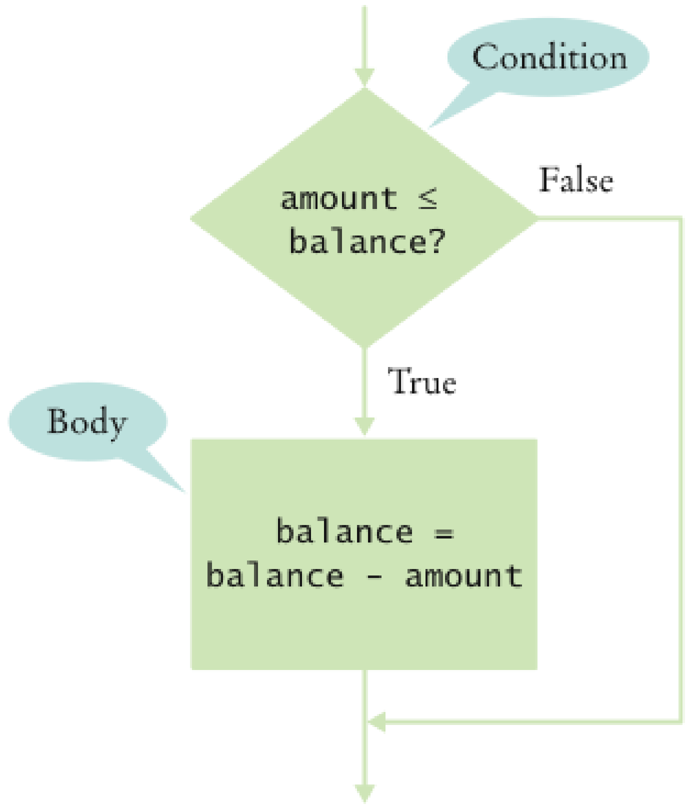
Chapter 5 – Using Objects

1. The *if* Statement
2. The *if/else* Statement
3. Comparing Values: Relational Operators
4. Comparing Values: Relational Operators
5. Comparing *Objects*
6. Testing for *null*
7. Multiple Alternatives: Sequences of Comparisons
8. Using Boolean Expressions

The `if` Statement

- The `if` statement lets a program carry out different actions depending on a condition

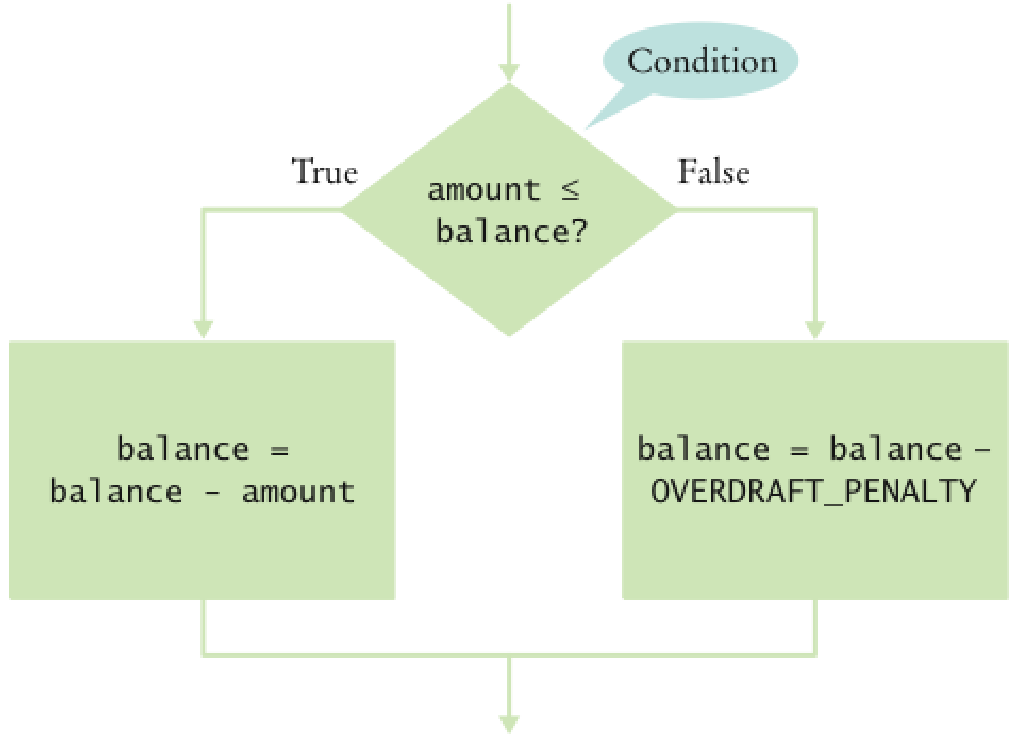
```
if (amount <= balance)
   balance = balance - amount;
```



