Chapter Goals

• To be able to program loops with the \texttt{while}, \texttt{for}, and \texttt{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?

```java
while (balance < targetBalance) {
    year++;
    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.
 */

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;
    }

    // Continued...
/**
   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
   */
Continued…
```java
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 InvestmentTester
{
    public static void main(String[] args)
    {
        final double INITIAL_BALANCE = 10000;
        final double RATE = 5;
        Investmen...
The investment doubled after 15 years.
while Loop Flowchart

Figure 1: Flowchart of a while Loop
Syntax 7.1: The **while** Statement

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

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

**Purpose:**
To repeatedly execute a statement as long as a condition is true
Self Check

1. How often is the statement in the loop
   
   ```java
   while (false) statement;
   ```
   
   executed?

2. What would happen if RATE was set to 0 in the main method of the InvestmentTester program?
Answers

1. Never

2. The `waitForBalance` method would never return due to an infinite loop
Common Error: Infinite Loops

- Loops run forever—must kill program

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

```java
int years = 20;
while (years > 0)
{
    years++; // Oops, should have been years--
    double interest = balance * rate / 100;
    balance = balance + interest;
}
```
Common Error: Off-By-One Errors

```java
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

Therefore: years must start at 0, not at 1.

Continued...
Avoiding Off-by-One Error

• interest rate: 100%
  after one year: balance is $2 \times \text{initialBalance}$
  loop should stop
  Therefore: must use $<$

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

- **Executes loop body at least once:**

  ```java
  do
      statement while (condition);
  ```

- **Example: Validate input**

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

*Continued...*
Loops

• Alternative:

```java
boolean done = false;
while (!done)
{
    System.out.print("Please enter a positive number: ");
    value = in.nextDouble();
    if (value > 0) done = true;
}
```
Figure 2: Flowchart of a do Loop
Spaghetti Code

Figure 3: Spaghetti Code
for Loops

- for (initialization; condition; update) statement

Example:

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

Continued...
for Loops

• Equivalent to

  initialization;
  while (condition)
  { statement; update; }

• Other examples:

  for (years = n; years > 0; years--) . . .

  for (x = -10; x <= 10; x = x + 0.5) . . .
Flowchart for **for** Loop

Figure 4: Flowchart of a **for** Loop
Syntax 7.2: 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;
    }

    // Continued...
```
/**
 * 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;
    }
}
/**
 * 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;
}
private double balance;
private double rate;
private int years;
}
/**
 * This program computes how much an investment grows in a given number of years.
 */

public class InvestmentTester {
  public static void main(String[] args) {
    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);
The balance after 20 years is 26532.98
Self Check

1. Rewrite the for loop in the waitYears method as a while loop

2. How many times does the following for loop execute?

```java
for (i = 0; i <= 10; i++)
    System.out.println(i * i);
```
1. 11 times
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);
```

• A missing semicolon

```java
for (years = 1; (balance = balance + balance * 
    rate / 100) < targetBalance; years++)
System.out.println(years);
```
Nested Loops

• Create triangle pattern

[[]]
 [[][]]
 [[][][]]
 [[][][][]]
 [[][][][]]

• Loop through rows

```java
for (int i = 1; i <= n; i++)
{
    // make triangle row
}
```
Nested Loops

- *Make triangle row* is another loop

```java
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:
   * []
   * [][]
   * [][][
   * ][][]
   */

public class Triangle {

   /**
   * Constructs a triangle.
   * @param aWidth the number of [] in the last row of the triangle.
   */
   public Triangle(int aWidth) {
      width = aWidth;
   }

   // Continued...
/**
 * Computes a string representing the triangle.
 * @return a string consisting of [] and newline characters
 */

public String toString()
{
    String r = "";
    for (int i = 1; i <= width; i++)
    {
        // Make triangle row
        for (int j = 1; j <= i; j++)
        {
            r = r + "[]";
        }
        r = r + "\n";
    }
    return r;
}

private int width;
/**
 * This program tests the Triangle class.
 */

class TriangleTester {
    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());
    }
}
Self Check

1. How would you modify the nested loops so that you print a square instead of a triangle?

2. What is the value of n after the following nested loops?

```java
int n = 0;
for (int i = 1; i <= 5; i++)
    for (int j = 0; j < i; j++)
        n = n + j;
```
Answers

