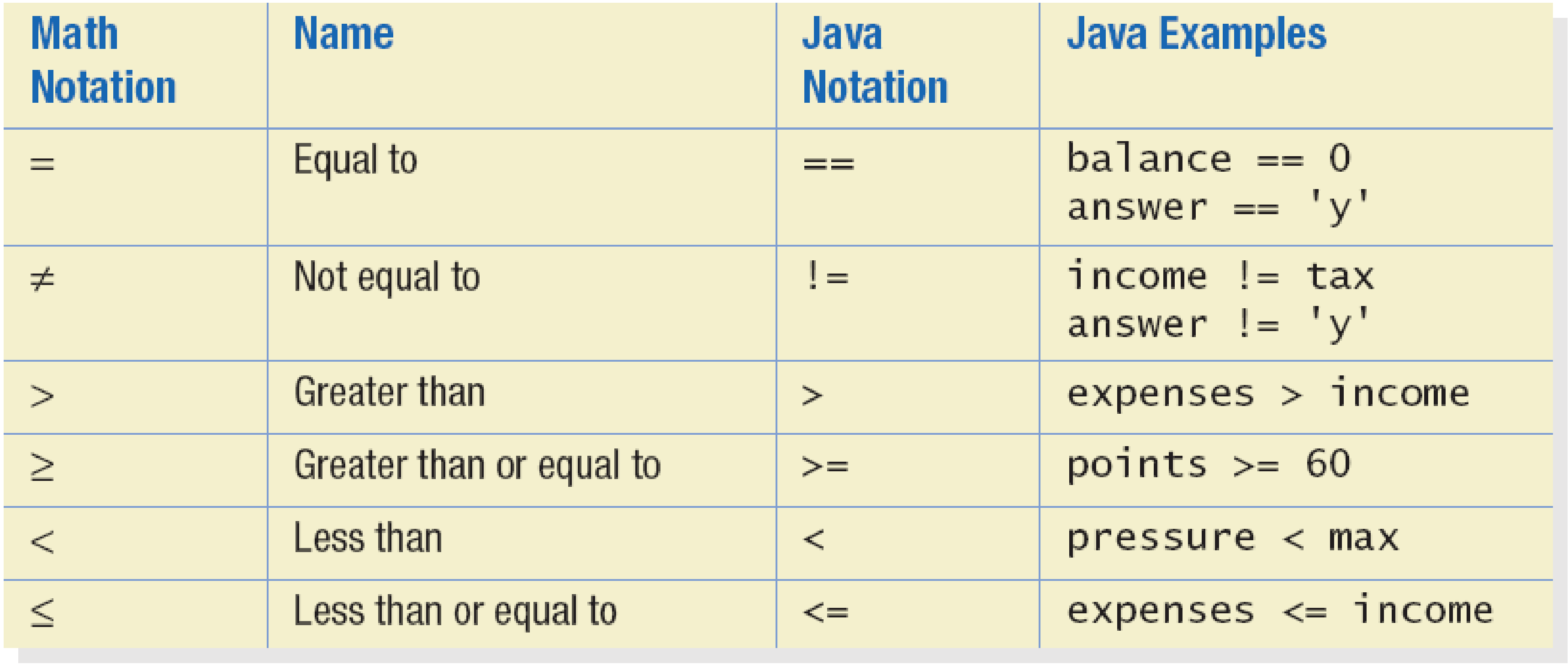
Java Comparison Operators

- Figure 3.4 Java Comparison Operators



Math Notation	Name	Java Notation	Java Examples

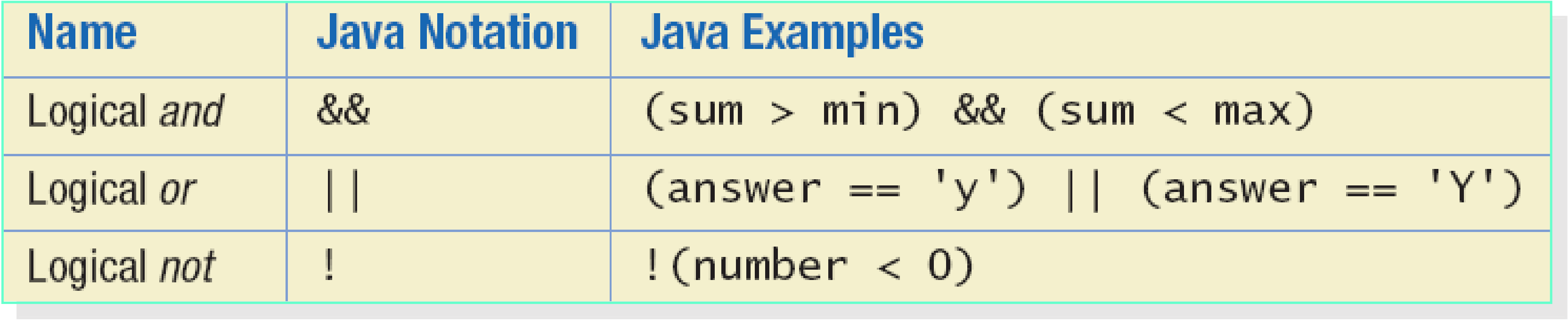
=	Equal to	`==`	`balance == 0`
			`answer == 'y'`

≠	Not equal to	`!=`	`income != tax`
			`answer != 'y'`

>	Greater than	`>`	`expenses > income`