The `if/else` Statement

- The `if/else` statement lets a program carry out different actions depending on a condition (True or False)

```
if (amount <= balance)
      balance = balance - amount;
else
       balance = balance - OVERDRAFT_PENALTY
```


Condition
True
False
amount ≤ balance?
balance = balance - amount
balance = balance - OVERDRAFT_PENALTY

The `if/else` Statement

Syntax

```
if (condition)
   statement
```

```
if (condition)
   statement1
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;
}
```

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.

Omit the `else` branch if there is nothing to do.

Lining up braces is a good idea.

Comparing Values: Relational Operators

- Relational operators compare values

Java	Math Notation	Description
`>`	$>$	Greater than
`>=`	$\geq$	Greater than or equal
`<`	$<$	Less than
`<=`	$\leq$	Less than or equal
`==`	$=$	Equal
`!=`	$\neq$	Not equal

Comparing Values: Relational Operators

- The `==` denotes equality testing:

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

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

Comparing Strings (Lexicographic Comparison)

- `string1.compareTo(string2) < 0` means:

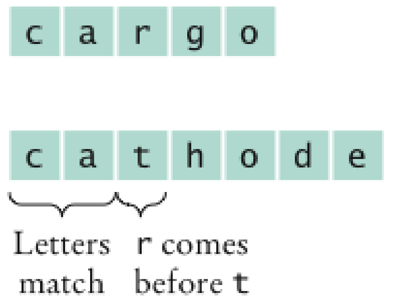
  `string1` comes before `string2` in the dictionary

- `string1.compareTo(string2) > 0` means:

  `string1` comes after `string2`

- `string1.compareTo(string2) == 0` means:

