Chapter 5 Loops in Java

Prepared By:

Dr. Muhanad Tahrir Younis

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

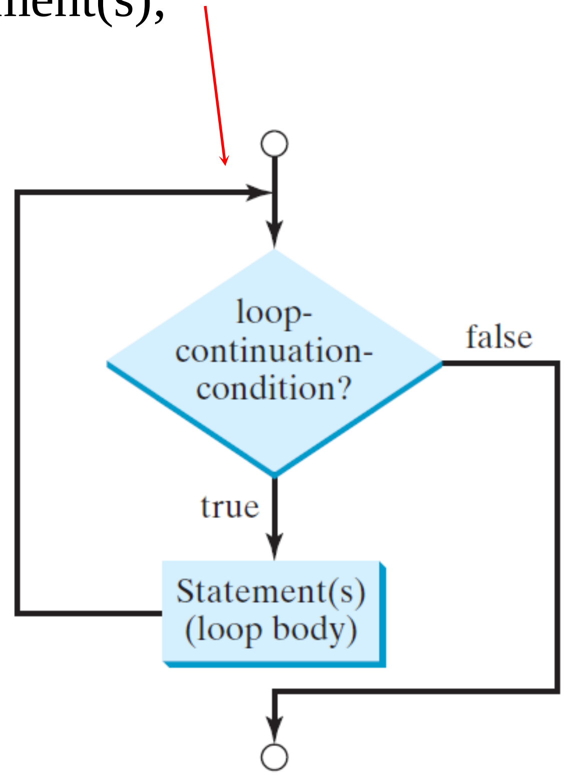
```
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");


…

…

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

Objectives

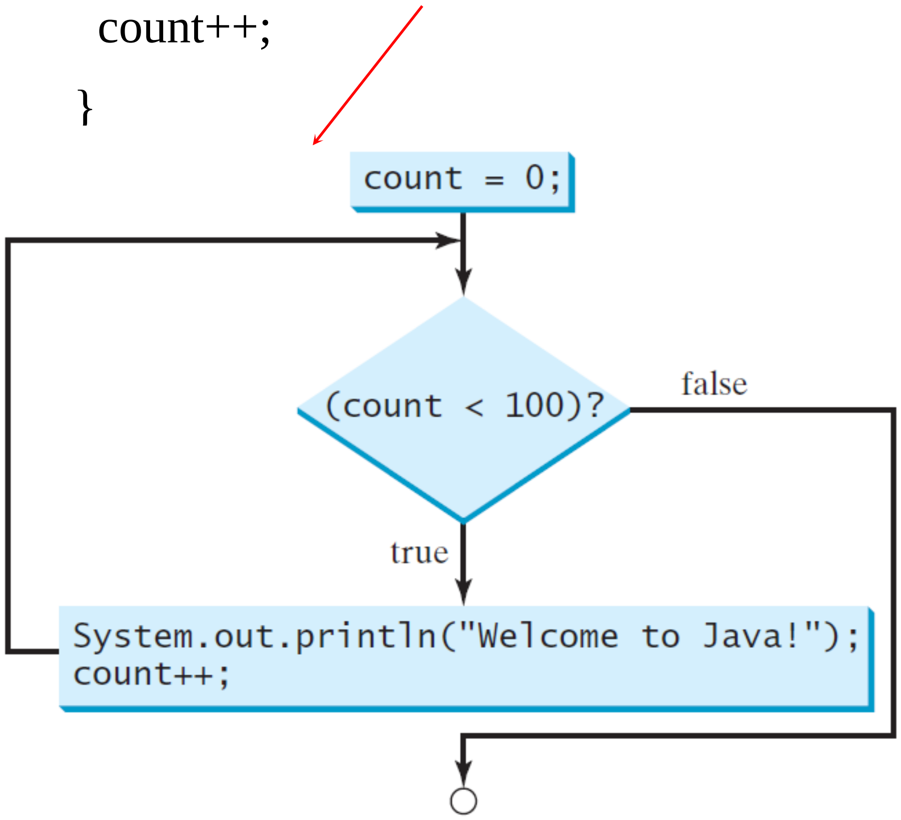
- To write programs for executing statements repeatedly using a **while** loop .
- To follow the loop design strategy to develop loops.
- To control a loop with a sentinel value.
- To write loops using **do-while** statements.
- To write loops using **for** statements.
- To discover the similarities and differences of three types of loop statements.
- To write nested loops.
- To learn loops from a variety of examples.
- To implement program control with **break** and **continue**.

`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++;
}
```

