Chapter Six: Iteration
Chapter Goals

• To be able to program loops with the `while`, `for`, and `do` statements
• To avoid infinite loops and off-by-one errors
• To understand nested loops
• To learn how to process input
• To implement simulations
while Loops

• Executes a block of code repeatedly
• A condition controls how often the loop is executed

    while (condition)
    
    statement

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

- Invest $10,000, 5% interest, compounded annually

<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>
Calculating the Growth of an Investment

• When has the bank account reached a particular balance?

    while (balance < targetBalance)
    {
        years++;  
        double interest = balance * rate / 100;  
        balance = balance + interest;
    }
A class to monitor the growth of an investment that accumulates interest at a fixed annual rate.

```java
public class Investment {
    
    Constructs an Investment object from a starting balance and interest rate.

    @param aBalance the starting balance
    @param aRate the interest rate in percent

    public Investment(double aBalance, double aRate) {
        balance = aBalance;
        rate = aRate;
        years = 0;
    }

    Keeps accumulating interest until a target balance has been reached.

    @param targetBalance the desired balance

    */
```
public void waitForBalance(double targetBalance) {
    while (balance < targetBalance) {
        years++;
        double interest = balance * rate / 100;
        balance = balance + interest;
    }
}

/**
 * Gets the current investment balance.
 * @return the current balance
 */
public double getBalance() {
    return balance;
}

/**
 * Gets the number of years this investment has accumulated interest.
 */
@return the number of years since the start of the investment
*/
    public int getYears()
    {
        return years;
    }

private double balance;
private double rate;
private int years;
This program computes how long it takes for an investment to double.

public class InvestmentRunner
{
    public static void main(String[] args)
    {
        final double INITIAL_BALANCE = 10000;
        final double RATE = 5;
        Investment invest = new Investment(INITIAL_BALANCE, RATE);
        invest.waitForBalance(2 * INITIAL_BALANCE);
        int years = invest.getYears();
        System.out.println("The investment doubled after "
            + years + " years");
    }
}
Output:
The investment doubled after 15 years
while Loop Flowchart

Figure 1 Flowchart of a while Loop
Syntax 6.1 The `while` Statement

```java
while (condition)
    statement
```

**Example:**

```java
while (balance < targetBalance)
{
    years++; 
    double interest = balance * rate / 100;
    balance = balance + interest;
}
```

**Purpose:**

To repeatedly execute a statement as long as a condition is true.
Self Check 6.1

How often is the statement in the loop

while (false) statement;

executed?
Self Check 6.2

What would happen if RATE was set to 0 in the main method of the InvestmentRunner program?
Common Error: Infinite Loops

- int years = 0;
  while (years < 20)
  {
    double interest = balance * rate / 100;
    balance = balance + interest;
  }

- int years = 20;
  while (years > 0)
  {
    years++; // Oops, should have been years--
    double interest = balance * rate / 100;
    balance = balance + interest;
  }

- Loops run forever – must kill program
Common Error: Off-by-One Errors

- int years = 0;
  while (balance < 2 * initialBalance) {
    years++;
    double interest = balance * rate / 100;
    balance = balance + interest;
  }
  System.out.println("The investment reached the target after " + years + " years.");

Should years start at 0 or 1?

Should the test be < or <=?
Avoiding Off-by-One Error

• Look at a scenario with simple values:
  initial balance: $100
  interest rate: 50%
  after year 1, the balance is $150
  after year 2 it is $225, or over $200
  so the investment doubled after 2 years
  the loop executed two times, incrementing years each time
  Therefore: years must start at 0, not at 1.

• interest rate: 100%
  after one year: balance is 2 * initialBalance
  loop should stop
  Therefore: must use <

• Think, don't compile and try at random
do Loops

• Executes loop body at least once:
  do
    statement
  while (condition);

• Example: Validate input
  double value;
  do
    {
      System.out.print("Please enter a positive number: ");
      value = in.nextDouble();
    }
  while (value <= 0);

Continued
Alternative:

```java
boolean done = false;
while (!done)
{
    System.out.print("Please enter a positive number: ");
    value = in.nextDouble();
    if (value > 0) done = true;
}
```
do Loop Flowchart

Flowchart of a do Loop
for **Loops**

• for (initialization; condition; update)
  
  statement

• Example:
  for (int i = 1; i <= n; i++)
  {
    double interest = balance * rate / 100;
    balance = balance + interest;
  }

