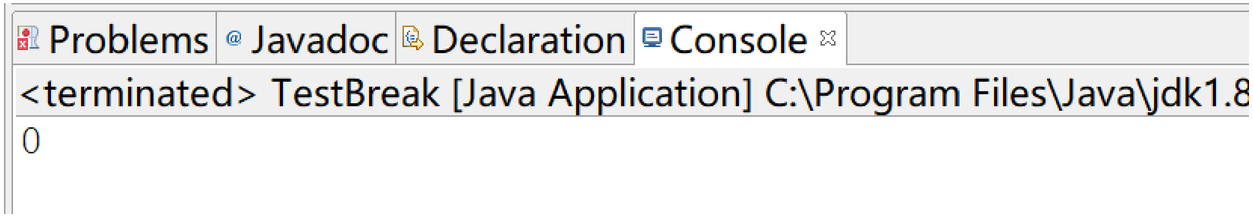
```
public class TestBreak {
    public static void main(String[]args){
        for(int j=0; j<5; j++){
            for(int i=0; i<5; i++){
                if(i == 0){
                    System.out.println(i);
                    break;//(1)
                }
            }
            System.out.println("跳出1层for循环到这啦");
            if(j == 0){
                System.out.println("终结者");
                break;//(2)
            }
        }
    }
}
```



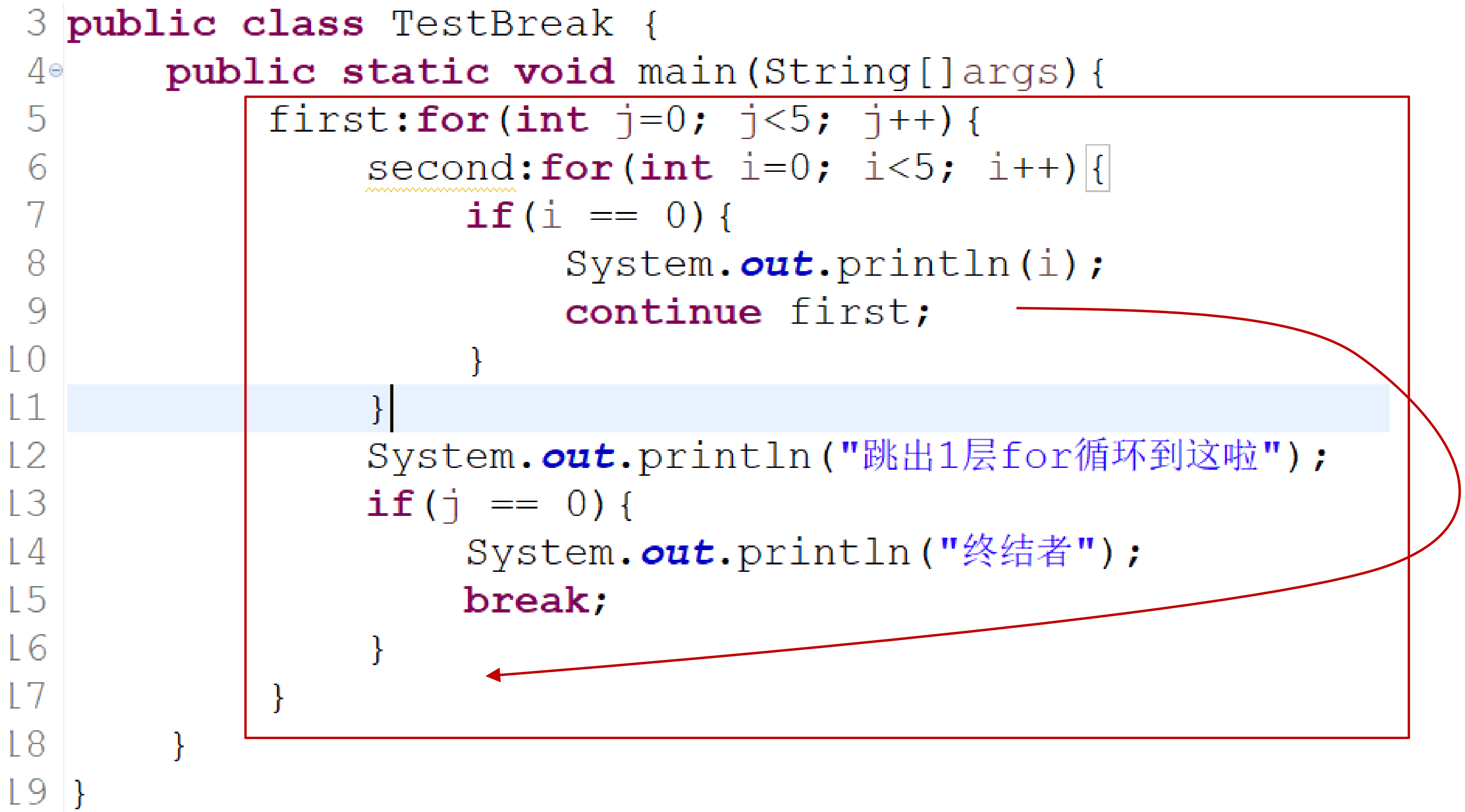
```
<terminated> TestBreak [Java Application] C:\Program Files\Java\jdk1.8.
0
跳出1层for循环到这啦
终结者
```