animation

Trace while Loop, cont.

(count < 2) is true

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

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.

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

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

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.

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.

Problem: Repeat Addition Until Correct

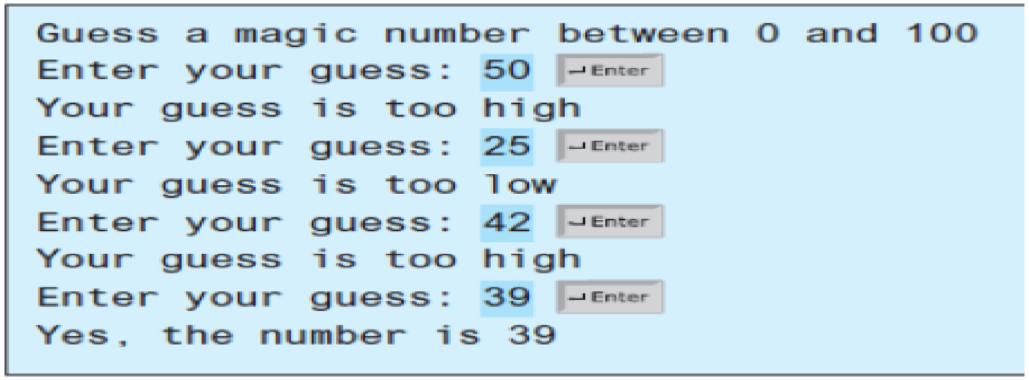
LISTING 5.1 RepeatAdditionQuiz.java

```java
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!");
  }
}
```

```
What is 5 + 9? 12 ↵Enter
Wrong answer. Try again. What is 5 + 9? 34 ↵Enter
Wrong answer. Try again. What is 5 + 9? 14 ↵Enter
You got it!
```

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.



```
Guess a magic number between 0 and 100
Enter your guess: 50 ↵Enter
Your guess is too high
Enter your guess: 25 ↵Enter
Your guess is too low
Enter your guess: 42 ↵Enter
Your guess is too high
Enter your guess: 39 ↵Enter
Yes, the number is 39
```

Problem: Guessing Numbers

LISTING 5.3 GuessNumber.java

```java
import java.util.Scanner;

public class GuessNumber {
  public static void main(String[] args) {
    // Generate a random number to be guessed
    int number = (int)(Math.random() * 101);

    Scanner input = new Scanner(System.in);
    System.out.println("Guess a magic number between 0 and 100");

    int guess = -1;
    while (guess != number) {
      // Prompt the user to guess the number
      System.out.print("\nEnter your guess: ");
      guess = input.nextInt();

      if (guess == number)
        System.out.println("Yes, the number is " + number);
      else if (guess > number)
        System.out.println("Your guess is too high");
      else
        System.out.println("Your guess is too low");
    } // End of 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*.

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

Ending a Loop with a Sentinel Value

LISTING 5.5 SentinelValue.java

```java
import java.util.Scanner;

public class SentinelValue {
  /** Main method */
  public static void main(String[] args) {
    // Create a Scanner
    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();
    }

