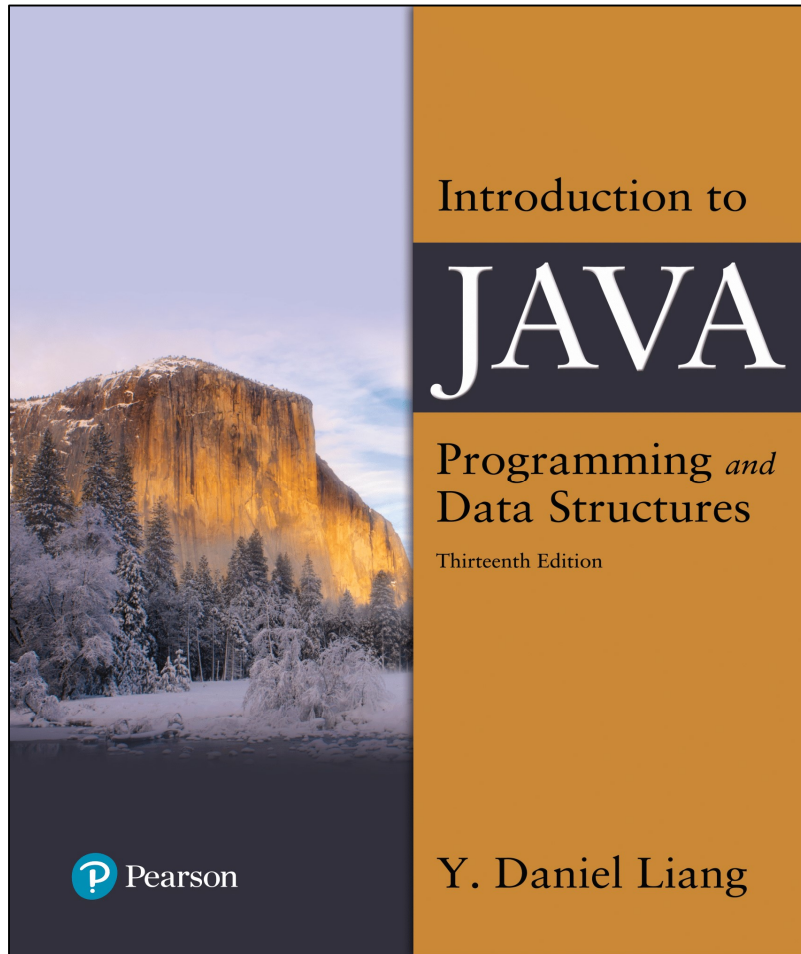


Introduction to Java Programming and Data Structures

Thirteenth Edition



Chapter 5

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!");
```

LOOPS

- Allows you to repeat block of statements several times.
- Java Loops
 - while loop
 - do while loop
 - for loops

F Counter Loop

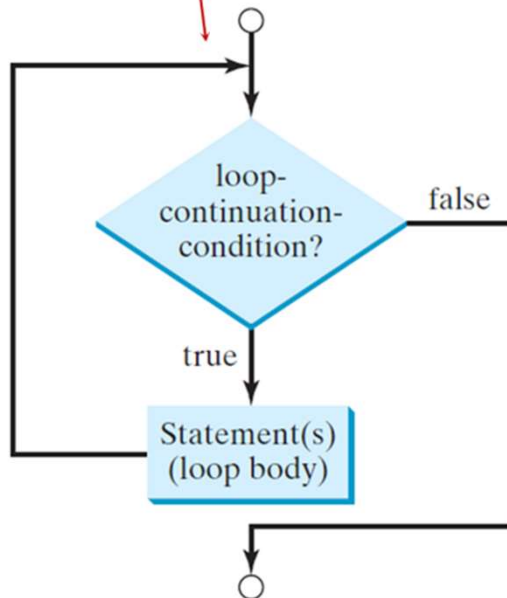
- Number of iterations are known before execution starts.

F Conditional Loop

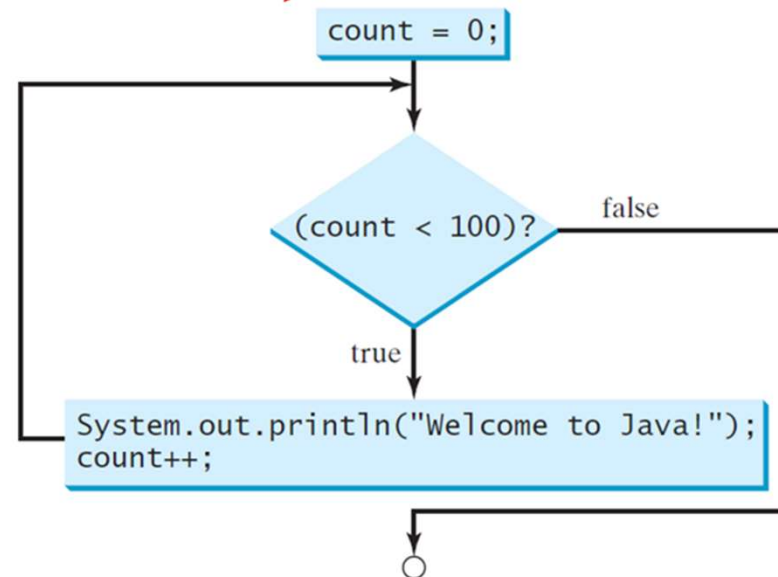
- Number of iterations are unknown before execution starts

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++;
}
```