≥	Greater than or equal to	`>=`	`points >= 60`
<	Less than	`<`	`pressure < max`
≤	Less than or equal to	`<=`	`expenses <= income`

Java Logical Operators

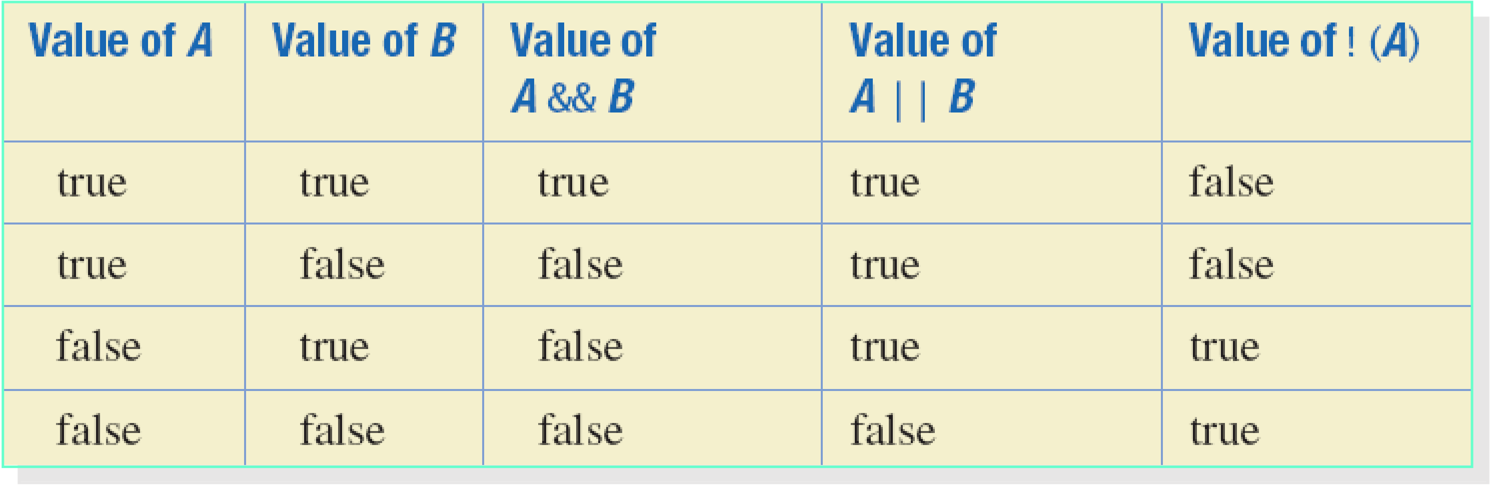
- Figure 3.6



Name	Java Notation	Java Examples
Logical *and*	`&&`	`(sum > min) && (sum < max)`
Logical *or*	`||`	`(answer == 'y') || (answer == 'Y')`
Logical *not*	`!`	`!(number < 0)`

