Chapter Goals

- To implement decisions using the `if` statement
- To compare integers, floating-point numbers, and Strings
- To write statements using the Boolean data type
- To develop strategies for testing your programs
- To validate user input
The **if** Statement

- A computer program often needs to make decisions based on input, or circumstances
- For example, buildings often ‘skip’ the 13th floor, and elevators should too
  - The 14th floor is really the 13th floor
  - So every floor above 12 is really ‘floor – 1’
    - If floor > 12, Actual floor = floor - 1
- The two keywords of the if statement are:
  - **if**
  - **else**
- The **if** statement allows a program to carry out different actions depending on the nature of the data to be processed.
Flowchart of the `if` Statement

- One of the two branches is executed once
  - True (`if`) branch or False (`else`) branch

```plaintext
int actualFloor;
if (floor > 13)
{
    actualFloor = floor - 1;
}
else
{
    actualFloor = floor;
}
```
Flowchart with only a True Branch

- An `if` statement may not need a ‘False’ (else) branch

```java
int actualFloor = floor;
if (floor > 13) {
    actualFloor--;  
} // No else needed
```
### Syntax 3.1 The `if` Statement

#### Syntax
```plaintext
if (condition) {
    statements
} else {
    statements2
}
```

A condition that is true or false. Often uses relational operators:

- `==`
- `!=`
- `<`
- `<=`
- `>`
- `>=`

(See Table 1.)

#### Notes:
- Braces are not required if the branch contains a single statement, but it's good to always use them. See Programming Tip 3.2.
- Omit the `else` branch if there is nothing to do. See Programming Tip 3.1.
- Lining up braces is a good idea. See Programming Tip 3.1.
- Don't put a semicolon here! See Common Error 3.1.

```plaintext
if (floor > 13) {
    actualFloor = floor - 1;
} else {
    actualFloor = floor;
}
```

If the condition is true, the statement(s) in this branch are executed in sequence; if the condition is false, they are skipped.

If the condition is false, the statement(s) in this branch are executed in sequence; if the condition is true, they are skipped.
```java
import java.util.Scanner;

/**
 * This program simulates an elevator panel that skips the 13th floor.
 */

public class ElevatorSimulation
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Floor: ");
        int floor = in.nextInt();

        // Adjust floor if necessary

        int actualFloor;
        if (floor > 13)
        {
            actualFloor = floor - 1;
        }
        else
        {
            actualFloor = floor;
        }

        System.out.println("The elevator will travel to the actual floor "+ actualFloor);
    }
}
```

**Program Run**

Floor: 20
The elevator will travel to the actual floor 19
Self Check 3.1

In some Asian countries, the number 14 is considered unlucky. Some building owners play it safe and *skip* both the thirteenth and the fourteenth floor. How would you modify the sample program to handle such a building?

**Answer:** Change the `if` statement to

```java
if (floor > 14)
{
    actualFloor = floor - 2;
}
```
Consider the following if statement to compute a discounted price:

```java
if (originalPrice > 100)
{
    discountedPrice = originalPrice - 20;
}
else
{
    discountedPrice = originalPrice - 10;
}
```

What is the discounted price if the original price is 95? 100? 105?

**Answer:** 85. 90. 85.
Self Check 3.3

Compare this if statement with the one in Self Check 2:

```java
if (originalPrice < 100)
{
    discountedPrice = originalPrice - 10;
}
else
{
    discountedPrice = originalPrice - 20;
}
```

Do the two statements always compute the same value? If not, when do the values differ?

**Answer:** The only difference is if `originalPrice` is 100. The statement in Self Check 2 sets `discountedPrice` to 90; this one sets it to 80.
Self Check 3.4

Consider the following statements to compute a discounted price:

```java
discountedPrice = originalPrice;
if (originalPrice > 100)
{
    discountedPrice = originalPrice - 10;
}
```

What is the discounted price if the original price is 95? 100? 105?

**Answer:** 95. 100. 95.
The variables `fuelAmount` and `fuelCapacity` hold the actual amount of fuel and the size of the fuel tank of a vehicle. If less than 10 percent is remaining in the tank, a status light should show a red color; otherwise it shows a green color. Simulate this process by printing out either "red" or "green".

**Answer:**

```java
if (fuelAmount < 0.10 * fuelCapacity) {
    System.out.println("red");
} else {
    System.out.println("green");
}
```
Tips On Using Braces

- Line up all pairs of braces vertically
  - Lined up
  - Not aligned (saves lines)

```plaintext
if (floor > 13) {
    floor--; 
}
```

- Always use braces
  - Although single statement clauses do not require them

```plaintext
if (floor > 13) {
    floor--; 
}
```

- Most programmer’s editors have a tool to align matching braces.
Tips on Indenting Blocks

▪ Use Tab to indent a consistent number of spaces

This is referred to as ‘block-structured’ code. Indenting consistently makes code much easier for humans to follow.
Common Error

- A semicolon after an `if` statement
- It is easy to forget and add a semicolon after an `if` statement
  - The true path is now the space just before the semicolon

```java
if (floor > 13) ;
{
  floor--; 
}
```

- The ‘body’ (between the curly braces) will always be executed in this case
The Conditional Operator