Trace while Loop (1 of 9)

```
int count = 0;
```

Initialize count

```
while (count < 2) {
```

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

```
    count++;
```

```
}
```

Trace while Loop (2 of 9)

```
int count = 0;
```

```
while (count < 2) {
```

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

```
    count++;
```

```
}
```

(count < 2) is true

Trace while Loop (3 of 9)

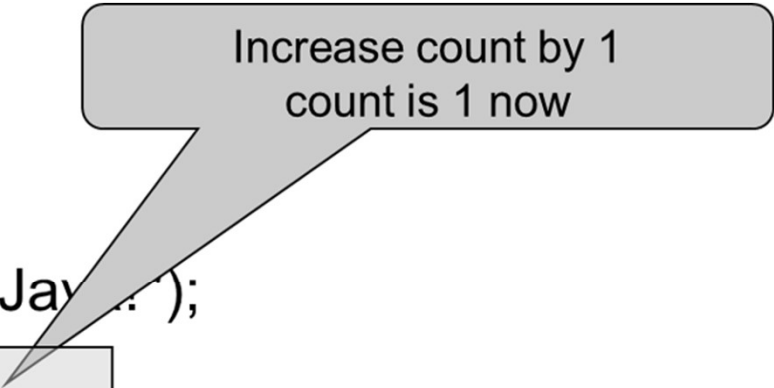
```
int count = 0;  
while (count < 2) {  
    System.out.println("Welcome to Java!");  
    count++;  
}
```

Print Welcome to Java



Trace while Loop (4 of 9)

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



Increase count by 1
count is 1 now

Trace while Loop (5 of 9)

```
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 (6 of 9)

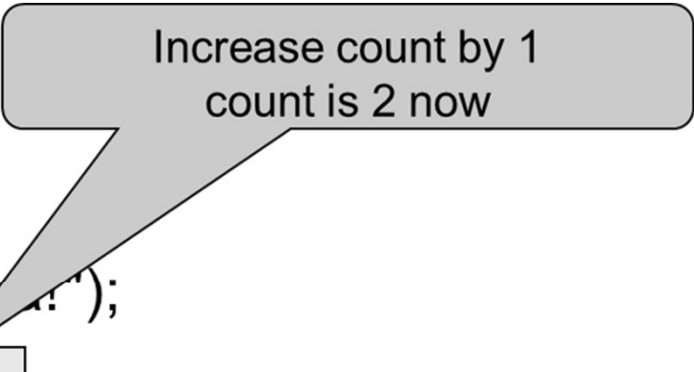
```
int count = 0;  
while (count < 2) {  
    System.out.println("Welcome to Java!");  
    count++;  
}
```



Print Welcome to Java

Trace while Loop (7 of 9)

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



Increase count by 1
count is 2 now

Trace while Loop (8 of 9)

```
int count = 0;
```

```
while (count < 2) {
```

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

```
    count++;
```

```
}
```

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

Trace while Loop (9 of 9)

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

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

EXAMPLES

Counter Loops

- 👉 F Print from 1 to 10
- 👉 Print from 20 to 50
- 👉 Print odd integers between 1 and 100
- 👉 Find $\sum_{k=1}^{100}(k)$
- 👉 Given list of 10 integer values find the sum.

EXAMPLES

Conditional Loops

- Given an integer value num, find the sum of its digits.

For example:

Input is 1024

Output is 7

- Given an integer value num, calculate and print the sum of num and its reverse

For example:

Input is 1024

Output is

$1024 + 4201 = 5225$

Problem: Guessing Numbers

Write a program that randomly generates an integer between **0** and **100**, 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:

[GuessNumberOneTime](#)

[GuessNumber](#)

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.

SentinelValue

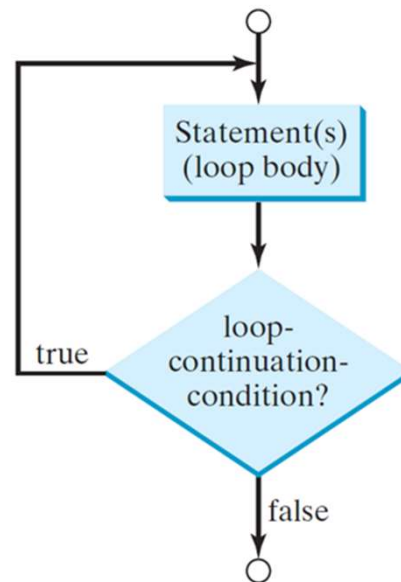
Caution (1 of 3)

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

