Repetition Statement (Loops)

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?

Liang, Introduction to Java Programming, Eighth Edition, (c) 2011 Pearson Education, Inc. All rights reserved. 0132130807

Opening 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!");
```

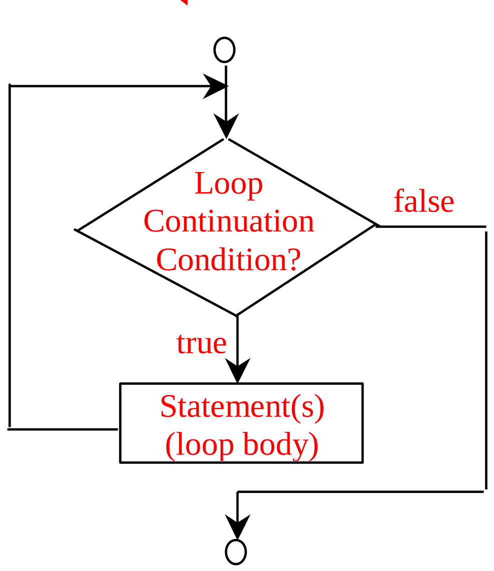
Introducing while Loops

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

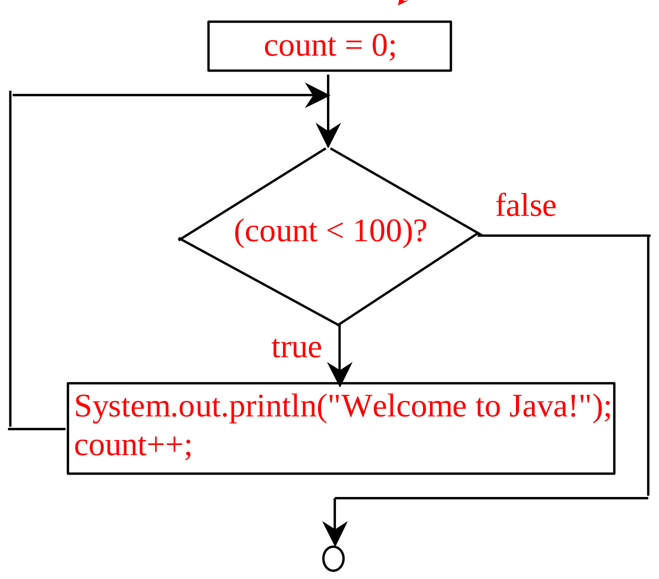
`while` Loop Flow Chart

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

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



(A)



(B)

Trace while Loop

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

Problem: Guessing Numbers

Write a program that randomly generates an integer between 0 and 10, inclusive. The program prompts the user to enter a number continuously until the number matches the randomly generated number. For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently. Here is a sample run:

Guessing Numbers

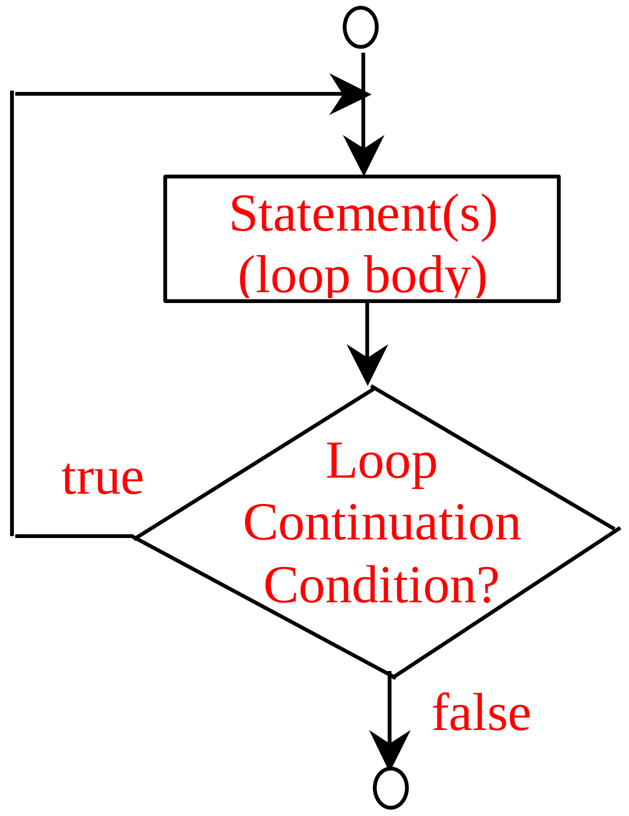
```
import java.util.Scanner;

public class GuessNumberUsingBreak {
  public static void main(String[] args) {
        // Generate a random number to be guessed
        int number = (int)(Math.random() * 11);
        Scanner input = new Scanner(System.in);
        System.out.println("Guess a magic number between 0 and 10");
```

Guessing Numbers

