Chapter Goals

- To implement \textit{while}, \textit{for}, and \textit{do} loops
- To hand-trace the execution of a program
- To become familiar with common loop algorithms
- To understand nested loops
- To implement programs that read and process data sets
- To use a computer for simulations

In this chapter, you will learn about loop statements in Java, as well as techniques for writing programs that simulate activities in the real world.

Contents

- The \texttt{while} loop
- Problem Solving: Hand-Tracing
- The \texttt{for} loop
- The \texttt{do} loop
- Application: Processing Sentinels
- Problem Solving: Storyboards
- Common Loop Algorithms
- Nested Loops
- Application: Random Numbers and Simulations

4.1 Code Reuse: \texttt{while} Loop

- A \texttt{while} statement executes a block of code repeatedly
- A condition controls how long the loop is executed

\begin{verbatim}
while (condition) statement
\end{verbatim}

- Most commonly, the statement is a block statement (set of statements delimited by \{ \})
Calculating the Growth of an Investment

How many years does it take for the bank account to reach a particular balance?

while (balance < TARGET) {
    year++;
    double interest = balance * RATE/100;
    balance = balance + interest;
}

What is the minimum number of times the block of statements is executed?

Execution of the Loop

- Initial balance: 10,000
- Rate: 5%
- Target: 20,000

<table>
<thead>
<tr>
<th>Year</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$10,000</td>
</tr>
<tr>
<td>1</td>
<td>$10,500</td>
</tr>
<tr>
<td>2</td>
<td>$11,025</td>
</tr>
<tr>
<td>3</td>
<td>$11,576.25</td>
</tr>
<tr>
<td>4</td>
<td>$12,155.06</td>
</tr>
<tr>
<td>5</td>
<td>$12,762.82</td>
</tr>
</tbody>
</table>

Execution of the loop - cont

- After executing the Loop for 15 times:
  - balance = 20,789.28
  - which is > TARGET
  - The while loop terminates.

DoubleInvestment.java

```java
public class DoubleInvestment {
    public static void main(String[] args) {
        final double RATE = 5;
        final double INITIAL_BALANCE = 10000;
        final double TARGET = 2 * INITIAL_BALANCE;
        double balance = INITIAL_BALANCE;
        int year = 0;

        // Count the years required for the investment to double
        while (balance < TARGET) {
            year++;
            double interest = balance * RATE / 100;
            balance = balance + interest;
            System.out.print("The investment doubled after ");
            System.out.print("year = "+year);
        }

        // The investment doubled after 15 years.
    }
}
```

Program Run

The investment doubled after 15 years.
while Loop Examples (1)

Common Error: Infinite Loop

- Forgot to update the variable value checked in the while condition statement.
- What are the outputs of the following while statements?

```java
int years = 0;
while (years < 20)
{
    System.out.println(years);
}
```

```java
int year = 0;
while (year < 20)
{
    System.out.println(year);
    year++;
}
```

while Loop Examples (2)

Common Error: Infinite Loop

- Update the variable value checked in the while condition statement wrongly.
- What are the outputs of the following while statements?

```java
int years = 20;
while (years < 20)
{
    System.out.println(years);
    years++;
}
```

```java
int years = 15;
while (years != 0)
{
    years = years - 2;
    System.out.println(years);
}
```
Common Error: Semicolon

- Extra ; at the end of the condition statement.
- The extra ; makes the while statement having an empty body.
- What are the outputs?

```java
int years = 20;
while (years > 0);
{
  System.out.println (years);
  years--;
}
```

```java
int years = 20;
while (years-- > 0);
{
  System.out.println (years);
  years--;
}
```

Common Error: off-by-one

- Off-by-One Errors: a loop executes one time too few, or one time too many.
- **Start at 0, use <**
- **Start at 1, use <=**

```java
int finger = 0;
final int FINGERS = 5;
while (finger < FINGERS)
{
  // paint finger
  finger++;
}
```

```java
int finger = 1;
final int FINGERS = 5;
while (finger <= FINGERS)
{
  // paint finger
  finger++;
}
```

4.2: Hand-Tracing

- Example: Calculate the sum of digits (1+7+2+9)
  - Make columns for key variables (n, sum, digit)
  - Examine the code and number the steps
  - Set variables to state before loop begins

<table>
<thead>
<tr>
<th>n</th>
<th>sum</th>
<th>digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1729</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

```java
int n = 1729;
int sum = 0;
while (n > 0)
{
  int digit = n % 10;
  sum = sum + digit;
  n = n / 10;
}
System.out.println(sum);
```

Tracing Sum of Digits

- Start executing loop body statements changing variable values on a new line
- Cross out values in previous line