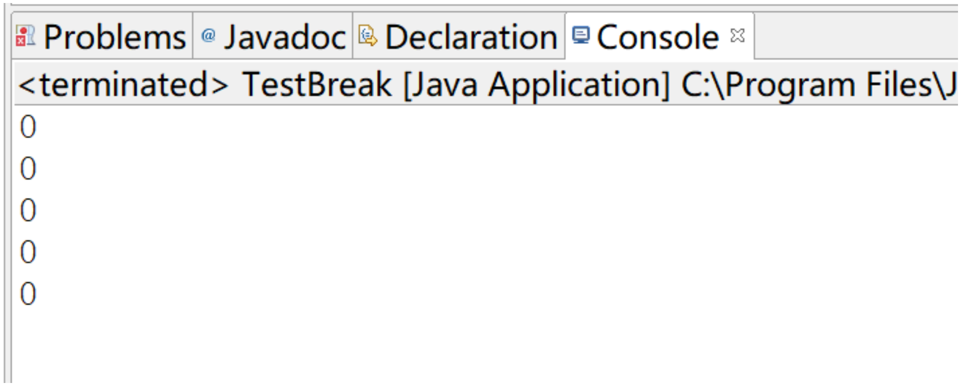
```java
public class TestBreak {
    public static void main(String[]args){
        first:for(int j=0; j<5; j++){
            second:for(int i=0; i<5; i++){
                if(i == 0){
                    System.out.println(i);
                    break first;
                }
            }
            System.out.println("跳出1层for循环到这啦");
            if(j == 0){
                System.out.println("终结者");
                break;
            }
        }
    }
}
```



```
<terminated> TestBreak [Java Application] C:\Program Files\Java\jdk1.8
0
```



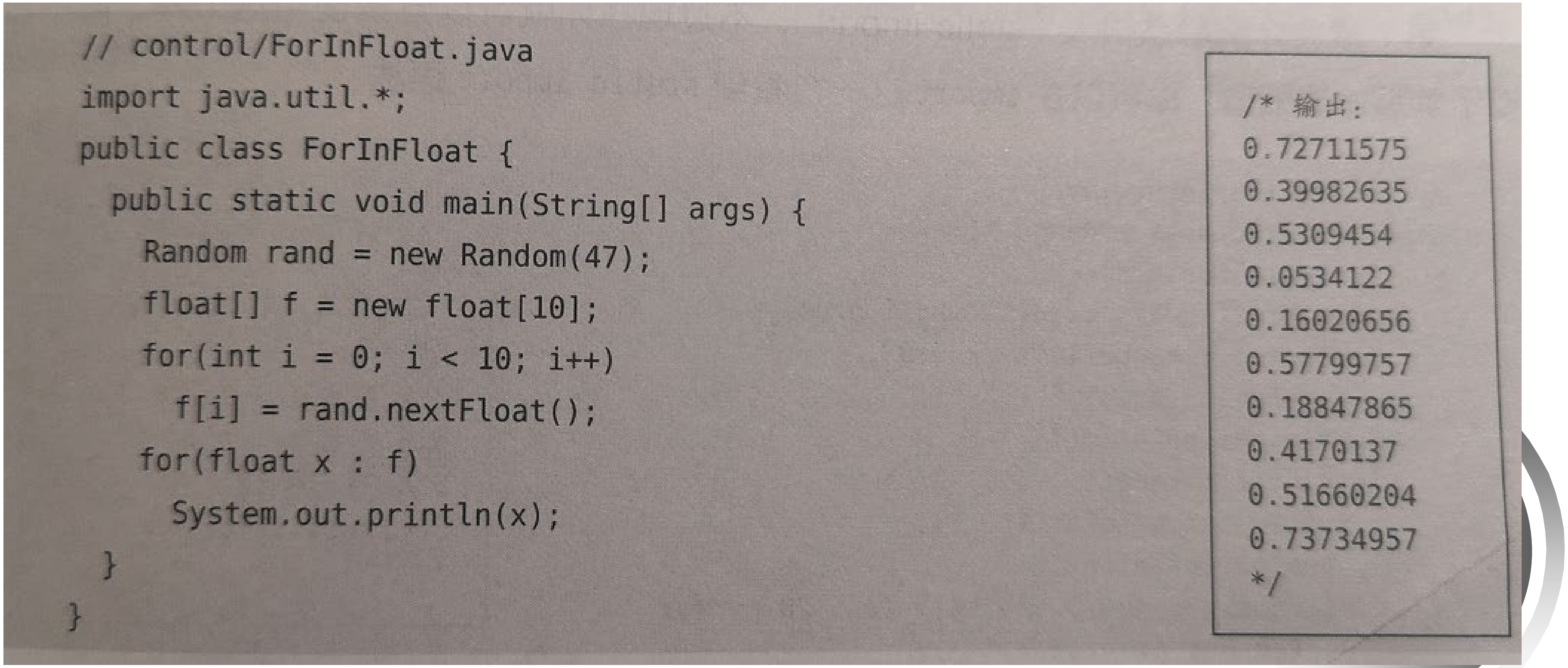
```java
public class TestBreak {
    public static void main(String[]args){
        first:for(int j=0; j<5; j++){
            second:for(int i=0; i<5; i++){
                if(i == 0){
                    System.out.println(i);
                    continue first;
                }
            }
            System.out.println("跳出1层for循环到这啦");
            if(j == 0){
                System.out.println("终结者");
                break;
            }
        }
    }
}
```



```
<terminated> TestBreak [Java Application] C:\Program Files\J
0
0
0
0
0
```