- A ‘shortcut’ you may find in existing code
  - It is not used in this book

  \[
  \text{actualFloor} = \text{floor} > 13 \ ? \ \text{floor} - 1 \ : \ \text{floor};
  \]

- Includes all parts of an if-else clause, but uses:
  - `?` To begin the true branch
  - `:` To end the true branch and start the false branch
Comparing Numbers and Strings

- Every if statement has a condition
  - Usually compares two values with an operator

```
if (floor > 13) ..
if (floor >= 13) ..
if (floor < 13) ..
if (floor <= 13) ..
if (floor == 13) ..
```

**Table 1** Relational Operators

<table>
<thead>
<tr>
<th>Java</th>
<th>Math Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>≥</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>≤</td>
<td>Less than or equal</td>
</tr>
<tr>
<td>==</td>
<td>=</td>
<td>Equal</td>
</tr>
<tr>
<td>!=</td>
<td>≠</td>
<td>Not equal</td>
</tr>
</tbody>
</table>
Syntax 3.2 Comparisons

These quantities are compared.

floor > 13

One of: == !< <= >= (See Table 1.)

floor == 13

Checks for equality.

String input;
if (input.equals("Y"))

Use equals to compare strings. (See Common Error 3.3.)

double x; double y; final double EPSILON = 1E-14;
if (Math.abs(x - y) < EPSILON)

Checks that these floating-point numbers are very close. (See Common Error 3.2.)

Check that you have the right direction:
> (greater) or < (less)

Check the boundary condition:
> (greater) or >= (greater or equal)?

Use ==, not =.
Operator Precedence

- The comparison operators have lower precedence than arithmetic operators
  - Calculations are done before the comparison
  - Normally your calculations are on the ‘right side’ of the comparison or assignment operator

```java
actualFloor = floor + 1;
if (floor > height + 1)
```
# Relational Operator Use

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 &lt;= 4</td>
<td>true</td>
<td>3 is less than 4; &lt;= tests for “less than or equal”.</td>
</tr>
<tr>
<td>3 &lt;= 4</td>
<td>Error</td>
<td>The “less than or equal” operator is &lt;=, not =&lt;. The “less than” symbol comes first.</td>
</tr>
<tr>
<td>3 &gt; 4</td>
<td>false</td>
<td>&gt; is the opposite of &lt;=.</td>
</tr>
<tr>
<td>4 &lt; 4</td>
<td>false</td>
<td>The left-hand side must be strictly smaller than the right-hand side.</td>
</tr>
<tr>
<td>4 &lt;= 4</td>
<td>true</td>
<td>Both sides are equal; &lt;= tests for “less than or equal”.</td>
</tr>
<tr>
<td>3 == 5 - 2</td>
<td>true</td>
<td>== tests for equality.</td>
</tr>
<tr>
<td>3 != 5 - 1</td>
<td>true</td>
<td>!= tests for inequality. It is true that 3 is not 5 - 1.</td>
</tr>
<tr>
<td>3 = 6 / 2</td>
<td>Error</td>
<td>Use == to test for equality.</td>
</tr>
<tr>
<td>1.0 / 3.0 == 0.3333333333</td>
<td>false</td>
<td>Although the values are very close to one another, they are not exactly equal. See Common Error 3.2 on page 93.</td>
</tr>
<tr>
<td>&quot;10&quot; &gt; 5</td>
<td>Error</td>
<td>You cannot compare a string to a number.</td>
</tr>
<tr>
<td>&quot;Tomato&quot;.substring(0, 3).equals(&quot;Tom&quot;)</td>
<td>true</td>
<td>Always use the equals method to check whether two strings have the same contents.</td>
</tr>
<tr>
<td>&quot;Tomato&quot;.substring(0, 3) == (&quot;Tom&quot;)</td>
<td>false</td>
<td>Never use == to compare strings; it only checks whether the strings are stored in the same location. See Common Error 3.3 on page 94.</td>
</tr>
</tbody>
</table>
Comparing Strings

- Strings are a bit ‘special’ in Java
- Do not use the `==` operator with Strings
  - The following compares the locations of two strings, and not their contents

```java
if (string1 == string2) ...
```

- Instead use the String’s `equals` method:

```java
if (string1.equals(string2)) ...
```
Self Check 3.6

Which of the following conditions are true, provided \(a\) is 3 and \(b\) is 4?

- a. \(a + 1 \leq b\)
- b. \(a + 1 \geq b\)
- c. \(a + 1 \neq b\)

**Answer:** (a) and (b) are both true, (c) is false.
Give the opposite of the condition

\[ \text{floor} > 13 \]

**Answer:** \[ \text{floor} \leq 13 \]
What is the error in this statement?

```java
if (scoreA = scoreB)
{
    System.out.println("Tie");
}
```

**Answer:** The values should be compared with `==`, not `=`.
Supply a condition in this `if` statement to test whether the user entered a `Y`:

```java
System.out.println("Enter Y to quit.");
String input = in.next();
if (. . .)
{
    System.out.println("Goodbye.");
}

**Answer:** input.equals("Y")
Self Check 3.10

How do you test that a string \texttt{str} is the empty string?

\textbf{Answer:} \texttt{str.equals(""')} or \texttt{str.length()} == 0
Common Error

- Comparison of Floating-Point Numbers
  - Floating-point numbers have limited precision
  - Round-off errors can lead to unexpected results