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

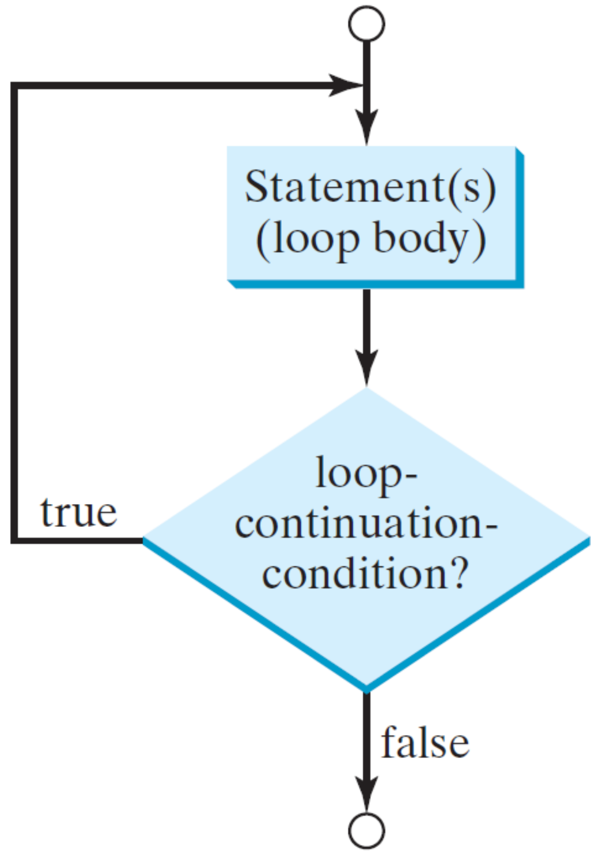
```
Enter an integer (the input ends if it is 0): 2
Enter an integer (the input ends if it is 0): 3
Enter an integer (the input ends if it is 0): 4
Enter an integer (the input ends if it is 0): 0
The sum is 9
```

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:

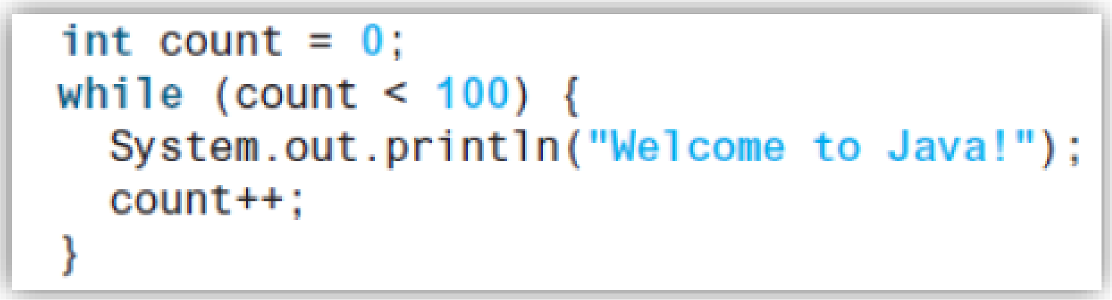
```
double item = 1; double sum = 0;
while (item != 0) { // No guarantee item will be 0
  sum += item;
  item -= 0.1;
}
System.out.println(sum);
```

do-while Loop



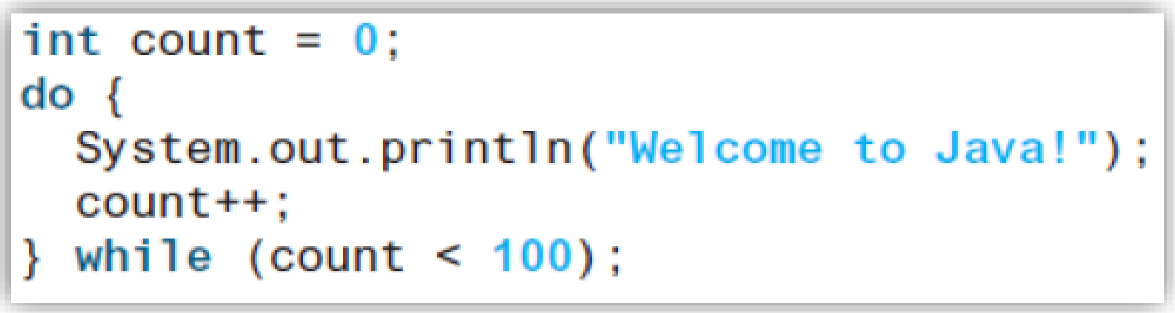
```
do {
   // Loop body;
   Statement(s);
} while (loop-continuation-condition);
```

`do-while` Loop



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

- ❑ The difference between a **while** loop and a **do-while** loop is the order in which the **loop-continuation-condition** is evaluated and the loop body is executed.