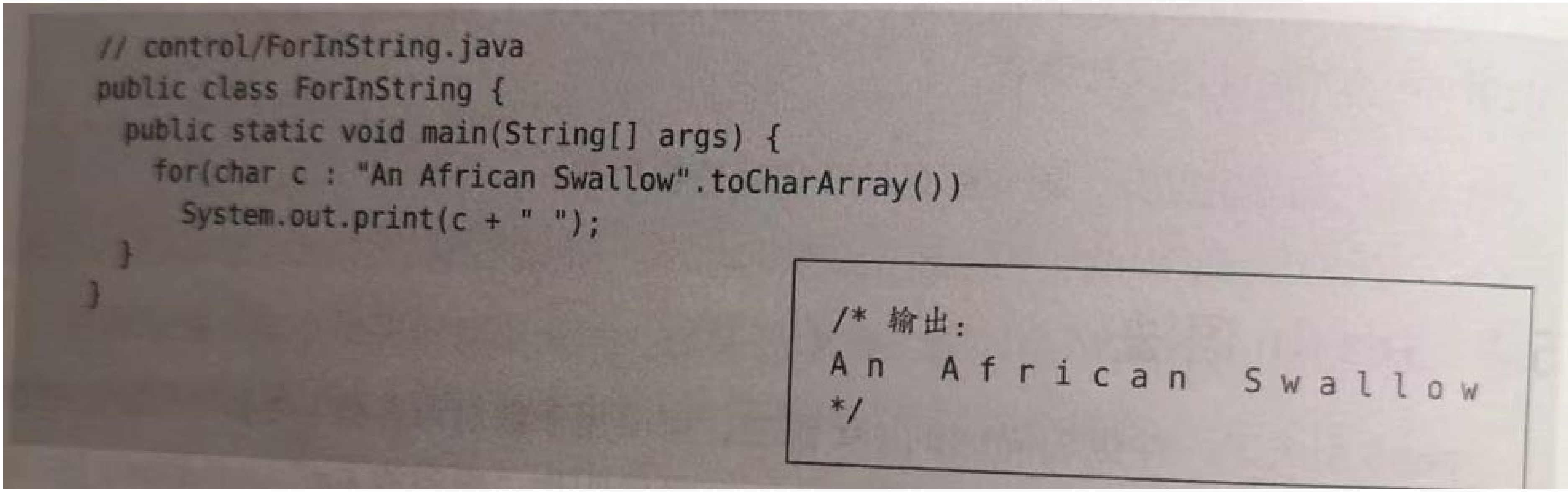
for-in 语法

- Java 5引入了更加简洁的for语法，用于数组和容器。



```
// control/ForInFloat.java
import java.util.*;
public class ForInFloat {
  public static void main(String[] args) {
    Random rand = new Random(47);
    float[] f = new float[10];
    for(int i = 0; i < 10; i++)
      f[i] = rand.nextFloat();
    for(float x : f)
      System.out.println(x);
  }
}
```

```
/* 输出:
0.72711575
0.39982635
0.5309454
0.0534122
0.16020656
0.57799757
0.18847865
0.4170137
0.51660204
0.73734957
*/
```