```java
double r = Math.sqrt(2.0);
if (r * r == 2.0)
{
    System.out.println("Math.sqrt(2.0) squared is 2.0");
}
else
{
    System.out.println("Math.sqrt(2.0) squared is not 2.0 but \" + r * r);
}
```

Output:
Math.sqrt(2.0) squared is not 2.0 but 2.00000000000000044
The Use of EPSILON

- Use a very small value to compare the difference if floating-point values are ‘close enough’
  - The magnitude of their difference should be less than some threshold
  - Mathematically, we would write that x and y are close enough if:

\[ |x - y| < \varepsilon \]

```java
final double EPSILON = 1E-14;
double r = Math.sqrt(2.0);
if (Math.abs(r * r - 2.0) < EPSILON) {
    System.out.println("Math.sqrt(2.0) squared is approx. 2.0");
}
```
Common Error

- Using `==` to compare Strings
  - `==` compares the locations of the Strings
- Java creates a new String every time a new word inside double-quotes is used
  - If there is one that matches it exactly, Java re-uses it

```java
String nickname = "Rob";

if (nickname == "Rob") // Test is true

String name = "Robert";
String nickname = name.substring(0, 3);

if (nickname == "Rob") // Test is false
```
Lexicographical Order

- To compare Strings in ‘dictionary’ order
  - When compared using `compareTo`, `string1` comes:
    - Before `string2` if `string1.compareTo(string2) < 0`
    - After `string2` if `string1.compareTo(string2) > 0`
    - Equal to `string2` if `string1.compareTo(string2) == 0`

- Notes
  - All UPPERCASE letters come before lowercase
  - ‘space’ comes before all other printable characters
  - Digits (0-9) come before all letters
  - See Appendix A for the Basic Latin Unicode (ASCII) table
Implementing an `if` Statement

1) Decide on a branching condition
   - original price < 128?
2) Write pseudocode for the true branch
   - discounted price = 0.92 x original price
3) Write pseudocode for the false branch
   - discounted price = 0.84 x original price
4) Double-check relational operators
   - Test values below, at, and above the comparison (127, 128, 129)
5) Remove duplication
   - discounted price = ___ x original price
6) Test both branches
   - discounted price = 0.92 x 100 = 92
   - discounted price = 0.84 x 200 = 168
7) Write the code in Java
Implemented Example

- The university bookstore has a Kilobyte Day sale every October 24, giving an 8 percent discount on all computer accessory purchases if the price is less than $128, and a 16 percent discount if the price is at least $128.

```java
if (originalPrice < 128)
{
    discountRate = 0.92;
}
else
{
    discountRate = 0.84;
}
discountedPrice = discountRate * originalPrice;
```
Multiple Alternatives

- What if you have more than two branches?
- Count the branches for the following earthquake effect example:

  - 8.0 (or greater)
  - 7.0 but < 8.0
  - 6.0 but < 7.0
  - 4.5 but < 6.00
  - Less than 4.5

<table>
<thead>
<tr>
<th>Value</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Most structures fall</td>
</tr>
<tr>
<td>7</td>
<td>Many buildings destroyed</td>
</tr>
<tr>
<td>6</td>
<td>Many buildings considerably damaged,</td>
</tr>
<tr>
<td></td>
<td>some collapse</td>
</tr>
<tr>
<td>4.5</td>
<td>Damage to poorly constructed buildings</td>
</tr>
</tbody>
</table>

- When using multiple `if` statements, test general conditions after more specific conditions.
Flowchart of Multiway Branching

> 8.0?
True
Most Structures Fall
False

>= 7.0?
True
Many Buildings Destroyed
False

>= 6.0?
True
Many buildings considerably damaged, some collapse
False

>= 4.5?
True
Damage to poorly constructed buildings
False
No destruction of buildings
if, else if Multiway Branching

```java
if (richter >= 8.0) // Handle the 'special case' first
{
    System.out.println("Most structures fall");
}
else if (richter >= 7.0)
{
    System.out.println("Many buildings destroyed");
}
else if (richter >= 6.0)
{
    System.out.println("Many buildings damaged, some collapse");
}
else if (richter >= 4.5)
{
    System.out.println("Damage to poorly constructed buildings");
}
else // so that the 'general case' can be handled last
{
    System.out.println("No destruction of buildings");
}
```
What Is Wrong with this Code?