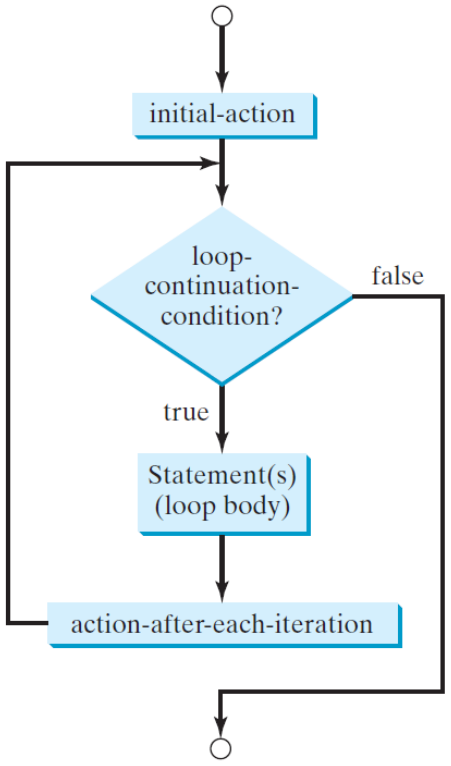
```
int count = 0;
do {
  System.out.println("Welcome to Java!");
  count++;
} while (count < 100);
```

- ❑ In the case of a **do-while** loop, the loop body is executed at least once.

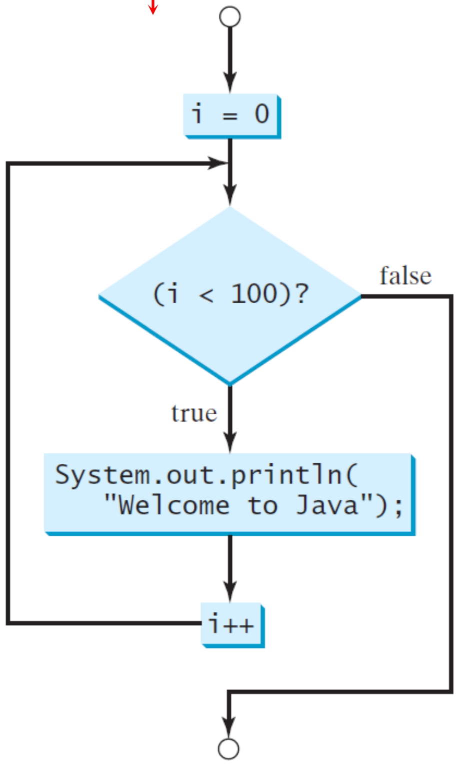
`for` Loops

```
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!");
}
```



(a)



(b)

animation

Trace for Loop

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

Declare i

animation

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.

(i < 2) is true
since i is 0

```
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 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!");
}
```

animation

Trace for Loop, cont.

(i < 2) is false
since i is 2

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

animation

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 :

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

(a)