<table>
<thead>
<tr>
<th>n</th>
<th>sum</th>
<th>digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1729</td>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>
```java
int n = 1729;
int sum = 0;
while (n > 0)
{
  int digit = n % 10;
  sum = sum + digit;
  n = n / 10;
}
System.out.println(sum);
```
Tracing Sum of Digits

<table>
<thead>
<tr>
<th>n</th>
<th>sum</th>
<th>digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1729</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>172</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>18</td>
<td>7</td>
</tr>
</tbody>
</table>

- Continue executing loop statements changing variables
  - 1729 / 10 leaves 172 (no remainder)

Tracing Sum of Digits

<table>
<thead>
<tr>
<th>n</th>
<th>sum</th>
<th>digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1729</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>172</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>18</td>
<td>7</td>
</tr>
</tbody>
</table>

- Test condition. If true, execute loop again
  - Variable n is 172, is 172 > 0?, True!
  - Make a new line for the second time through and update variables

Tracing Sum of Digits

<table>
<thead>
<tr>
<th>n</th>
<th>sum</th>
<th>digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1729</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>172</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>18</td>
<td>7</td>
</tr>
</tbody>
</table>

- Third time through
  - Variable n is 17 which is still greater than 0
- Execute loop statements and update variables

Tracing Sum of Digits

<table>
<thead>
<tr>
<th>n</th>
<th>sum</th>
<th>digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1729</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>172</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>18</td>
<td>7</td>
</tr>
</tbody>
</table>

- Fourth loop iteration:
  - Variable n is 1 at start of loop. 1 > 0? True
  - Executes loop and changes variable n to 0 (1/10 = 0)
Tracing Sum of Digits

<table>
<thead>
<tr>
<th>n</th>
<th>sum</th>
<th>digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1729</td>
<td>0'</td>
<td></td>
</tr>
<tr>
<td>172</td>
<td>9'</td>
<td>9'</td>
</tr>
<tr>
<td>17</td>
<td>1'</td>
<td>2'</td>
</tr>
<tr>
<td>X</td>
<td>1'</td>
<td>7'</td>
</tr>
<tr>
<td>0</td>
<td>19</td>
<td>1'</td>
</tr>
</tbody>
</table>

- Because n is 0, the expression \( n > 0 \) is False
- Loop body is not executed
  - Jumps to next statement after the loop body
- Finally prints the sum!

Summary of the while Loop

- **while** loops are very commonly used for code reuse.
  - Initialize variables before testing them in the condition
  - The condition is tested BEFORE the loop body
  - Something inside the loop should change one of the condition variables used in the test
  - The variable values should change toward the terminating condition
- Watch out for infinite loops!

Review: while Statement

- A for loop can do everything a while loop can do
  - The for loop is more concise in that it puts
    - Initialization
    - Condition test
    - Update
    - All on one line!
When to use a for Loop?

- **Use a for loop when**
  - The control variable has a **fixed starting and ending value**.
  - The control variable has a **constant increment (or decrement)** of.

In general, the for loop:
```
for (initialization; condition; update)
{  statements
}
```

has exactly the same effect as the while loop:
```
while (condition)
{  statements
    update
}
```

A for Loop Example

- Print the balance at the end of each year for a number of years:

```
for (int year = 1; year <= nyears; year++)
{
    double interest = balance * rate / 100;
    balance = balance + interest;
}
```

year is a local variable belongs to the loop body

- how many times the block of statements is executed?
- Under what condition the block of statements would never be executed?

Execution of a for Loop

1. **Initialize counter**
   ```java
   int counter = 1;
   System.out.println(counter);
   ```

2. **Check condition**
   ```java
   counter = 1;
   System.out.println(counter);
   ```

3. **Execute loop body**
   ```java
   counter = 1;
   System.out.println(counter);
   ```

4. **Update counter**
   ```java
   counter = 2;
   System.out.println(counter);
   ```

Syntax 4.2: for Statement

- Two semicolons separate the three parts
  - Initialization; Condition; Update

Three expressions should be related. See page 195.

This initialization happens once before the loop starts. The condition is checked before each iteration.

This update is executed after each iteration.

The variable 1 is defined only in this for loop. See page 195.
for Loops examples
Avoid using != to evaluate condition

<table>
<thead>
<tr>
<th>Loop</th>
<th>Values of i</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>for (i = 0; i &lt;= 5; i++)</td>
<td>0 1 2 3 4 5</td>
<td>Note that the loop is executed 6 times. (See Programming Tip 4.4 on page 153.)</td>
</tr>
<tr>
<td>for (i = 5; i &gt;= 0; i--)</td>
<td>5 4 3 2 1 0</td>
<td>Use -- for decreasing values.</td>
</tr>
<tr>
<td>for (i = 0; i &lt; s.length(); i++)</td>
<td>0 1 2 3 4 5 6 7 8</td>
<td>Use s[i] for accessing elements in the string.</td>
</tr>
<tr>
<td>for (i = 0; i &lt; s.length(); i++)</td>
<td>0 1 2</td>
<td>In the loop body, use the expression s.charAt(i) to get the ith character.</td>
</tr>
</tbody>
</table>

Remember str is indexed from 0 to length()-1, so make sure i is initialized to 0.