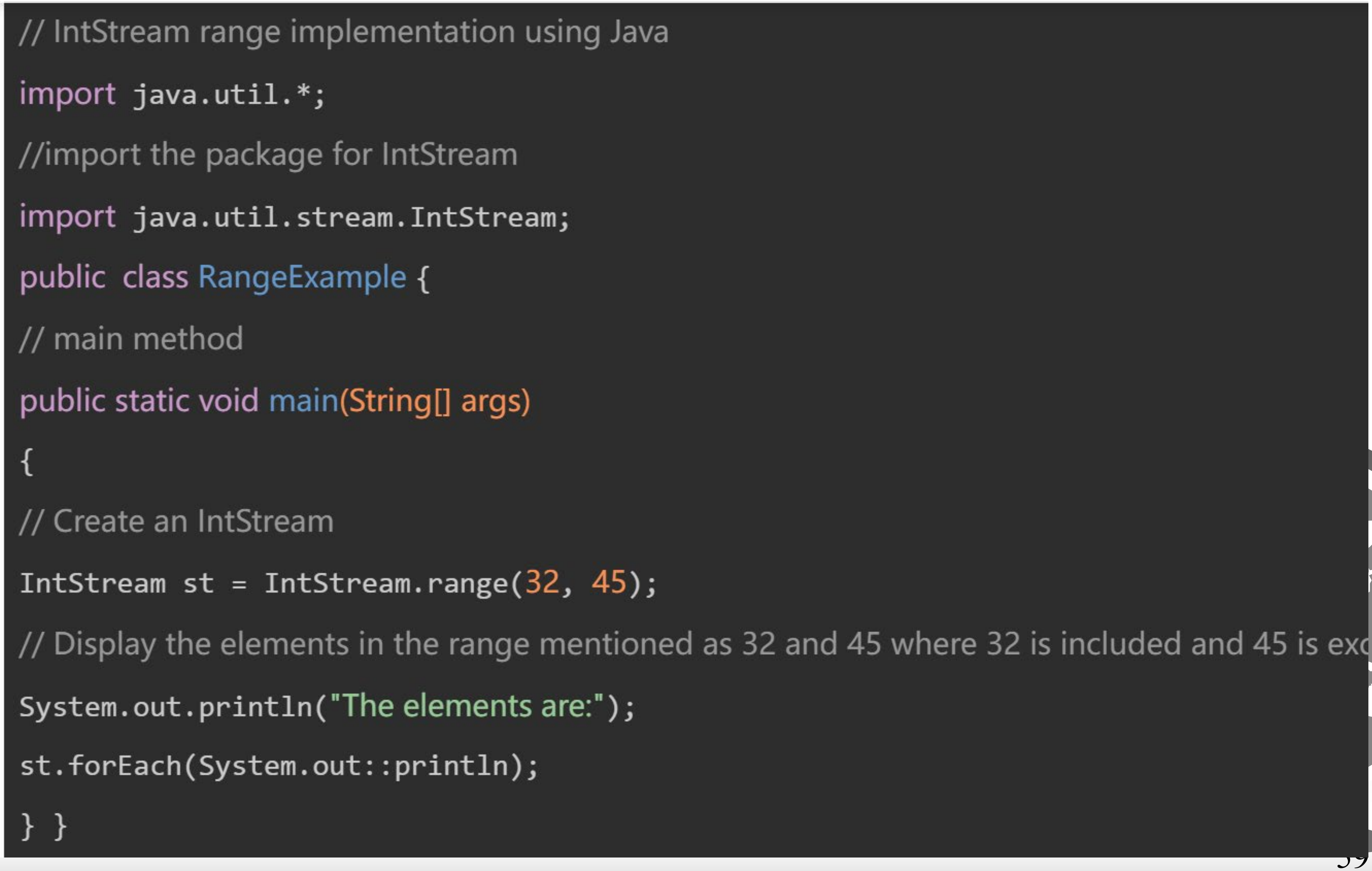


```
// control/ForInString.java
public class ForInString {
  public static void main(String[] args) {
    for(char c : "An African Swallow".toCharArray())
      System.out.print(c + " ");
  }
}
```

```
/* 输出:
A n   A f r i c a n   S w a l l o w
*/
```

range()方法

- Java 8中，可以利用range()方法来建立流。【后面会详细讲stream相关内容】



```
// IntStream range implementation using Java
import java.util.*;
//import the package for IntStream
import java.util.stream.IntStream;
public  class RangeExample {
// main method
public static void main(String[] args)
{
// Create an IntStream
IntStream st = IntStream.range(32, 45);
// Display the elements in the range mentioned as 32 and 45 where 32 is included and 45 is exc
System.out.println("The elements are:");
st.forEach(System.out::println);
} }
```