```
double item = 1; double sum = 0;
while (item != 0) { // No guarantee item will be 0
    sum += item;
    item -= 0.1;
}
System.out.println(sum);
```

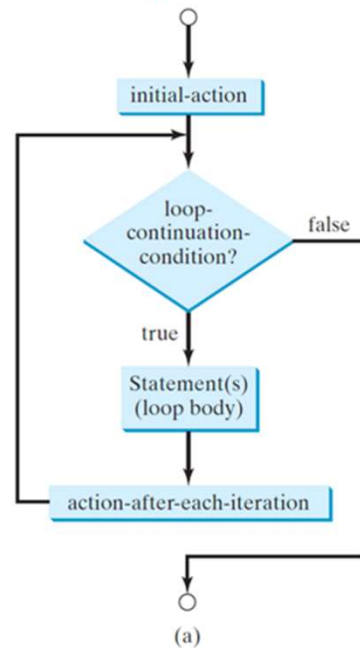
do-while Loop

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

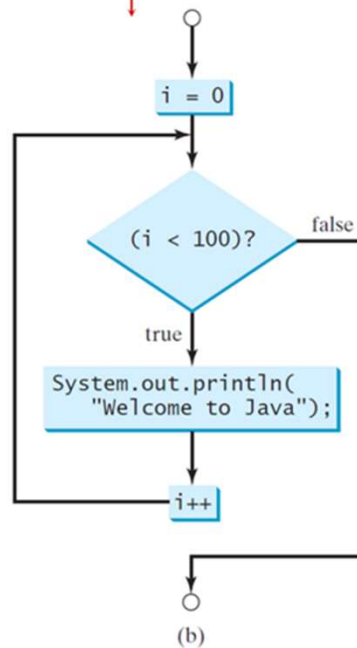


for Loops

```
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!");  
}
```



Trace for Loop (1 of 10)

```
int i;
```

Declare i

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

Trace for Loop (2 of 10)

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

Execute initializer
i is now 0

Trace for Loop (3 of 10)

```
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 (4 of 10)

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



Print Welcome to Java

Trace for Loop (5 of 10)

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

Execute adjustment statement
i now is 1

Trace for Loop (6 of 10)

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

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

Trace for Loop (7 of 10)

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

Print Welcome to Java

Trace for Loop (8 of 10)

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

Execute adjustment statement
i now is 2

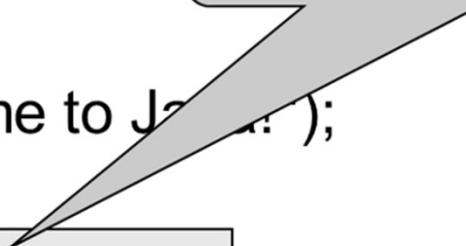
Trace for Loop (9 of 10)

```
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 (10 of 10)

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



Exit the loop. Execute the next statement after the loop

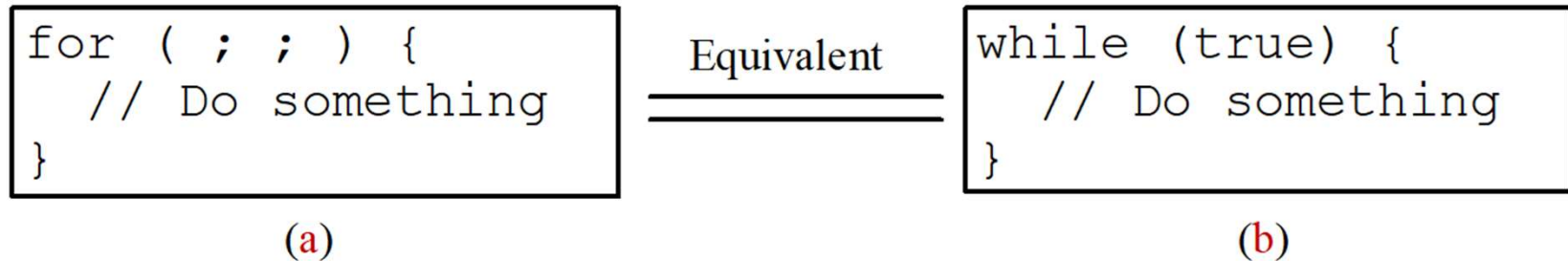
Note (1 of 2)

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 (2 of 2)

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:




Caution (2 of 3)

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);  
}
```