Common Errors: Semicolons

- Are the following two statements the same?
  ```java
  for (int years = 1; (balance = balance + balance * rate / 100) < targetBalance; years++) // empty body.
  
  for (int years = 1; balance < targetBalance; years++)
  {
    double interest = balance * rate / 100;
    balance = balance + interest;
  }
  ```

- A semicolon that shouldn’t be there:
  ```java
  int = 0;
  for (i = 1; i <= 10; i++) // remove ; to include the next statement
  sum = sum + i; // as the body of the for-loop
  System.out.println(sum); // what is the output?
  ```

for Loop variable Scope

```java
for( int x = 1; x < 10; x = x + 1 )
{
  // steps to do inside the loop
  // You can use 'x' anywhere in this box
  
  if (x > 100) // Error! x is out of scope!
  }
``` 

- Scope is the ‘lifetime’ of a variable.
- When x is declared in the for statement, x exists only inside the ‘block’ of the for loop.
- Solution: Declare x outside the for loop

```java
int x;
for(x = 1; x < 10; x = x + 1)
``` 

4.4 The do Loop

- Executes loop body at least once:
  ```java
  do
  { statement
  } while (condition);
  ```

- Example: Validate input,

```java
int value;
do
{
  System.out.println("Enter an integer < 100: ");
  value = in.nextInt();
} 
while (value >= 100); // test
```
### Summary of loop structures

- **while (conditions){statements;}**
  - Check condition before executing statements;
  - The number of times the statements are executed: Min: 0; max: ∞

- **do{statements;} while(conditions);**
  - Execute statements before checking conditions
  - The number of times, statements are executed: Min: 1; max: ∞

- **for(i=a; conditions; i++) {statements;}**
  - Initialize; check conditions; execute statements;
  - The number of times, statements are executed: Min: 0; max: ∞

### Converting Loops

```java
// double s; int x = 1;
// do{
s = 1.0/x;
x++; // update
} while (s > 0.01);// test

// double s; int x = 1;
s = 1.0/x;
x++;
while (s > 0.01) // test
{
s = 1.0/x;
x++; // update
}
```

### 4.7 Common Loop Algorithms

1: Computing Sum and Average
2: Computing Max and Min
3: Counting the Matches
4: Finding the First Match
5: Prompting until a match is found
6: Comparing Adjacent Values
Computing Sum and Average

- **Computing Sum**
  - Initialize total to 0
  - Use hasNextDouble method

```java
double total = 0;
while (in.hasNextDouble())
{
    double input = in.nextDouble();
    total = total + input;
}
```

- **Computing Average**
  - Initialize total and count to 0;
  - Computing Sum and count
  - Initialize average (is it necessary?)
  - Check count is not 0 before divide!

```java
double total = 0;
int count = 0;
while (in.hasNextDouble())
{
    double input = in.nextDouble();
    total = total + input;
    count++;
}
double average = 0;
if (count > 0)
{
    average = total / count;
}
```

Computing Max and Min

- **Computing Max**
  - Read the first input value as the largest (or smallest) that you have seen so far!

```java
double largest = in.nextDouble();
while (in.hasNextDouble())
{
    double input = in.nextDouble();
    if (input > largest)
    {
        largest = input;
    }
}
```

- **Computing Min**
  - Update largest (or smallest) if new input is larger (or smaller) than the largest (or smallest)

```java
double smallest = in.nextDouble();
while (in.hasNextDouble())
{
    double input = in.nextDouble();
    if (input < smallest)
    {
        smallest = input;
    }
}
```

Counting the Matches

- Counting how many uppercase letters in a string:
- Index starts from 0 and ends at str.length() – 1;
- Example: String str="Hello";

```java
int upperCaseLetters = 0;
for (int i = 0; i < str.length(); i++)
{
    char ch = str.charAt(i);
    if (Character.isUpperCase(ch))
    {
        upperCaseLetters++;
    }
}
```

Finding the First Match

- Find the first lowercase letter in a string;
- Once found, no need to loop through the entire String.

```java
boolean found = false;
char ch = '?' ; // why?
int position = 0;
while (!found &&
{
    ch = str.charAt(position);
    if (Character.isLowerCase(ch))
    {
        found = true;
    }
    else { position++;
    // update position only when not found
}
```
Prompt Until a Match is Found

