

Chapter 5 – Decisions

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Statement Types

- Simple statement:

```
balance = balance - amount;
```

- Compound statement:

```
if (balance >= amount) balance = balance - amount;
```

Also loop statements — Chapter 6

- Block statement:

```
{
    double newBalance = balance - amount;
    balance = newBalance;
}
```

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Syntax 5.1 The `if` Statement

Syntax

```
if (condition)
    statement
else
    statement2
```

Example

A condition that is true or false.
Often uses relational operators: == != < <= > >=

Don't put a semicolon here!

Braces are not required if the body contains a single statement.

```
if (amount <= balance)
    balance = balance - amount;
else
    System.out.println("Insufficient funds");
    balance = balance - OVERDRAFT_PENALTY;
```

Omit the else branch if there is nothing to do.

Lining up braces is a good idea.

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

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

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Comparing Values: Relational Operators

- The `==` denotes equality testing:

```
a = 5; // Assign 5 to a
if (a == 5) ... // Test whether a equals 5
```

- Relational operators have lower precedence than arithmetic operators:

```
amount + fee <= balance
```

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Comparing Floating-Point Numbers

- Consider this code:

```
double r = Math.sqrt(2);
double d = r * r - 2;
if (d == 0)
    System.out.println("sqrt(2) squared minus 2 is 0");
else
    System.out.println("sqrt(2) squared minus 2 is not 0 but "
        + d);
```

- It prints:

```
sqrt(2) squared minus 2 is not 0 but 4.440892098500626E-16
```

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Comparing Floating-Point Numbers

- To avoid roundoff errors, don't use `==` to compare floating-point numbers
- To compare floating-point numbers test whether they are *close enough*: $|x - y| \leq \epsilon$

```
final double EPSILON = 1E-14;
if (Math.abs(x - y) <= EPSILON)
    // x is approximately equal to y
```

- ϵ is a small number such as 10^{-14}

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Comparing Strings

- To test whether two strings are equal to each other, use `equals` method:

```
if (string1.equals(string2)) . . .
```

- Don't use `==` for strings!

```
if (string1 == string2) // Not useful
```

- `==` tests identity, `equals` tests equal contents

- Case insensitive test:

```
if (string1.equalsIgnoreCase(string2))
```

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Comparing Objects

- `==` tests for identity, `equals` for identical content

```
Rectangle box1 = new Rectangle(5, 10, 20, 30);
Rectangle box2 = box1;
Rectangle box3 = new Rectangle(5, 10, 20, 30);
```

- `box1 != box3`, **but** `box1.equals(box3)`

- `box1 == box2`

- **Caveat:** `equals` must be defined for the class

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Object Comparison

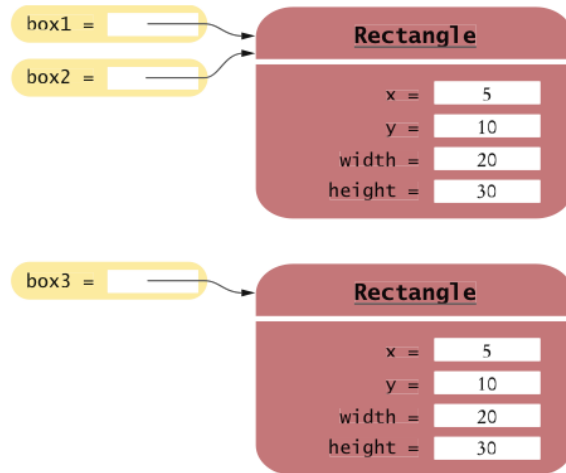


Figure 4
Comparing Object References

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Testing for `null`

- `null` reference refers to no object:

```
String middleInitial = null; // Not set
if ( ... )
    middleInitial = middleName.substring(0, 1);
```

- Can be used in tests:

```
if (middleInitial == null)
    System.out.println(firstName + " " + lastName);
else
    System.out.println(firstName + " " + middleInitial +
        ". " + lastName);
```

- Use `==`, not `equals`, to test for `null`
- `null` is not the same as the empty string `""`

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Multiple Alternatives: Sequences of Comparisons

- ```
if (condition1)
 statement1;
else if (condition2)
 statement2;
...
else
 statement4;
```

- The first matching condition is executed

- Order matters:

```
if (richter >= 0) // always passes
 r = "Generally not felt by people";
else if (richter >= 3.5) // not tested
 r = "Felt by many people, no destruction";
...

```

*Big Java* by Cay Horstmann  
Copyright © 2009 by John Wiley & Sons. All rights reserved.

## Multiple Alternatives: Sequences of Comparisons

---

- Don't omit `else`:

```
if (richter >= 8.0)
 r = "Most structures fall";
if (richter >= 7.0) // omitted else--ERROR
 r = "Many buildings destroyed";

```

*Big Java* by Cay Horstmann  
Copyright © 2009 by John Wiley & Sons. All rights reserved.

## Multiple Alternatives: Nested Branches

---

- Branch inside another branch:

```
if (condition1)
{
 if (condition1a)
 statement1a;
 else
 statement1b;
}
else
 statement2;
```

*Big Java* by Cay Horstmann  
Copyright © 2009 by John Wiley & Sons. All rights reserved.

## Self Check 5.5

---

The `if/else/else` statement for the earthquake strength first tested for higher values, then descended to lower values. Can you reverse that order?

