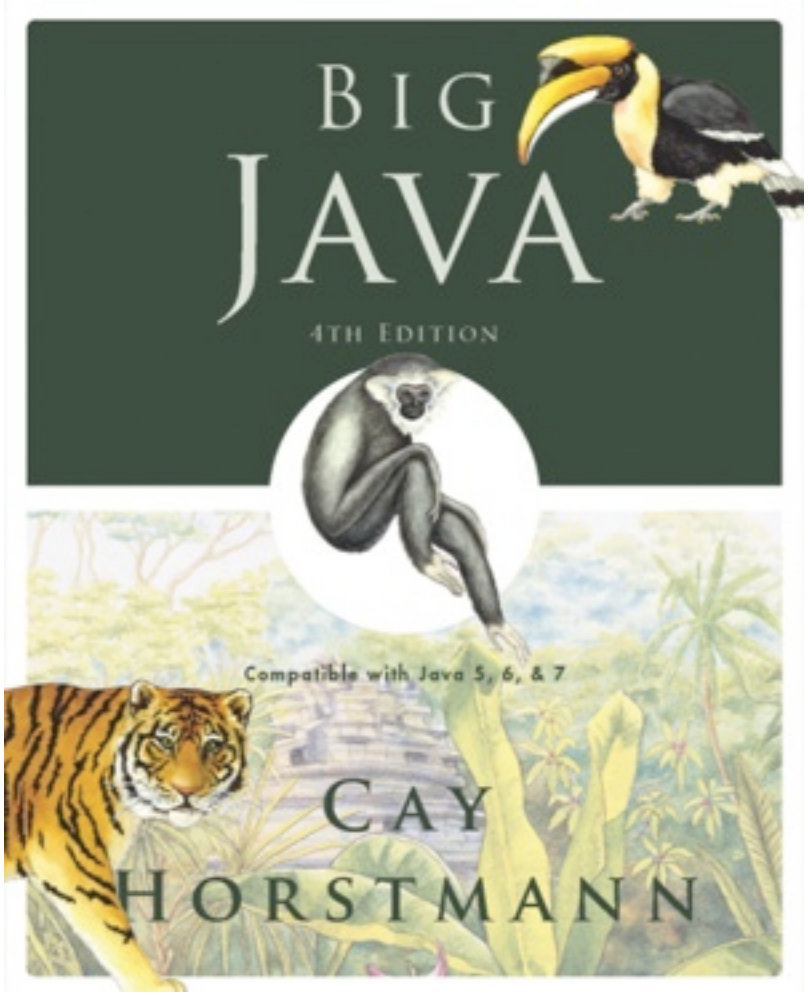
ICOM 4015: Advanced Programming

Lecture 5

Reading: Chapter Five: Decisions



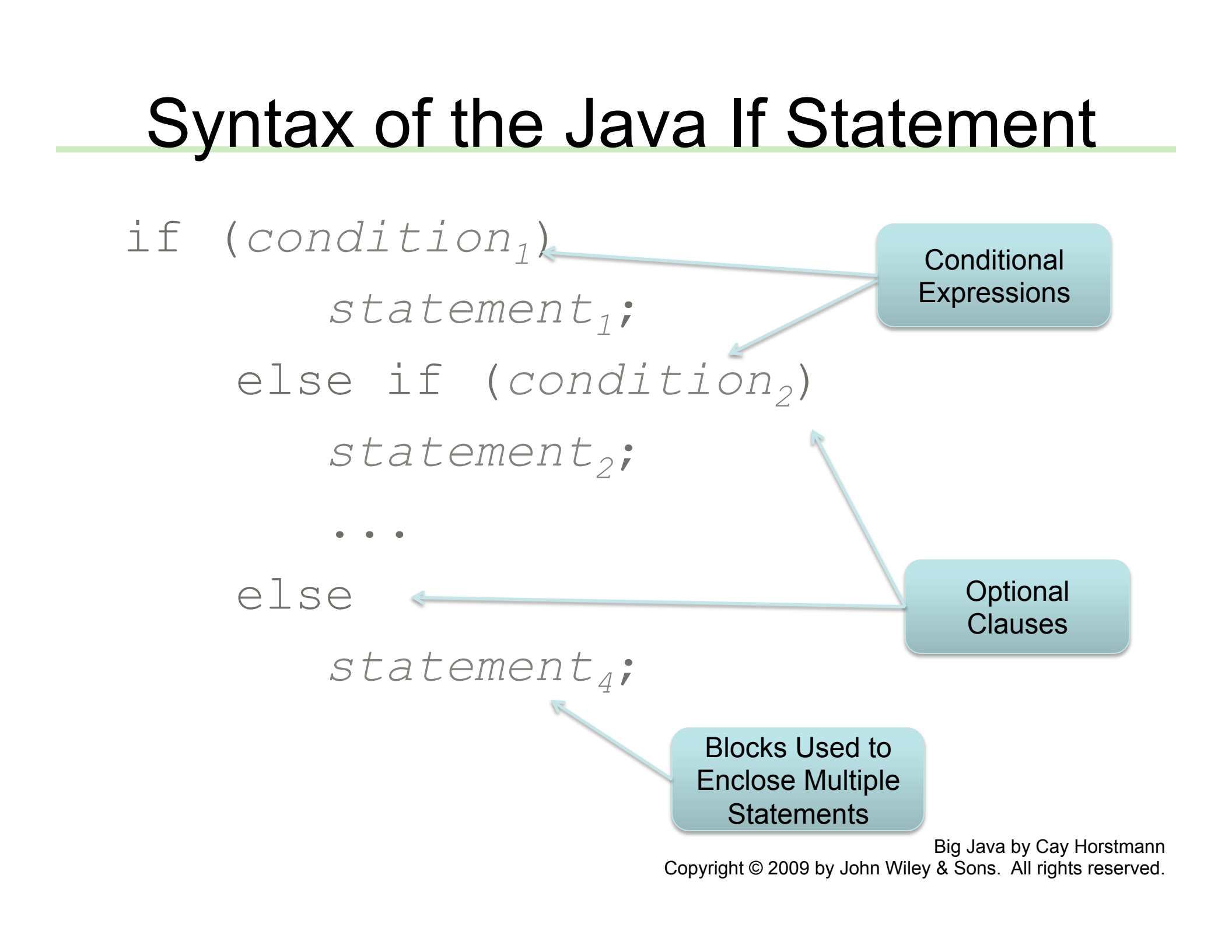
Chapter 5 – Decisions

Chapter Goals

- To be able to implement decisions using `if` statements
- To understand how to group statements into blocks
- To learn how to compare integers, floating-point numbers, strings, and objects
- To recognize the correct ordering of decisions in multiple branches
- To program conditions using Boolean operators and variables
- To understand the importance of test coverage