- Keep asking the user to enter a positive value < 100 until the user provides a correct input.
- What is the minimum number of prompt?

```java
boolean valid = false;
double input;
while (!valid) {
    System.out.print("Please enter a positive value < 100: ");
    input = in.nextDouble();
    if (0 < input && input < 100) {
        valid = true;
    } else {
        System.out.println("Invalid input.");
    }
}
```

Comparing Adjacent Values

- Check whether a sequence of inputs contains adjacent duplicates such as 1 7 2 9 9 4 9:

```java
double input = in.nextDouble();
//read the first input outside the loop
while (in.hasNextDouble()) {
    double previous = input;
    //store the current value as the previous
    input = nextDouble();
    //read the next input
    if (input == previous) {
        System.out.println("Duplicate input");
    }
}
```

4.8 Nested Loops

- When the body of a loop contains another loop, the loops are nested.
- A typical use of nested loops is to process two-dimensional structures, such as tables.

```java
final int ROWS = 4;
final int COLUMNS = 4;
int value = 1;
for (int row = 1; row <= ROWS; row++) {
    for (int col = 1; col <= COLUMNS; col++) {
        System.out.print(value + ",");
        value++;
    }
    System.out.println();
}
```

Table.java

```java
final int ROWS = 4;
final int COLUMNS = 4;
int value = 1;
for (int row = 1; row <= ROWS; row++) {
    for (int col = 1; col <= COLUMNS; col++) {
        System.out.print(value + ",");
        value++;
    }
    System.out.println();
}
```
Flowchart of a Nested Loop

Outer Loop

Inner Loop

row = 1
row <= 4?
True
col= 1
col <= 4?
True
Print value
value++
False
Print new line
col++
row++

Done

False

Inner Loop Examples

for (i = 1; i <= 4; i++)
{
    for (j = 1; j <= 4; j++)
    {
        print "*"
        print new line
    }
}

Prints 3 rows of 4 asterisks each.

for (i = 1; i <= 4; i++)
{
    for (j = 1; j <= 3; j++)
    {
        print "*"  
        print new line
    }
}

Prints 4 rows of 3 asterisks each.

for (i = 1; i <= 4; i++)
{
    for (j = 1; j <= 2; j++)
    {
        print "*"  
        print new line
    }
}

Prints 4 rows of lengths 1, 2, 3, and 4.

More Nested Loop Examples

for (i = 1; i <= 3; i++)
{
    for (j = 1; j <= 5; j++)
    {
        if (j % 2 == 0) { print "+" }
        else { print "-" }
        System.out.println();
    }
}

Prints asterisks in even columns, dashes in odd columns.

for (i = 1; i <= 3; i++)
{
    for (j = 1; j <= 5; j++)
    {
        if (j % 2 == 0) { print "*" }
        else { print "*" }
        System.out.println();
    }
}

Prints a checkerboard pattern.

Exercises

final int ROWS=4;
final int COLUMNS=4;

for (int row=ROWS; row >=1; row--)
{
    for (int col=COLUMNS; col >=1; col--)
    {
        System.out.print(row","+col);

        System.out.println();
    }
}

This is in your Quiz next Monday

4,4 4,3 4,2 4,1
4,3 4,2 4,1 4,0
3,4 3,3 3,2 3,1
3,3 3,2 3,1 3,0
2,4 2,3 2,2 2,1
2,3 2,2 2,1 2,0
1,4 1,3 1,2 1,1
1,3 1,2 1,1 1,0
1,2 1,1 1,0 1,0
0,4 0,3 0,2 0,1
0,3 0,2 0,1 0,0
Converting Double-Loop

```java
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
        System.out.println(i + “,” + j);

int i = 0;
while (i < 3){
    int j = 0;
    while (j < 3){
        System.out.println(i + “,” + j);
        j++;
    }
    i++;
}
```

Watch out for initialization and updating of loop counter when converting:

Do-while loop

```java
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
        System.out.println(i + “,” + j);

int i = 0;
do{
    j = 0;
do{
        System.out.println(i + “,” + j);
        j = j++;
    }while (j < 3);
    i++;
}while (i < 3);
```

Watch out for initialization and updating of loop counter when converting:

Summary: Types of Loops

- There are three types of loops:
  - while Loops
  - for Loops
  - do Loops
- Each loop requires the following steps:
  - Initialization (setup variables to start looping)
  - Condition (test if we should execute loop body)
  - Update (change something each time through)

Summary

- A loop executes instructions repeatedly while a condition is true.
- An off-by-one error is a common error when programming loops.
  - Think through simple test cases to avoid this type of error.
- The for loop is used when a value runs from a starting point to an ending point with a constant increment or decrement.
- The do loop is appropriate when the loop body must be executed at least once.
Summary

- You can use a boolean variable to control a loop.
  - Set the variable to true before entering the loop, then
    set it to false to leave the loop.
- When the body of a loop contains another loop, the loops are nested.
  - A typical use of nested loops is printing a table with rows and columns.