```java
if (richter >= 8.0)
{
    System.out.println("Most structures fall");
}
if (richter >= 7.0)
{
    System.out.println("Many buildings destroyed");
}
if (richter >= 6.0)
{
    System.out.println("Many buildings damaged, some collapse");
}
if (richter >= 4.5)
{
    System.out.println("Damage to poorly constructed buildings");
}
```
Self Check 3.11

In a game program, the scores of players A and B are stored in variables `scoreA` and `scoreB`. Assuming that the player with the larger score wins, write an `if/else if/else` sequence that prints out "A won", "B won", or "Game tied".

Answer:

```java
if (scoreA > scoreB)
{
    System.out.println("A won");
}
else if (scoreA < scoreB)
{
    System.out.println("B won");
}
else
{
    System.out.println("Game tied");
}
```
Self Check 3.12

Write a conditional statement with three branches that sets s to 1 if x is positive, to –1 if x is negative, and to 0 if x is zero.

Answer:

```plaintext
if (x > 0) { s = 1; }
else if (x < 0) { s = -1; }
else { s = 0; }
```
How could you achieve the task of Self Check 12 with only two branches?

**Answer:** You could first set \( s \) to one of the three values:

\[
\begin{align*}
    s &= 0; \\
    &\text{if (} x > 0 \text{) \{} s = 1; \} \\
    &\text{else if (} x < 0 \text{) \{} s = -1; \}
\end{align*}
\]
Self Check 3.14

Beginners sometimes write statements such as the following:

```c
if (price > 100)
{
    discountedPrice = price - 20;
}
else if (price <= 100)
{
    discountedPrice = price - 10;
}
```

Explain how this code can be improved.

**Answer:** The `if (price <= 100)` can be omitted (leaving just `else`), making it clear that the `else` branch is the sole alternative.
Self Check 3.15

Suppose the user enters -1 into the earthquake program. What is printed?

**Answer:** No destruction of buildings.
Suppose we want to have the earthquake program check whether the user entered a negative number. What branch would you add to the if statement, and where?

**Answer:** Add a branch before the final else:

```java
else if (richter < 0)
{
    System.out.println("Error: Negative input");
}
```
Another Way to Multiway Branch

- The `switch` statement chooses a `case` based on an integer value.
- `break` ends each `case`
- `default` catches all other values
- If the `break` is missing, the case falls through to the next `case`'s statements.

```java
int digit = ...;
switch (digit)
{
    case 1: digitName = "one"; break;
    case 2: digitName = "two"; break;
    case 3: digitName = "three"; break;
    case 4: digitName = "four"; break;
    case 5: digitName = "five"; break;
    case 6: digitName = "six"; break;
    case 7: digitName = "seven"; break;
    case 8: digitName = "eight"; break;
    case 9: digitName = "nine"; break;
    default: digitName = ""; break;
}
```
Nested Branches

- You can *nest* an `if` inside either branch of an `if` statement.
- Simple example: Ordering drinks
  - Ask the customer for their drink order
    - `if` customer orders wine
      - Ask customer for ID
        - `if` customer’s age is 21 or over
          - Serve wine
        - Else
          - Politely explain the law to the customer
    - Else
      - Serve customers a non-alcoholic drink
Flowchart of a Nested if

- Nested **if-else** inside true branch of an **if** statement.
- Three paths

Ask for order

Wine? True
  - Check ID
    - True: Serve wine
    - False: Read law

Wine? False
  - Serve non-alcoholic drink

Done
Tax Example: Nested if's

- Four outcomes (branches)

<table>
<thead>
<tr>
<th>Single</th>
<th>&lt;= 32000</th>
<th>=&gt; 32000</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Married</th>
<th>&lt;= 64000</th>
<th>=&gt; 64000</th>
</tr>
</thead>
</table>

### Table 4  Federal Tax Rate Schedule

<table>
<thead>
<tr>
<th>If your status is Single and if the taxable income is</th>
<th>the tax is</th>
<th>of the amount over</th>
</tr>
</thead>
<tbody>
<tr>
<td>at most $32,000</td>
<td>10%</td>
<td>$0</td>
</tr>
<tr>
<td>over $32,000</td>
<td>$3,200 + 25%</td>
<td>$32,000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>If your status is Married and if the taxable income is</th>
<th>the tax is</th>
<th>of the amount over</th>
</tr>
</thead>
<tbody>
<tr>
<td>at most $64,000</td>
<td>10%</td>
<td>$0</td>
</tr>
<tr>
<td>over $64,000</td>
<td>$6,400 + 25%</td>
<td>$64,000</td>
</tr>
</tbody>
</table>
import java.util.Scanner;

/**
   * This program computes income taxes, using a simplified tax schedule.
   */

public class TaxCalculator
{

    public static void main(String[] args)
    {
        final double RATE1 = 0.10;
        final double RATE2 = 0.25;
        final double RATE1_SINGLE_LIMIT = 32000;
        final double RATE1_MARRIED_LIMIT = 64000;

        double tax1 = 0;
        double tax2 = 0;

        // Read income and marital status
        Scanner in = new Scanner(System.in);
        System.out.print("Please enter your income: ");
        double income = in.nextDouble();

        System.out.print("Please enter s for single, m for married: ");
        String maritalStatus = in.next();
The 'True' branch (Married)
  Two branches within this branch

```java
// Compute taxes due
if (maritalStatus.equals("s")) {
    if (income <= RATE1_SINGLE_LIMIT) {
        tax1 = RATE1 * income;
    } else {
        tax1 = RATE1 * RATE1_SINGLE_LIMIT;
        tax2 = RATE2 * (income - RATE1_SINGLE_LIMIT);
    }
}
```
The ‘False’ branch (Not married)