**Answer:** Yes, if you also reverse the comparisons:

```
if (richter < 3.5)
 r = "Generally not felt by people";
else if (richter < 4.5)
 r = "Felt by many people, no destruction";
else if (richter < 6.0)
 r = "Damage to poorly constructed buildings";
...

```

*Big Java* by Cay Horstmann  
Copyright © 2009 by John Wiley & Sons. All rights reserved.

## Using Boolean Expressions: The `boolean` Type



- George Boole (1815-1864): pioneer in the study of logic
- value of expression `amount < 1000` is `true` or `false`
- `boolean` type: one of these 2 truth values

*Big Java* by Cay Horstmann  
Copyright © 2009 by John Wiley & Sons. All rights reserved.

## Using Boolean Expressions: Predicate Method

- A predicate method returns a `boolean` value:

```
public boolean isOverdrawn()
{
 return balance < 0;
}
```

- Use in conditions:

```
if (harrysChecking.isOverdrawn())
```

- Useful predicate methods in `Character` class:

```
isDigit
isLetter
isUpperCase
isLowerCase
```

*Big Java* by Cay Horstmann  
Copyright © 2009 by John Wiley & Sons. All rights reserved.



## Using Boolean Expressions: Predicate Method

---

- `if (Character.isUpperCase(ch)) ...`
- **Useful predicate methods in Scanner class:** `hasNextInt()` and `hasNextDouble()`:  

```
if (in.hasNextInt()) n = in.nextInt();
```

*Big Java* by Cay Horstmann  
 Copyright © 2009 by John Wiley & Sons. All rights reserved.

## Using Boolean Expressions: The Boolean Operators



---

- `&&` and
- `||` or
- `!` not
- `if (0 < amount && amount < 1000) . . .`
- `if (input.equals("S") || input.equals("M")) . . .`  
 .
- `if (!input.equals("S")) . . .`

*Big Java* by Cay Horstmann  
 Copyright © 2009 by John Wiley & Sons. All rights reserved.

## Boolean Operators

Table 3 Boolean Operators

| Expression                                                                                                            | Value                                                    | Comment                                                                                                                                             |
|-----------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>0 &lt; 200 &amp;&amp; 200 &lt; 100</code>                                                                       | false                                                    | Only the first condition is true.                                                                                                                   |
| <code>0 &lt; 200    200 &lt; 100</code>                                                                               | true                                                     | The first condition is true.                                                                                                                        |
| <code>0 &lt; 200    100 &lt; 200</code>                                                                               | true                                                     | The <code>  </code> is not a test for “either-or”. If both conditions are true, the result is true.                                                 |
|  <code>0 &lt; 100 &lt; 200</code>    | Syntax error                                             | <b>Error:</b> The expression <code>0 &lt; 100</code> is true, which cannot be compared against 200.                                                 |
|  <code>0 &lt; x    x &lt; 100</code> | true                                                     | <b>Error:</b> This condition is always true. The programmer probably intended <code>0 &lt; x &amp;&amp; x &lt; 100</code> . (See Common Error 5.5). |
| <code>0 &lt; x &amp;&amp; x &lt; 100    x == -1</code>                                                                | <code>(0 &lt; x &amp;&amp; x &lt; 100)    x == -1</code> | The <code>&amp;&amp;</code> operator binds more strongly than the <code>  </code> operator.                                                         |
| <code>!(0 &lt; 200)</code>                                                                                            | false                                                    | <code>0 &lt; 200</code> is true, therefore its negation is false.                                                                                   |
| <code>frozen == true</code>                                                                                           | frozen                                                   | There is no need to compare a Boolean variable with true.                                                                                           |
| <code>frozen == false</code>                                                                                          | !frozen                                                  | It is clearer to use <code>!</code> than to compare with false.                                                                                     |

*Big Java* by Cay Horstmann  
Copyright © 2009 by John Wiley & Sons. All rights reserved.

## Truth Tables

| A     | B          | A && B |
|-------|------------|--------|
| true  | true       | true   |
| true  | false      | false  |
| false | <i>Any</i> | false  |

| A     | B          | A    B |
|-------|------------|--------|
| true  | <i>Any</i> | true   |
| false | true       | true   |
| false | false      | false  |

| A     | !A    |
|-------|-------|
| true  | false |
| false | true  |

*Big Java* by Cay Horstmann  
Copyright © 2009 by John Wiley & Sons. All rights reserved.

## Using Boolean Variables

---

- `private boolean married;`

- Set to truth value:

```
married = input.equals("M");
```

- Use in conditions:

```
if (married) ... else ...
if (!married) ...
```

- Also called *flag*

- It is considered gauche to write a test such as

```
if (married == true) ... // Don't
```

- Just use the simpler test

```
if (married) ...
```

*Big Java* by Cay Horstmann  
Copyright © 2009 by John Wiley & Sons. All rights reserved.

## Self Check 5.7

---

When does the statement

```
system.out.println (x > 0 || x < 0);
```

print `false`?

**Answer:** When `x` is zero.

*Big Java* by Cay Horstmann  
Copyright © 2009 by John Wiley & Sons. All rights reserved.

## Self Check 5.8

---

Rewrite the following expression, avoiding the comparison with `false`:

```
if (character.isDigit(ch) == false) ...
```

**Answer:** `if (!Character.isDigit(ch)) ...`