Logic Error



Caution (3 of 3)

Similarly, the following loop is also wrong:

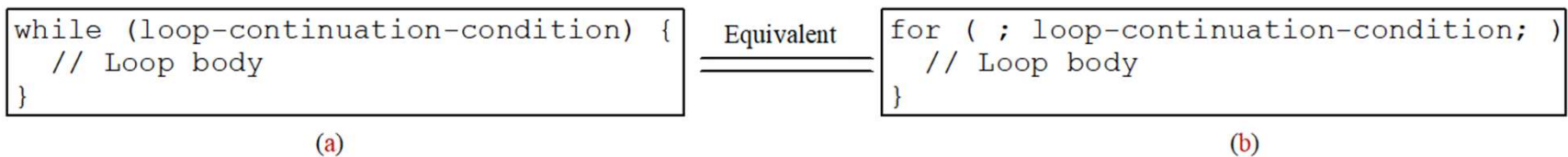
```
int i=0;
while (i < 10); ← Logic Error
{
    System.out.println("i is " + i);
    i++;
}
```

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

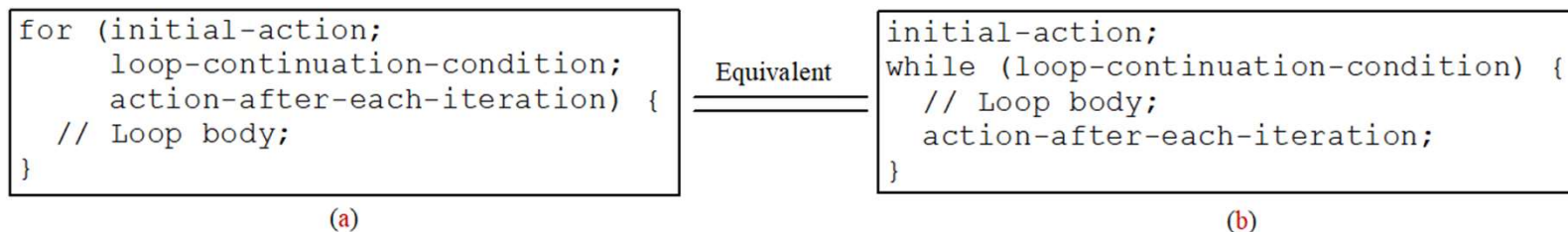
```
int i=0;
do {
    System.out.println("i is " + i);
    i++;
} while (i<10); ← Correct
```

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):



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):



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

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	8	4	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

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 common 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](#)

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?

FutureTuition

Problem: Predicating the Future Tuition

```
double tuition = 10000; 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  
...
```

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

$h_n, h_{n-1}, h_{n-2}, \dots, h_2, h_1$, and h_0 such that

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

These hexadecimal digits can be found by successively dividing d by 16 until the quotient is 0. The remainders are

$h_0, h_1, h_2, \dots, h_{n-2}, h_{n-1}$, and h_n .

[Dec2Hex](#)

Using break and continue

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

- TestBreak.java


[TestBreak](#)

- TestContinue.java

[TestContinue](#)

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);  
    }  
}
```

Guessing Number Problem Revisited

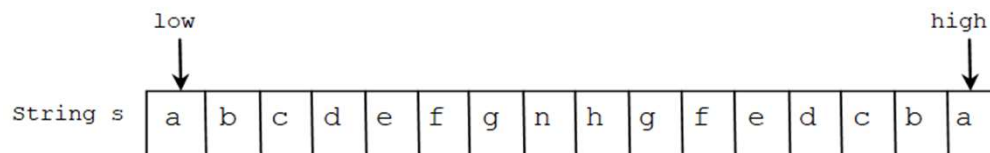
Here is a program for guessing a number. You can rewrite it using a **break** statement.

[GuessNumberUsingBreak](#)

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.

The problem is to write a program that prompts the user to enter a string and reports whether the string is a palindrome. One solution is to check whether the first character in the string is the same as the last character. If so, check whether the second character is the same as the second-to-last character. This process continues until a mismatch is found or all the characters in the string are checked, except for the middle character if the string has an odd number of characters.



Palindrome

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.

[PrimeNumber](#)

Practice Problems

(Find the factors of an integer) Write a program that reads an integer and displays all its smallest factors in an increasing order.

For example, if the input integer is 120, the output should be as follows: 2, 2, 2, 3, 5.

(*Sum a series*) Write a program to compute the following summation:

$$\frac{1}{3} + \frac{3}{5} + \frac{5}{7} + \frac{7}{9} + \dots + \frac{95}{97} + \frac{97}{99}$$