```java
else {
    if (income <= RATE1_MARRIED_LIMIT)
        tax1 = RATE1 * income;
    else
        tax1 = RATE1 * RATE1_MARRIED_LIMIT;
        tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT);
}

double totalTax = tax1 + tax2;
System.out.println("The tax is $" + totalTax);
```
Self Check 3.17

What is the amount of tax that a single taxpayer pays on an income of $32,000?

**Answer:** 3200.
Would that amount change if the first nested if statement changed from
if (income <= RATE1_SINGLE_LIMIT)
to
if (income < RATE1_SINGLE_LIMIT)

**Answer:** No. Then the computation is $0.10 \times 32000 + 0.25 \times (32000 - 32000)$. 
Suppose Harry and Sally each make $40,000 per year. Would they save taxes if they married?

**Answer:** No. Their individual tax is $5,200 each, and if they married, they would pay $10,400. Actually, taxpayers in higher tax brackets (which our program does not model) may pay higher taxes when they marry, a phenomenon known as the *marriage penalty*. 
Self Check 3.20

How would you modify the `TaxCalculator.java` program in order to check that the user entered a correct value for the marital status (i.e., s or m)?

**Answer:** Change `else` in line 41 to

```java
else if (maritalStatus.equals("m"))
```

and add another branch after line 52:

```java
else {
    System.out.println("Error: marital status should be s or m.");
}
```
Self Check 3.21

Some people object to higher tax rates for higher incomes, claiming that you might end up with less money after taxes when you get a raise for working hard. What is the flaw in this argument?

**Answer:** The higher tax rate is only applied on the income in the higher bracket. Suppose you are single and make $31,900. Should you try to get a $200 raise? Absolutely: you get to keep 90 percent of the first $100 and 75 percent of the next $100.
Hand-Tracing

- Hand-tracing helps you understand whether a program works correctly
- Create a table of key variables
  - Use pencil and paper to track their values
- Works with pseudocode or code
  - Track location with a marker such as a paper clip
- Use example input values that:
  - You know what the correct outcome should be
  - Will test each branch of your code
Hand-Tracing Tax Example (1)

Setup
- Table of variables
- Initial values

```
8   public static void main(String[] args) {
9     final double RATE1 = 0.10;
10    final double RATE2 = 0.25;
11    final double RATE1_SINGLE_LIMIT = 32000;
12    final double RATE1_MARRIED_LIMIT = 64000;
13     double tax1 = 0;
14     double tax2 = 0;
```
Hand-Tracing Tax Example (2)

- Input variables
  - From user
  - Update table

<table>
<thead>
<tr>
<th>tax1</th>
<th>tax2</th>
<th>income</th>
<th>marital status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>80000</td>
<td>m</td>
</tr>
</tbody>
</table>

20    Scanner in = new Scanner(System.in);
21    System.out.print("Please enter your income: ");
22    double income = in.nextDouble();
23
24    System.out.print("Please enter s for single, m for married: ");
25    String maritalStatus = in.next();

- Because marital status is not “s” we skip to the else on line 41

29    if (maritalStatus.equals("s"))
30    {
41          else
42    {

Because income is not <= 64000, we move to the else clause on line 47
- Update variables on lines 49 and 50
- Use constants

<table>
<thead>
<tr>
<th>tax1</th>
<th>tax2</th>
<th>income</th>
<th>marital status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>80000</td>
<td>m</td>
</tr>
</tbody>
</table>

```
43  if (income <= RATE1_MARRIED_LIMIT) {
44       tax1 = RATE1 * income;
45  } else {
47        tax1 = RATE1 * RATE1_MARRIED_LIMIT;
49        tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT);
51    }
```
Output

- Calculate
- As expected?

<table>
<thead>
<tr>
<th>tax1</th>
<th>tax2</th>
<th>income</th>
<th>marital status</th>
<th>total tax</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>80000</td>
<td>m</td>
<td>10400</td>
</tr>
<tr>
<td>6400</td>
<td>4000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```java
54    double totalTax = tax1 + tax2;
55    System.out.println("The tax is "+ totalTax);
57    }
```
Common Error

- The Dangling `else` Problem
  - When an `if` statement is nested inside another `if` statement, the following can occur:

```java
double shippingCharge = 5.00; // $5 inside continental U.S.
if (country.equals("USA"))
  if (state.equals("HI"))
      shippingCharge = 10.00; // Hawaii is more expensive
else // Pitfall!
    shippingCharge = 20.00; // As are foreign shipment
```

- The indentation level suggests that the `else` is related to the `if` `country` ("USA")
  - Else clauses always associate to the closest `if`
Enumerated Types

- Java provides an easy way to name a finite list of values that a variable can hold
  - It is like declaring a new type, with a list of possible values

```java
public enum FilingStatus {
    SINGLE, MARRIED, MARRIED_FILING_SEPARATELY
}
```

- You can have any number of values, but you must include them all in the `enum` declaration

- You can declare variables of the enumeration type:

```java
FilingStatus status = FilingStatus.SINGLE;
```

- And you can use the comparison operator with them:

```java
if (status == FilingStatus.SINGLE) . . .
```
Problem Solving: Flowcharts

- You have seen a few basic flowcharts
- A flowchart shows the structure of decisions and tasks to solve a problem
- Basic flowchart elements:
  - Connect them with arrows
  - But never point an arrow
  - inside another branch!
Conditional Flowcharts

- **Two Outcomes**
  - Condition
    - True branch
    - False branch

- **Multiple Outcomes**
  - Choice 1
    - True branch
    - False
      - Choice 2
        - True branch
        - False
          - Choice 3
            - True branch
            - False
              - “Other” branch
Shipping costs are $5 inside the United States, except that to Hawaii and Alaska they are $10. International shipping costs are also $10.