• Equivalent to
  initialization;
  while (condition)
  {
    statement;
    update;  }

*Continued*
Other examples:

```java
for (years = n; years > 0; years--)
```

```java
for (x = -10; x <= 10; x = x + 0.5)
```
**Figure 2** Flowchart of a for Loop
Syntax 6.2 The for Statement

for (initialization; condition; update)
    statement

Example:

for (int i = 1; i <= n; i++)
{ 
    double interest = balance * rate / 100;
    balance = balance + interest;
}

Purpose:

To execute an initialization, then keep executing a statement and updating an expression while a condition is true.
A class to monitor the growth of an investment that accumulates interest at a fixed annual rate

```java
public class Investment {
    /**
     * Constructs an Investment object from a starting balance and interest rate.
     * @param aBalance the starting balance
     * @param aRate the interest rate in percent
     */
    public Investment(double aBalance, double aRate) {
        balance = aBalance;
        rate = aRate;
        years = 0;
    }
    /**
     * Keeps accumulating interest until a target balance has been reached.
     */
}  
```
@param targetBalance the desired balance

*/

{
while (balance < targetBalance)
{
    years++;
    double interest = balance * rate / 100;
    balance = balance + interest;
}

/**
* Keeps accumulating interest for a given number of years.
* @param n the number of years
*/
public void waitYears(int n)
{
    for (int i = 1; i <= n; i++)
    {
        double interest = balance * rate / 100;
        balance = balance + interest;
    }
years = years + n;
}

/**
 * Gets the current investment balance.
 * @return the current balance
 */
public double getBalance()
{
    return balance;
}

/**
 * Gets the number of years this investment has accumulated interest.
 * @return the number of years since the start of the investment
 */
public int getYears()
{
    return years;
}
ch06/invest2/Investment.java (cont.)

67:  
68:     private double balance;
69:     private double rate;
70:     private int years;
71:  }
This program computes how much an investment grows in a given number of years.

```java
public class InvestmentRunner {
    public static void main(String[] args) {
        final double INITIAL_BALANCE = 10000;
        final double RATE = 5;
        final int YEARS = 20;
        Investment invest = new Investment(INITIAL_BALANCE, RATE);
        invest.waitYears(YEARS);
        double balance = invest.getBalance();
        System.out.printf("The balance after %d years is %.2f\n", YEARS, balance);
    }
}
```

Output:
The balance after 20 years is 26532.98
How many times does the following for loop execute?
for (i = 0; i <= 10; i++)
    System.out.println(i * i);
Common Errors: Semicolons

- A semicolon that shouldn't be there
  
  ```java
  sum = 0;
  for (i = 1; i <= 10; i++);
      sum = sum + i;
  System.out.println(sum);
  ```
Nested Loops

• Create triangle pattern
  
  []
  [[]]
  [[][]]
  [[][[[]]]]

• Loop through rows
  for (int i = 1; i <= n; i++)
  {
    // make triangle row
  }

  *Make triangle row* is another loop
  for (int j = 1; j <= i; j++)
    r = r + "[]";
  r = r + "\n";

• Put loops together → Nested loops
This class describes triangle objects that can be displayed as shapes like this:

```
[]
[][]
[][][]
```

```
01: /**
02:     This class describes triangle objects that can be displayed
03:     as shapes like this:
04:     []
05:     [][]
06:     [][][]
07: */
08: public class Triangle
09: {
10:     /**
11:         Constructs a triangle.
12:         @param aWidth the number of [] in the last row of the triangle.
13:     */
14:     public Triangle(int aWidth)
15:     {
16:         width = aWidth;
17:     }
18: }
19: /**
20:     Computes a string representing the triangle.
21:     @return a string consisting of [] and newline characters
22: */
23:     public String toString()
24:     {
25:         String r = "";
26:         for (int i = 1; i <= width; i++)
27:             {
28:                 // Make triangle row
29:                 for (int j = 1; j <= i; j++)
30:                     r = r + "[]";
31:                     r = r + "\n";
32:             }
33:         return r;
34:     }
35:     }
36:     private int width;
This program prints two triangles.