Boolean Operators

- FIGURE 3.7 The Effect of the Boolean Operators && (and), || (or), and ! (not) on Boolean values



Value of *A*	Value of *B*	Value of *A* && *B*	Value of *A* || *B*	Value of ! (*A*)
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Using ==

- == is appropriate for determining if two integers or characters have the same value.

  ```
  if (a == 3)
  ```

  where a is an integer type

- == is **not** appropriate for determining if two floating points values are equal.

Using `==`

- `==` is not appropriate for determining if two objects have the same value.
  - `if (s1 == s2),` where `s1` and `s2` refer to strings, determines only if s1 and s2 refer the a common memory location.
  - If `s1` and `s2` refer to strings with identical sequences of characters, but stored in different memory locations, `(s1 == s2)` is false.

Using ==

- To test the equality of objects of class String, use method **equals**.

```
s1.equals(s2)
```

or

```
s2.equals(s1)
```

- To test for equality ignoring case, use method **equalsIgnoreCase**.

```
("Hello".equalsIgnoreCase("hello"))
```

equals and **equalsIgnoreCase**

- Syntax

```
String.equals(Other_String)
String.equalsIgnoreCase(Other_String)
```

Nested `if-else` Statements

- An `if-else` statement can contain any sort of statement within it.
- In particular, it can contain another `if-else` statement.
  - An `if-else` may be nested within the "if" part.
  - An `if-else` may be nested within the "else" part.
  - An `if-else` may be nested within both parts.

Nested Statements

- Syntax

```
if (Boolean_Expression_1)
   if (Boolean_Expression_2)
      Statement_1)
   else
      Statement_2)
else
  if (Boolean_Expression_3)
      Statement_3)
   else
      Statement_4);
```

Nested Statements

- Each **`else`** is paired with the nearest unmatched **`if`**.
- When used properly, indentation communicates which **`if`** goes with which **`else`**.
- Braces can be used like parentheses to group statements.

Nested Statements

- Subtly different forms

First Form

```
if (a > b)
{
    if (c > d)
        e = f
}
    else
        g = h;
```

Second Form

```
if (a > b)
   if (c > d)
        e = f
    else
        g = h;

// oops
```

Compound Statements

- When a list of statements is enclosed in braces (`{}`), they form a single compound statement.
- Syntax

```
{
    Statement_1;
    Statement_2;
  …
}
```

Compound Statements

- A compound statement can be used wherever a statement can be used.
- Example

```
if (total > 10)
{
   sum = sum + total;
   total = 0;
}
```