- Three Branches:

  - International Branch
  - Hawaii/Alaska Branch
  - Lower 48 Branch
Don’t Connect Branches!

Shipping costs are $5 inside the United States, except that to Hawaii and Alaska they are $10. International shipping costs are also $10.

- Do not do this!
Shipping costs are $5 inside the United States, except that to Hawaii and Alaska they are $10. International shipping costs are also $10.

- Completed
Self Check 3.22

Draw a flowchart for a program that reads a value `temp` and prints “Frozen” if it is less than zero.

Answer:
Self Check 3.23

What is wrong with the flowchart at right?

Answer: The “True” arrow from the first decision points into the “True” branch of the second decision, creating spaghetti code.
Self Check 3.24

How do you fix the flowchart of Self Check 23?

**Answer:** Here is one solution. In Section 3.7, you will see how you can combine the conditions for a more elegant solution.
Self Check 3.25

Draw a flowchart for a program that reads a value \( x \). If it is less than zero, print “Error”. Otherwise, print its square root.

Answer:
Self Check 3.26

Draw a flowchart for a program that reads a value $\text{temp}$. If it is less than zero, print “Ice”. If it is greater than 100, print “Steam”. Otherwise, print “Liquid”.

Answer:
Problem Solving: Test Cases

- Aim for complete *coverage* of all decision points:
  - There are two possibilities for the marital status and two tax brackets for each status, yielding four test cases
  - Test a handful of *boundary* conditions, such as an income that is at the boundary between two tax brackets, and a zero income
  - If you are responsible for error checking (which is discussed in Section 3.8), also test an invalid input, such as a negative income
Choosing Test Cases

- Choose input values that:
  - Test boundary cases and 0 values
  - A *boundary case* is a value that is tested in the code
  - Test each branch

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Expected Output</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>30,000</td>
<td>s</td>
<td>3,000</td>
</tr>
<tr>
<td>72,000</td>
<td>s</td>
<td>13,200</td>
</tr>
<tr>
<td>50,000</td>
<td>m</td>
<td>5,000</td>
</tr>
<tr>
<td>104,000</td>
<td>m</td>
<td>16,400</td>
</tr>
<tr>
<td>32,000</td>
<td>m</td>
<td>3,200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Self Check 3.27

Using Figure 1 on page 85 as a guide, follow the process described in Section 3.6 to design a set of test cases for the `ElevatorSimulation.java` program in Section 3.1.

Answer:

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Expected Output</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>Below 13th floor</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>Above 13th floor</td>
</tr>
<tr>
<td>13</td>
<td>?</td>
<td>The specification is not clear—See Section 3.8 for a version of this program with error handling</td>
</tr>
</tbody>
</table>
What is a boundary test case for the algorithm in How To 3.1 on page 95? What is the expected output?

**Answer:** A boundary test case is a price of $128. A 16 percent discount should apply because the problem statement states that the larger discount applies if the price is *at least* $128. Thus, the expected output is $107.52.
Self Check 3.29

Using Figure 3 on page 99 as a guide, follow the process described in Section 3.6 to design a set of test cases for the `EarthquakeStrength.java` program in Section 3.3.

**Answer:**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Expected Output</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Most structures fall</td>
<td></td>
</tr>
<tr>
<td>7.5</td>
<td>Many buildings destroyed</td>
<td></td>
</tr>
<tr>
<td>6.5</td>
<td>Many buildings ...</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Damage to poorly...</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>No destruction...</td>
<td></td>
</tr>
<tr>
<td>8.0</td>
<td>Most structures fall</td>
<td>Boundary case. In this program, boundary cases are not as significant because the behavior of an earthquake changes gradually.</td>
</tr>
<tr>
<td>-1</td>
<td>The specification is not clear—see Self Check 16 for a version of this program with error handling.</td>
<td></td>
</tr>
</tbody>
</table>
Suppose you are designing a part of a program for a medical robot that has a sensor returning an \( x \)- and \( y \)-location (measured in cm). You need to check whether the sensor location is inside the circle, outside the circle, or on the boundary (specifically, having a distance of less than 1 mm from the boundary). Assume the circle has center \((0, 0)\) and a radius of 2 cm. Give a set of test cases.