```
        while (true) {
            // Prompt the user to guess the number
            System.out.print("\nEnter your guess: ");
            int guess = input.nextInt();
            if (guess == number) {
              System.out.println("Yes, the number is " + number);
              break;
            } else if (guess > number)
              System.out.println("Your guess is too high");
            else System.out.println("Your guess is too low");
        } // End of loop
    } // end main function
} // end of class
```

`do-while` Loop

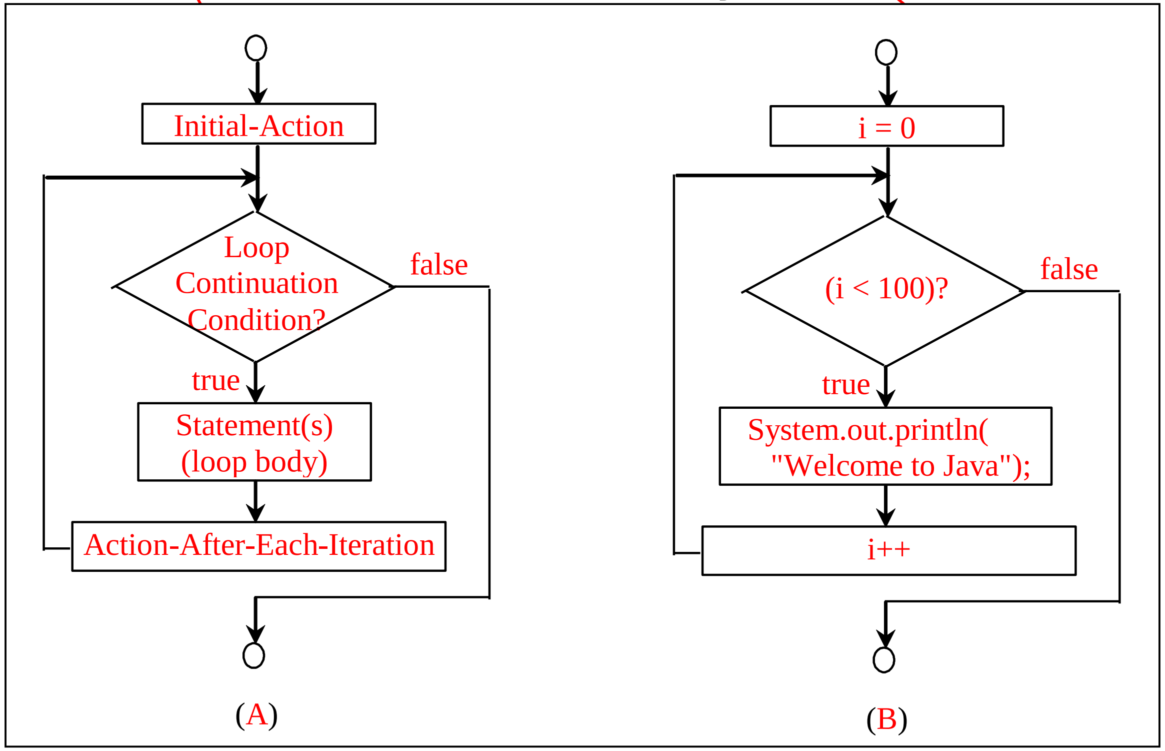


```
do {
  // Loop body;
  Statement(s);
} while (loop-continuation-condition);
```

`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)

Trace for Loop

```
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:

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);
{
  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

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.

Nested Loops

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

Using `break` and `continue`

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 n1 and n2. You know number 1 is a common divisor, but it may not be the greatest commons divisor. So you can check whether k (for k = 2, 3, 4, and so on) is a common divisor for n1 and n2, until k is greater than n1 or n2.

GreatestCommonDivisor

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.