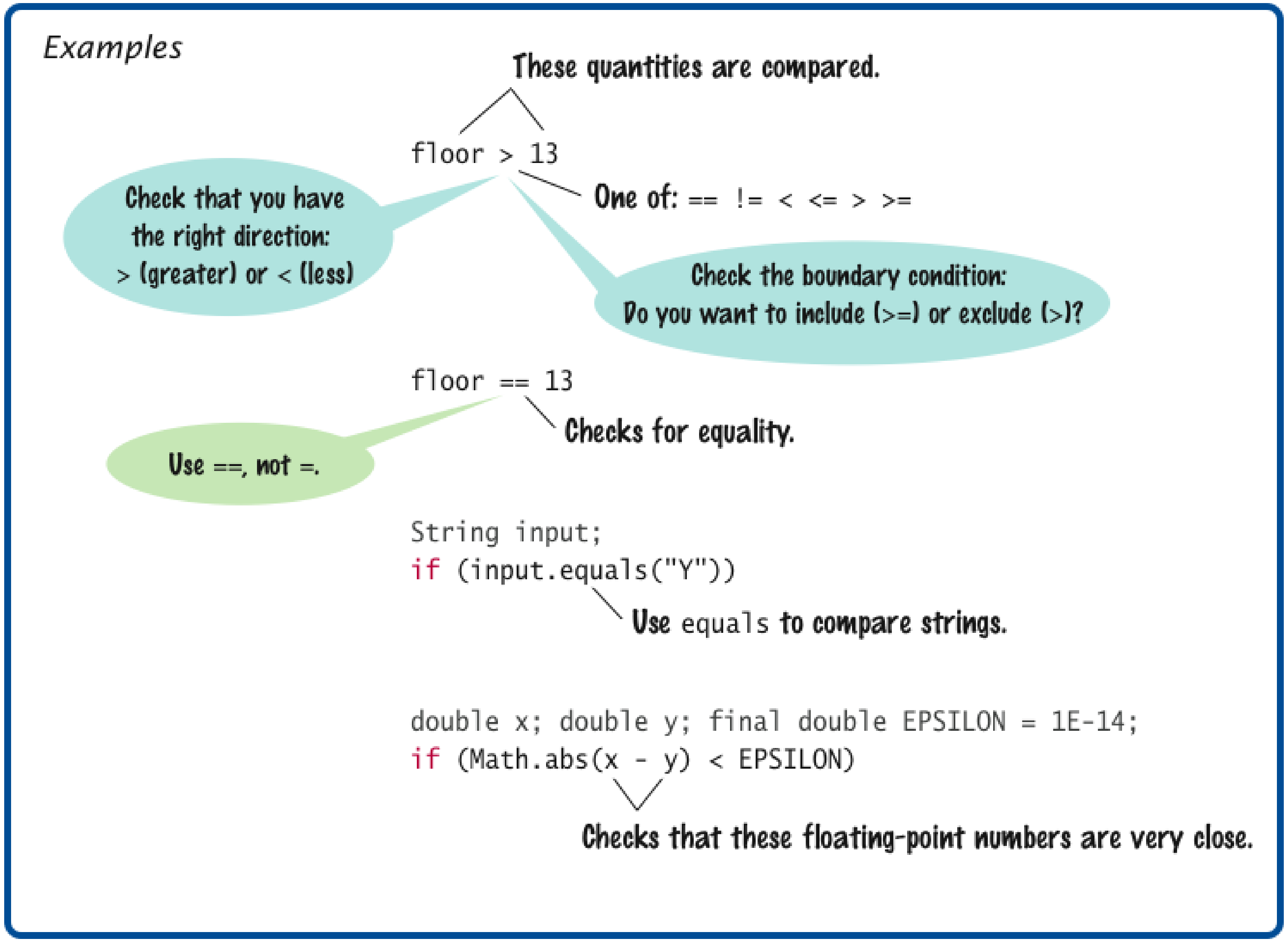
  `string1` equals `string2`



`"car"` comes before `"cargo"`

All uppercase letters come before lowercase: `"Hello"` comes before `"car"`

Comparisons

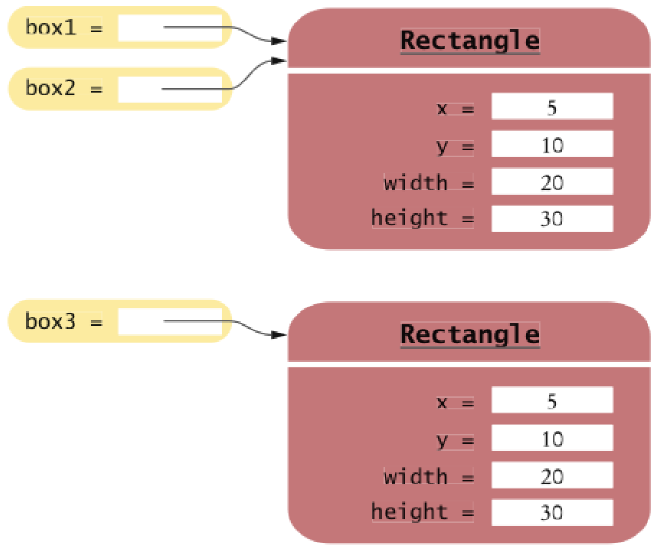

Examples
These quantities are compared.
One of: == != < <= > >=
Check that you have the right direction: > (greater) or < (less)
Check the boundary condition: Do you want to include (>=) or exclude (>)?
Checks for equality.
Use ==, not =.
Use equals to compare strings.
Checks that these floating-point numbers are very close.

```
floor > 13

floor == 13