**Answer:**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Expected Output</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>((0.5, 0.5))</td>
<td>inside</td>
<td></td>
</tr>
<tr>
<td>((4, 2))</td>
<td>outside</td>
<td></td>
</tr>
<tr>
<td>((0, 2))</td>
<td>on the boundary</td>
<td>Exactly on the boundary</td>
</tr>
<tr>
<td>((1.414, 1.414))</td>
<td>on the boundary</td>
<td>Close to the boundary</td>
</tr>
<tr>
<td>((0, 1.9))</td>
<td>inside</td>
<td>Not less than 1 mm from the boundary</td>
</tr>
<tr>
<td>((0, 2.1))</td>
<td>outside</td>
<td>Not less than 1 mm from the boundary</td>
</tr>
</tbody>
</table>
Boolean Variables

- **Boolean Variables**
  - A Boolean variable is often called a flag because it can be either up (**true**) or down (**false**)
  - **boolean** is a Java data type
    - ```java
        boolean failed = true;
      ```
  - Can be either **true** or **false**

- **Boolean Operators:** `&&` and `||`
  - They combine multiple conditions
    - `&&` is the **and** operator
    - `||` is the **or** operator
Character Testing Methods

The `Character` class has a number of handy methods that return a boolean value:

```java
if (Character.isDigit(ch))
{
    ...
}
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Examples of Accepted Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isDigit</code></td>
<td>0, 1, 2</td>
</tr>
<tr>
<td><code>isLetter</code></td>
<td>A, B, C, a, b, c</td>
</tr>
<tr>
<td><code>isUpperCase</code></td>
<td>A, B, C</td>
</tr>
<tr>
<td><code>isLowerCase</code></td>
<td>a, b, c</td>
</tr>
<tr>
<td><code>isWhiteSpace</code></td>
<td>space, newline, tab</td>
</tr>
</tbody>
</table>
### Combined Conditions: `&&`

- Combining two conditions is often used in range checking
  - Is a value between two other values?
  - Both sides of the *and* must be true for the result to be true

```java
if (temp > 0 && temp < 100) {
    System.out.println("Liquid");
}
```

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A &amp;&amp; B</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
Combined Conditions:  ||

- If only one of two conditions need to be true
  - Use a compound conditional with an or:

```java
if (balance > 100 || credit > 100)
{
    System.out.println("Accepted");
}
```

- If either is true
  - The result is true

| A    | B    | A ||| B |
|------|------|------|
| true | true | true |
| true | false| true |
| false| true | true |
| false| false| false |
The *not* Operator: !

- If you need to invert a boolean variable or comparison, precede it with !

```java
if (!attending || grade < 60)
{
    System.out.println("Drop?");
}
```

- If using !, try to use simpler logic:

```java
if (attending && (grade >= 60))
```

and Flowchart

- This is often called ‘range checking’
  - Used to validate that input is between two values

```java
if (temp > 0 && temp < 100) {
    System.out.println("Liquid");
}
```
Another form of ‘range checking’
- Checks if value is outside a range

```
if (temp <= 0 || temp >= 100)
{
    System.out.println("Not Liquid");
}
```
### Boolean Operator Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 &lt; 200 &amp;&amp; 200 &lt; 100</td>
<td>false</td>
<td>Only the first condition is true.</td>
</tr>
<tr>
<td>0 &lt; 200</td>
<td></td>
<td>200 &lt; 100</td>
</tr>
<tr>
<td>0 &lt; 200</td>
<td></td>
<td>100 &lt; 200</td>
</tr>
<tr>
<td>0 &lt; (x) &amp;&amp; (x &lt; 100)</td>
<td></td>
<td>(x == -1)</td>
</tr>
<tr>
<td>(\boxed{0 &lt; x &lt; 100})</td>
<td>Error</td>
<td><strong>Error</strong>: This expression does not test whether (x) is between 0 and 100. The expression (0 &lt; x) is a Boolean value. You cannot compare a Boolean value with the integer 100.</td>
</tr>
<tr>
<td>(\boxed{x &amp;&amp; y &gt; 0})</td>
<td>Error</td>
<td><strong>Error</strong>: This expression does not test whether (x) and (y) are positive. The left-hand side of &amp;&amp; is an integer, (x), and the right-hand side, (y &gt; 0), is a Boolean value. You cannot use &amp;&amp; with an integer argument.</td>
</tr>
<tr>
<td>!(0 &lt; 200)</td>
<td>false</td>
<td>0 &lt; 200 is true, therefore its negation is false.</td>
</tr>
<tr>
<td>frozen == true</td>
<td>frozen</td>
<td>There is no need to compare a Boolean variable with true.</td>
</tr>
<tr>
<td>frozen == false</td>
<td>!frozen</td>
<td>It is clearer to use ! than to compare with false.</td>
</tr>
</tbody>
</table>
Self Check 3.31

Suppose x and y are two integers. How do you test whether both of them are zero?

**Answer:** \[ x == 0 \land y == 0 \]
Self Check 3.32

How do you test whether at least one of them is zero?

**Answer:** \( x == 0 || y == 0 \)
Self Check 3.33

How do you test whether exactly one of them is zero?

**Answer:** \((x == 0 && y != 0) || (y == 0 && x != 0)\)
What is the value of `!!frozen`?

**Answer:** The same as the value of `frozen`. 
What is the advantage of using the type boolean rather than strings "false"/"true" or integers 0/1?

**Answer:** You are guaranteed that there are no other values. With strings or integers, you would need to check that no values such as "maybe" or −1 enter your calculations.
Common Error

- Combining Multiple Relational Operators

```java
if (0 <= temp <= 100) // Syntax error!
```

- This format is used in math, but not in Java!
- It requires two comparisons:

```java
if (0 <= temp && temp <= 100)
```

- This is also not allowed in Java:

```java
if (input == 1 || 2) // Syntax error!
```

- This also requires two comparisons:

```java
if (input == 1 || input == 2)
```
Common Error

- Confusing && and || Conditions
  - It is a surprisingly common error to confuse && and || conditions
  - A value lies between 0 and 100 if it is at least 0 and at most 100
  - It lies outside that range if it is less than 0 or greater than 100
  - There is no golden rule; you just have to think carefully
Short-Circuit Evaluation:  

- Combined conditions are evaluated from left to right
  - If the left half of an *and* condition is false, why look further?