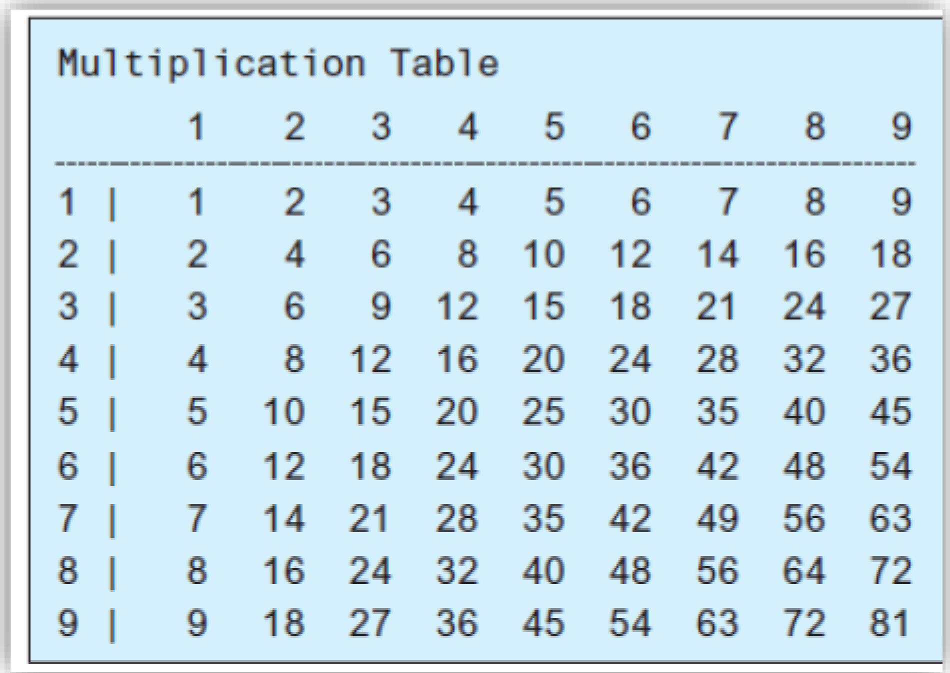
Equivalent

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

(b)

Nested Loops

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



```
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 |     4    8   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
```

Nested Loops

LISTING 5.7 MultiplicationTable.java

```java
public class MultiplicationTable {
  /** Main method */
  public static void main(String[] args) {
    // Display the table heading
    System.out.println("           Multiplication Table");

    // Display the number title
    System.out.print("    ");
    for (int j = 1; j <= 9; j++)
      System.out.print("   " + j);


     System.out.println("\n -----------------------------------");

    // Display table body
    for (int i = 1; i <= 9; i++) {
      System.out.print(i + " | ");
      for (int j = 1; j <= 9; j++) {
        // Display the product and align properly
        System.out.printf("%4d", i * j);
      }
      System.out.println();
    }
  }
}
```

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);
  }
}
```

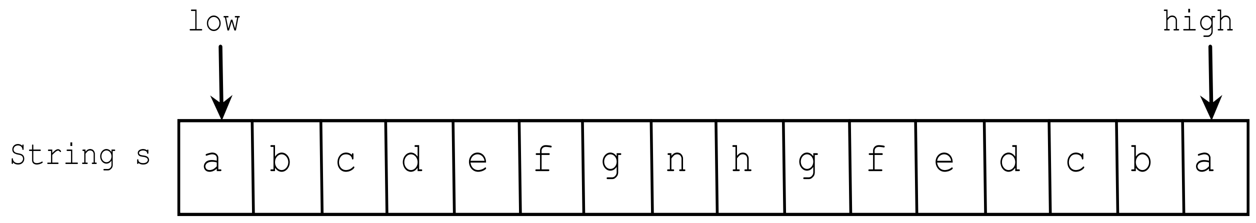
continue