Multibranch `if-else` Statements

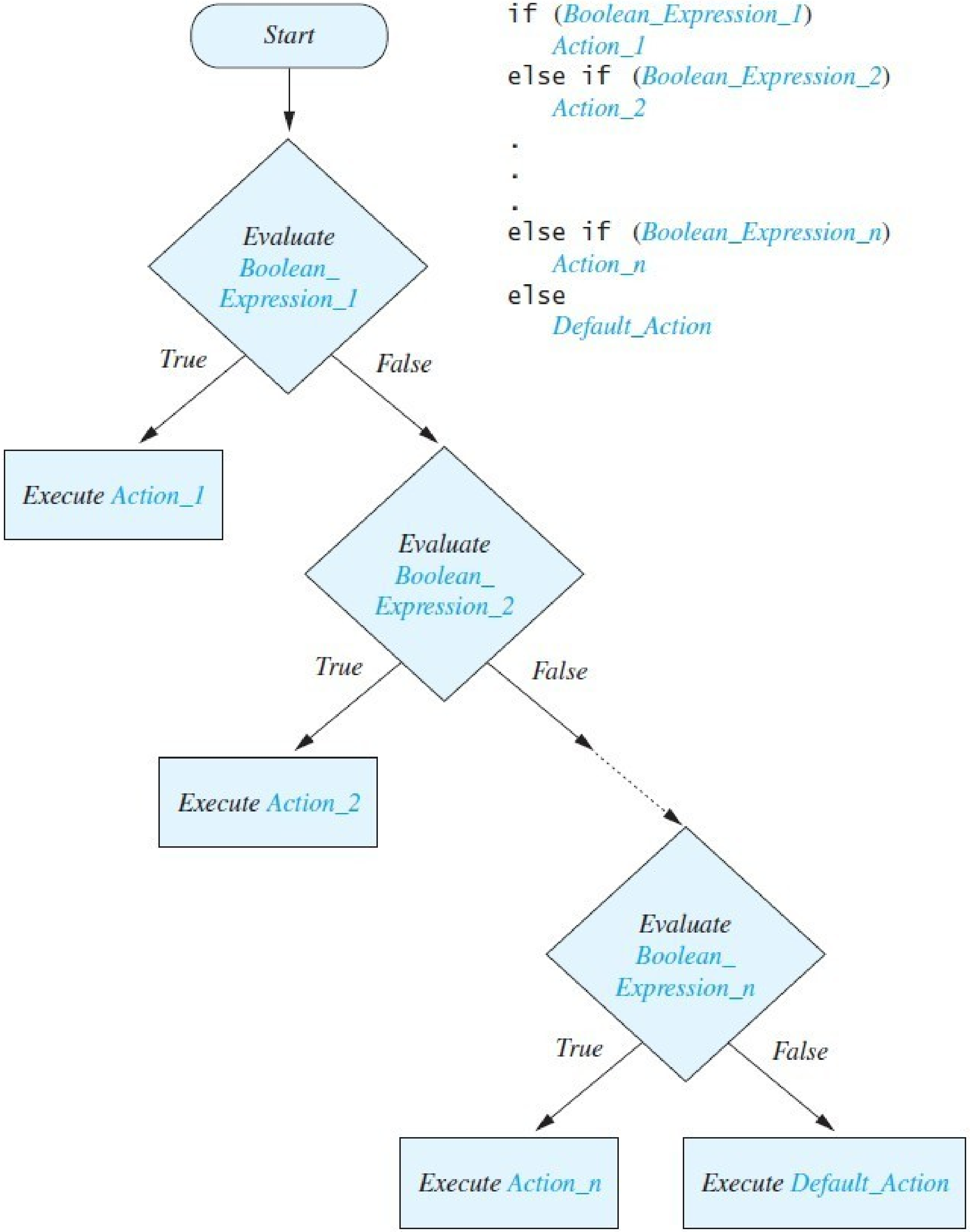
- Syntax

```
if (Boolean_Expression_1)
   Statement_1
else if (Boolean_Expression_2)
   Statement_2
else if (Boolean_Expression_3)
   Statement_3
else if …
else
   Default_Statement
```

Multibranch `if-else` Statements

- Figure 3.8 Semantics

```
if (Boolean_Expression_1)
    Action_1
else if (Boolean_Expression_2)
    Action_2
.
.
.
else if (Boolean_Expression_n)
    Action_n
else
    Default_Action
```


Start
Evaluate Boolean_ Expression_1
True
False
Execute Action_1
Evaluate Boolean_ Expression_2
True
False
Execute Action_2
Evaluate Boolean_ Expression_n
True
False
Execute Action_n
Execute Default_Action