```java
if (temp > 0 && temp < 100) {
    System.out.println("Liquid");
}
```

```java
if (quantity > 0 && price / quantity < 10)
```
Short-Circuit Evaluation: ||

- If the left half of the or is true, why look further?
- Java doesn’t!
- Don’t do these second:
  - Assignment
  - Output

```java
if (temp <= 0 || temp >= 100)
{
    System.out.println("Not Liquid");
}
```

Diagram:
- If either condition is true, the program will print "Not Liquid".
- Otherwise, it will check the next condition.
- If both conditions are false, the program will print "Done!".

At least one condition must be true for the program to print "Not Liquid".
De Morgan’s Law

- De Morgan’s law tells you how to negate && and || conditions:
  - !(A && B) is the same as !A || !B
  - !(A || B) is the same as !A && !B

- Example: Shipping is higher to AK and HI

```java
if (!(country.equals("USA")
    && !state.equals("AK")
    && !state.equals("HI")))
shippingCharge = 20.00;
```

- To simplify conditions with negations of and or or expressions, it is usually a good idea to apply De Morgan’s Law to move the negations to the innermost level.
Accepting user input is dangerous

- Consider the Elevator program:
  - The user may input an invalid character or value
  - Must be an integer
    - Scanner can help!
      - hasNextInt
        - True if integer
        - False if not

- Then range check value
  - We expect a floor number to be between 1 and 20
    - NOT 0, 13 or > 20

```java
if (in.hasNextInt())
{
    int floor = in.nextInt();
    // Process the input value
}
else
{
    System.out.println("Not integer.");
}
```
public class ElevatorSimulation2 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Floor: ");
        if (!in.hasNextInt()) {
            // Now we know that the user entered an integer
            int floor = in.nextInt();
            if (floor == 13) {
                System.out.println("Error: There is no thirteenth floor.");
            } else if (floor <= 0 || floor > 20) {
                System.out.println("Error: The floor must be between 1 and 20.");
            } else {
                // Now we know that the input is valid
            }
        }
    }
}
ElevatorSimulation2.java

```java
30 int actualFloor = floor;
31 if (floor > 13)
32 {
33     actualFloor = floor - 1;
34 }
35
36 System.out.println("The elevator will travel to the actual floor "+ actualFloor);
37 }
38 else
39 {
40     System.out.println("Error: Not an integer.");
41 }
42
43 }
44
45 ```

**Program Run**

Floor: 13
Error: There is no thirteenth floor.
Self Check 3.36

In the ElevatorSimulation2 program, what is the output when the input is
   a. 100?
   b. –1?
   c. 20?
   d. thirteen?

Answer:
   (a) Error: The floor must be between 1 and 20.
   (b) Error: The floor must be between 1 and 20.
   (c) 19
   (d) Error: Not an integer.
Self Check 3.37

Your task is to rewrite lines 19–26 of the ElevatorSimulation2 program so that there is a single if statement with a complex condition. What is the condition?

```java
if (...) {
    System.out.println("Error: Invalid floor number");
}
```

**Answer:** `floor == 13 || floor <= 0 || floor > 20`
In the Sherlock Holmes story “The Adventure of the Sussex Vampire”, the inimitable detective uttered these words: “Matilda Briggs was not the name of a young woman, Watson, ... It was a ship which is associated with the giant rat of Sumatra, a story for which the world is not yet prepared.” Over a hundred years later, researchers found giant rats in Western New Guinea, another part of Indonesia.

Suppose you are charged with writing a program that processes rat weights. It contains the statements

```java
    System.out.print("Enter weight in kg: ");
    double weight = in.nextDouble();
```

What input checks should you supply?
Self Check 3.38

**Answer:** Check for `in.hasNextDouble()` to make sure a researcher didn’t supply an input such as `oh my`. Check for `weight <= 0`, because any rat must surely have a positive weight. We don’t know how giant a rat could be, but the New Guinea rats weighed no more than 2 kg. A regular house rat (*rattus rattus*) weighs up to 0.2 kg, so we’ll say that any weight > 10 kg was surely an input error, perhaps confusing grams and kilograms. Thus, the checks are

```java
if (in.hasNextDouble())
{
    double weight = in.nextDouble();
    if (weight <= 0)
    {
        System.out.println("Error: Weight must be positive.");
    }
    else if (weight > 10)
    {
        System.out.println("Error: Weight > 10 kg.");
    }
    else
    {
        Process valid weight.
    }
}
else
{
    System.out.print("Error: Not a number");
}
```
Run the following test program and supply inputs 2 and three at the prompts. What happens? Why?

```java
import java.util.Scanner
public class Test
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        int m = in.nextInt();
        System.out.print("Enter another integer: ");
        int n = in.nextInt();
        System.out.println(m + " " + n);
    }
}
```

**Answer:** The second input fails, and the program terminates without printing anything.