```
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);
  }
}
```

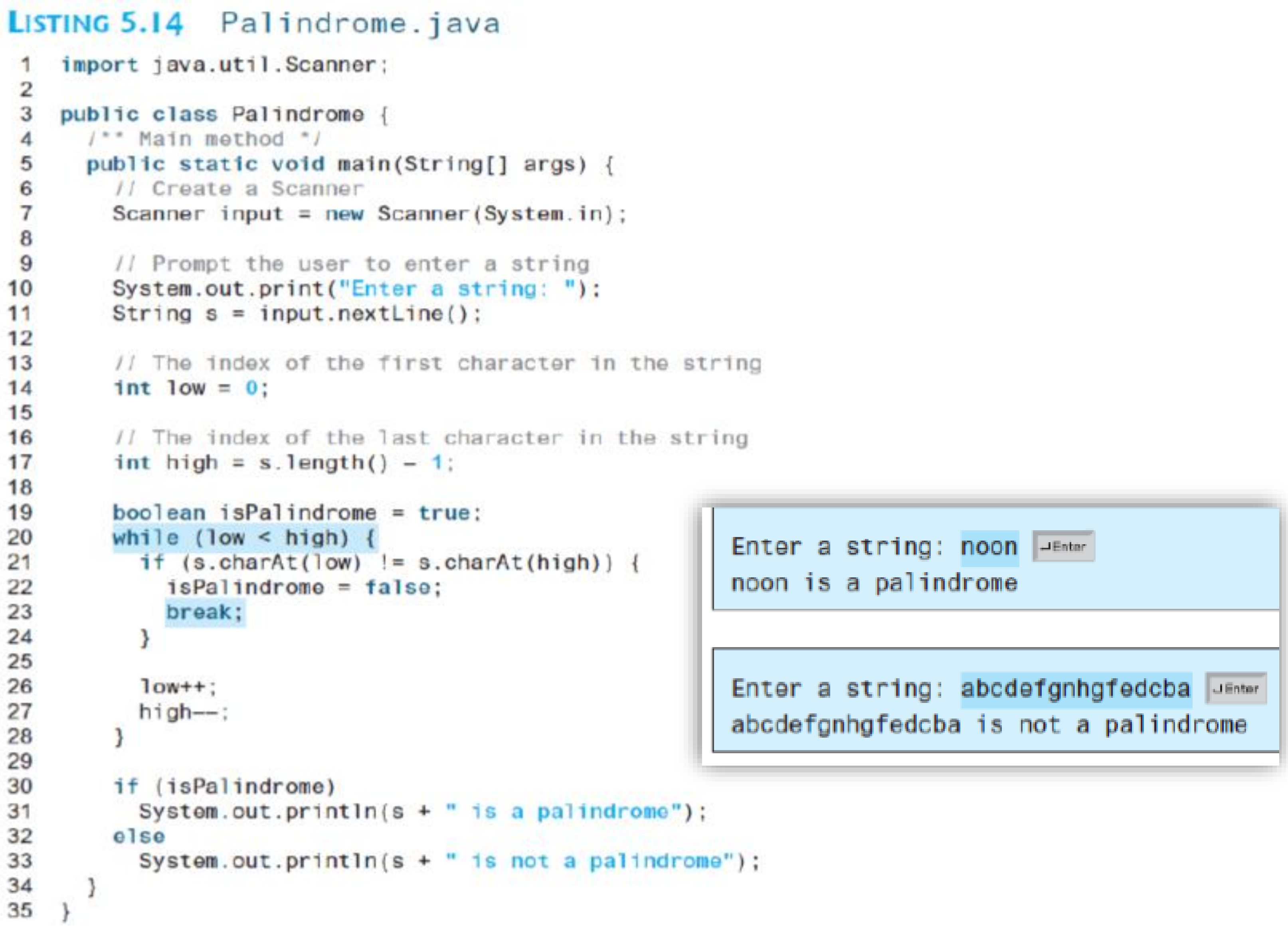
Problem: Checking Palindrome

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

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.



Problem: Checking Palindrome



LISTING 5.14 Palindrome.java

```java
import java.util.Scanner;

public class Palindrome {
  /** Main method */
  public static void main(String[] args) {
    // Create a Scanner
    Scanner input = new Scanner(System.in);

    // 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--;
    }

    if (isPalindrome)
      System.out.println(s + " is a palindrome");
    else
      System.out.println(s + " is not a palindrome");
  }
}
```

```
Enter a string: noon ↵Enter
noon is a palindrome
```

```
Enter a string: abcdefgnhgfedcba ↵Enter
abcdefgnhgfedcba is not a palindrome
```