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

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

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` (True)
- `box1.equals(box3)` (True)
- `box1 == box2` (True)

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

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 `""`

Multiple Alternatives: Sequences of Comparisons

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

- The first matching condition is executed

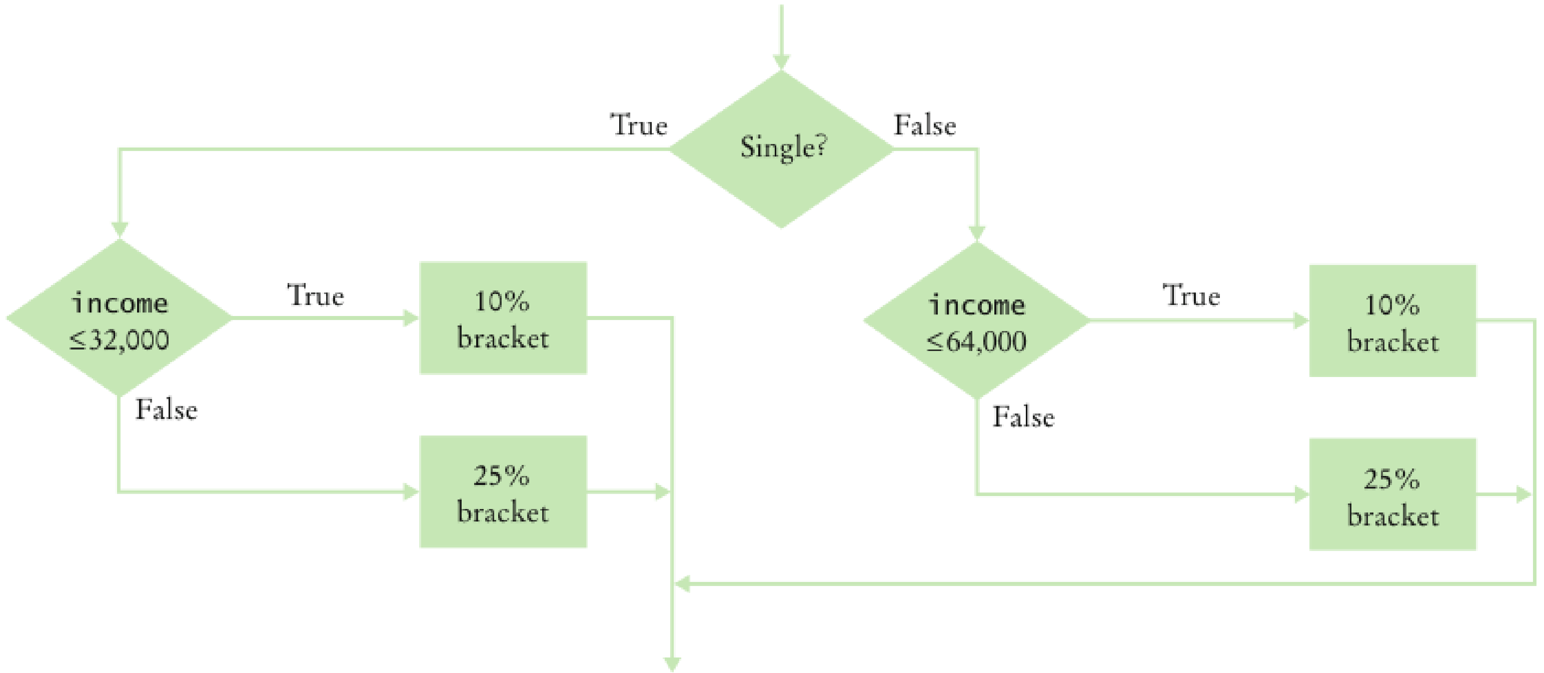
- Don't omit `else`:

Multiple Alternatives: Nested Branches

- Branch inside another branch:

```
if (condition_1)
{
    if (condition_1a)
        statement_1a;
    else
        statement_1b;
}
else
    statement_2;
```

Example



Using Boolean Expressions: The `boolean` Type

- value of expression `amount < 1000` is `true` or `false`
- `boolean` type: one of these 2 truth values

- A predicate method returns a `boolean` value:

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

- Use in conditions:

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

- Useful predicate methods in `Scanner` class: `hasNextInt()` `hasNextDouble()`

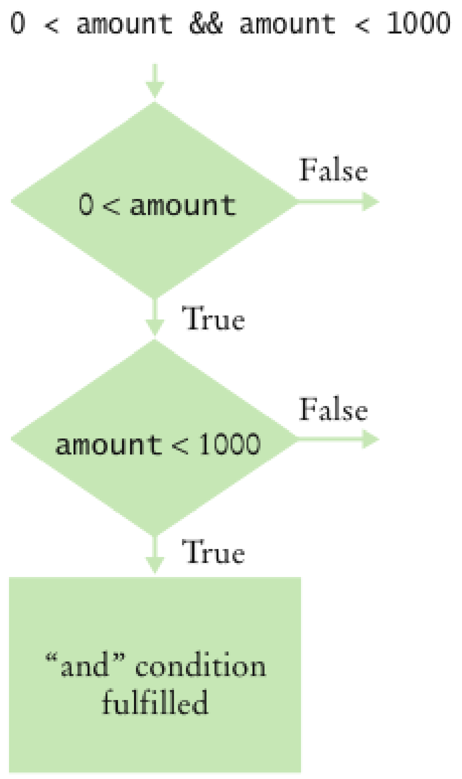
```
if (in.hasNextInt()) n = in.nextInt();
```

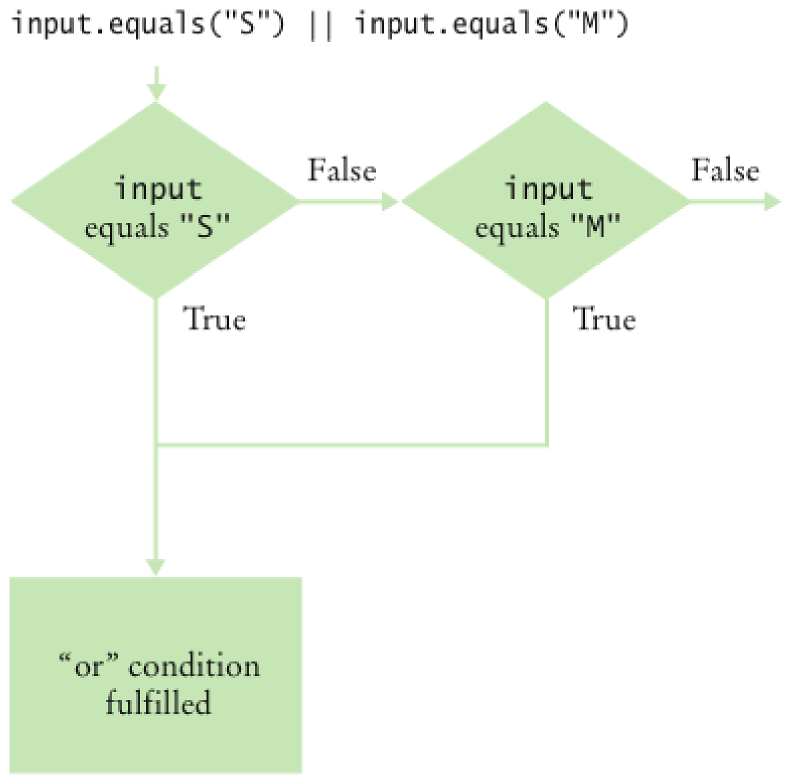
Using Boolean Expressions: The Boolean Operators

- `&&` and
- `||` or
- `!` not

Example

- `if (0 < amount && amount < 1000) . . .`
- `if (input.equals("S") ||input.equals("M")) . . .`
- `if (!input.equals("S")) . . .`





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*