```java
public class TriangleRunner {
    public static void main(String[] args) {
        public static void main(String[] args) {
            Triangle small = new Triangle(3);
            System.out.println(small.toString());

            Triangle large = new Triangle(15);
            System.out.println(large.toString());
        }
    }
}
```
File TriangleRunner.java (cont.)

Output:

```
[ ]
[ ] [ ]
[ ] [ ] [ ]
[ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
```

Continued
File TriangleRunner.java  (cont.)

Output (continued):

```
[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
```

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.
Self Check 6.5

How would you modify the nested loops so that you print a square instead of a triangle?
Loop and a half

- Sometimes termination condition of a loop can only be evaluated in the middle of the loop

- Then, introduce a boolean variable to control the loop:

  ```java
  boolean done = false;
  while (!done) {
    Print prompt
    String input = read input;
    if (end of input indicated)
      done = true;
    else
    {
      Process input
    }
  }
  ```
import java.util.Scanner;

/**
 * This program computes the average and maximum of a set of input values.
 */

public class DataAnalyzer {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        DataSet data = new DataSet();

        boolean done = false;
        while (!done) {
            System.out.print("Enter value, Q to quit: ");
            String input = in.next();
            if (input.equalsIgnoreCase("Q"))
                done = true;

            // parse input and add to data set
            // ... (continued)
        }
    }
}

Continued
else
{
    double x = Double.parseDouble(input);
    data.add(x);
}

System.out.println("Average = " + data.getAverage());
System.out.println("Maximum = " + data.getMaximum());
Computes the average of a set of data values.

```
public class DataSet {
    /**
     * Constructs an empty data set.
     */
    public DataSet() {
        sum = 0;
        count = 0;
        maximum = 0;
    }

    /**
     * Adds a data value to the data set
     * @param x a data value
     */
    public void add(double x) {
        Continued
```
Continued

ch06/dataset/DataSet.java (cont.)

```java
22:     sum = sum + x;
23:     if (count == 0 || maximum < x) maximum = x;
24:     count++;
25: }
26:
27: /**
28:     Gets the average of the added data.
29:     @return the average or 0 if no data has been added
30: */
31: public double getAverage()
32: {
33:     if (count == 0) return 0;
34:     else return sum / count;
35: }
36:
37: /**
38:     Gets the largest of the added data.
39:     @return the maximum or 0 if no data has been added
40: */
```
41:   public double getMaximum()
42:   {
43:       return maximum;
44:   }
45:
46:   private double sum;
47:   private double maximum;
48:   private int count;
49: }

Output:
Enter value, Q to quit: 10
Enter value, Q to quit: 0
Enter value, Q to quit: -1
Enter value, Q to quit: Q
Average = 3.0
Maximum = 10.0
Why does the `DataAnalyzer` class call `in.next` and not `in.nextDouble`?

**Answer:** Because we don't know whether the next input is a number or the letter Q.
Self Check 6.8

Would the `DataSet` class still compute the correct maximum if you simplified the update of the maximum field in the `add` method to the following statement?

```java
if (maximum < x) maximum = x;
```
Random Numbers and Simulations

• In a simulation, you repeatedly generate random numbers and use them to simulate an activity

• Random number generator

   Random generator = new Random();
   int n = generator.nextInt(a); // 0 <= n < a
   double x = generator.nextDouble(); // 0 <= x < 1

• Throw die (random number between 1 and 6)

   int d = 1 + generator.nextInt(6);
import java.util.Random;

/**
 * This class models a die that, when cast, lands on a random face.
 */
public class Die {

    /**
     * Constructs a die with a given number of sides.
     * @param s the number of sides, e.g. 6 for a normal die
     */
    public Die(int s) {
        sides = s;
        generator = new Random();
    }

    /**
     * Simulates a throw of the die
     * @return the face of the die
     */
    Continued
23:   public int cast()
24:   {
25:       return 1 + generator.nextInt(sides);
26:   }
27:
28:   private Random generator;
29:   private int sides;
30:   }
This program simulates casting a die ten times.

```java
class DieSimulator {
    public static void main(String[] args) {
        Die d = new Die(6);
        final int TRIES = 10;
        for (int i = 1; i <= TRIES; i++) {
            int n = d.cast();
            System.out.print(n + " ");
        }
        System.out.println();
    }
}
```
Output:
6 5 6 3 2 6 3 4 4 1

Second Run:
3 2 2 1 6 5 3 4 1 2