1. Change the inner loop to
   
   ```java
   for (int j = 1; j <= width; j++)
   ```

2. 20
Processing Sentinel Values

- **Sentinel value**: Can be used for indicating the end of a data set
- **0 or -1 make poor sentinels; better use Q**

```java
System.out.print("Enter value, Q to quit: ");
String input = in.next();
if (input.equalsIgnoreCase("Q"))
    We are done
else
{
    double x = Double.parseDouble(input);
    ...
}
```
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
    }
}
```
```java
import java.util.Scanner;

/**
 * This program computes the average and maximum of a set of input values.
 */
public class InputTester {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        DataSet data = new DataSet();

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

class DataSet {
    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
         */
        // Continued...
public void add(double x) {
    sum = sum + x;
    if (count == 0 || maximum < x) maximum = x;
    count++;
}

/**
 * Gets the average of the added data.
 * @return the average or 0 if no data has been added
 */
public double getAverage() {
    if (count == 0) return 0;
    else return sum / count;
}
/**
 * Gets the largest of the added data.
 * @return the maximum or 0 if no data has been added
 */

public double getMaximum()
{
    return maximum;
}

private double sum;
private double maximum;
private int count;
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
Self Check

1. Why does the `InputTester` class call `in.next` and not `in.nextDouble`?

2. 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;
   ```
Answers

1. Because we don't know whether the next input is a number or the letter Q.

2. No. If all input values are negative, the maximum is also negative. However, the maximum field is initialized with 0. With this simplification, the maximum would be falsely computed as 0.
Random Numbers and Simulations

- In a simulation, you repeatedly generate random numbers and use them to simulate an activity
- Random number generator

```java
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)

```java
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
 */

public int cast()
{
    return 1 + generator.nextInt(sides);
}

private Random generator;
private int sides;
/**
 * This program simulates casting a die ten times.
 */

public class DieTester
{
    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
Buffon Needle Experiment

Figure 5:
The Buffon Needle Experiment
Figure 6:
When Does a Needle Fall on a Line?
Needle Position

• Needle length = 1, distance between lines = 2
• Generate random $y_{low}$ between 0 and 2
• Generate random angle $\alpha$ between 0 and 180 degrees
• $y_{high} = y_{low} + \sin(\alpha)$
• Hit if $y_{high} \geq 2$
import java.util.Random;

/**
 * This class simulates a needle in the Buffon needle experiment.
 */

public class Needle {

    /**
     * Constructs a needle.
     */
    public Needle() {
        hits = 0;
        tries = 0;
        generator = new Random();
    }

    // Continued…
Drops the needle on the grid of lines and remembers whether the needle hit a line.

```java
public void drop()
{
    double ylow = 2 * generator.nextDouble();
    double angle = 180 * generator.nextDouble();

    // Computes high point of needle
    double yhigh = ylow + Math.sin(Math.toRadians(angle));
    if (yhigh >= 2) hits++;
    tries++;
}
```

Gets the number of times the needle hit a line.

```java
@returns the hit count
```
```java
public int getHits()
{
    return hits;
}

/**
 * Gets the total number of times the needle was dropped.
 * @return the try count
 */
public int getTries()
{
    return tries;
}

private Random generator;
private int hits;
private int tries;
```
/**
   * This program simulates the Buffon needle experiment
   * and prints the resulting approximations of pi.
   */

public class NeedleTester
{
    public static void main(String[] args)
    {
        Needle n = new Needle();
        final int TRIES1 = 10000;
        final int TRIES2 = 1000000;
        Continued...
for (int i = 1; i <= TRIES1; i++)
    n.drop();

System.out.printf("Tries = %d, Tries / Hits = %8.5f\n", TRIES1, (double) n.getTries() / n.getHits());

for (int i = TRIES1 + 1; i <= TRIES2; i++)
    n.drop();

System.out.printf("Tries = %d, Tries / Hits = %8.5f\n", TRIES2, (double) n.getTries() / n.getHits());

Output

Tries = 10000, Tries / Hits = 3.08928
Tries = 1000000, Tries / Hits = 3.14204
Self Check

1. How do you use a random number generator to simulate the toss of a coin?

2. Why is the NeedleTester program not an efficient method for computing π?
Answers

1. `int n = generator.nextInt(2); // 0 = heads, 1 = tails`

2. The program repeatedly calls `Math.toRadians(angle)`. You could simply call `Math.toRadians(180)` to compute \( \pi \)