Syntax of the Java If Statement

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



The `if` Statement

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

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

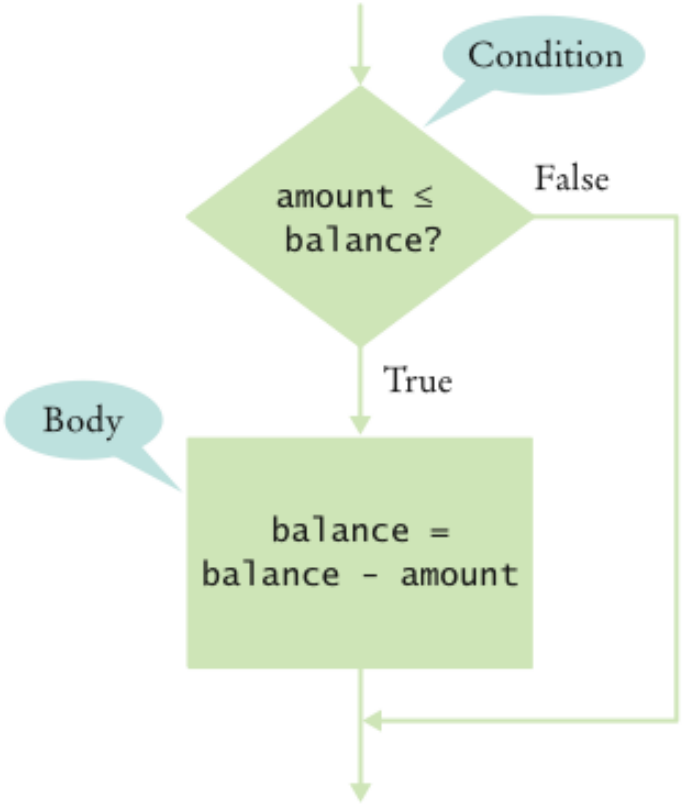


Figure 1
Flowchart for an `if` Statement

The `if/else` Statement

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

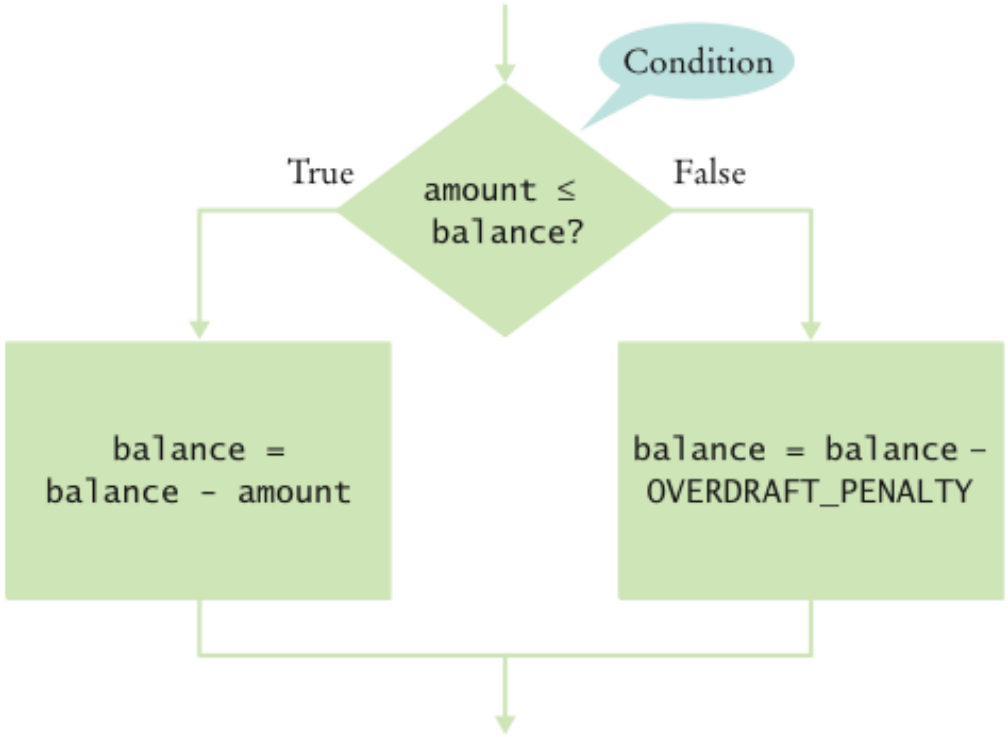


Figure 2
Flowchart for an `if/else` Statement

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;
}
```

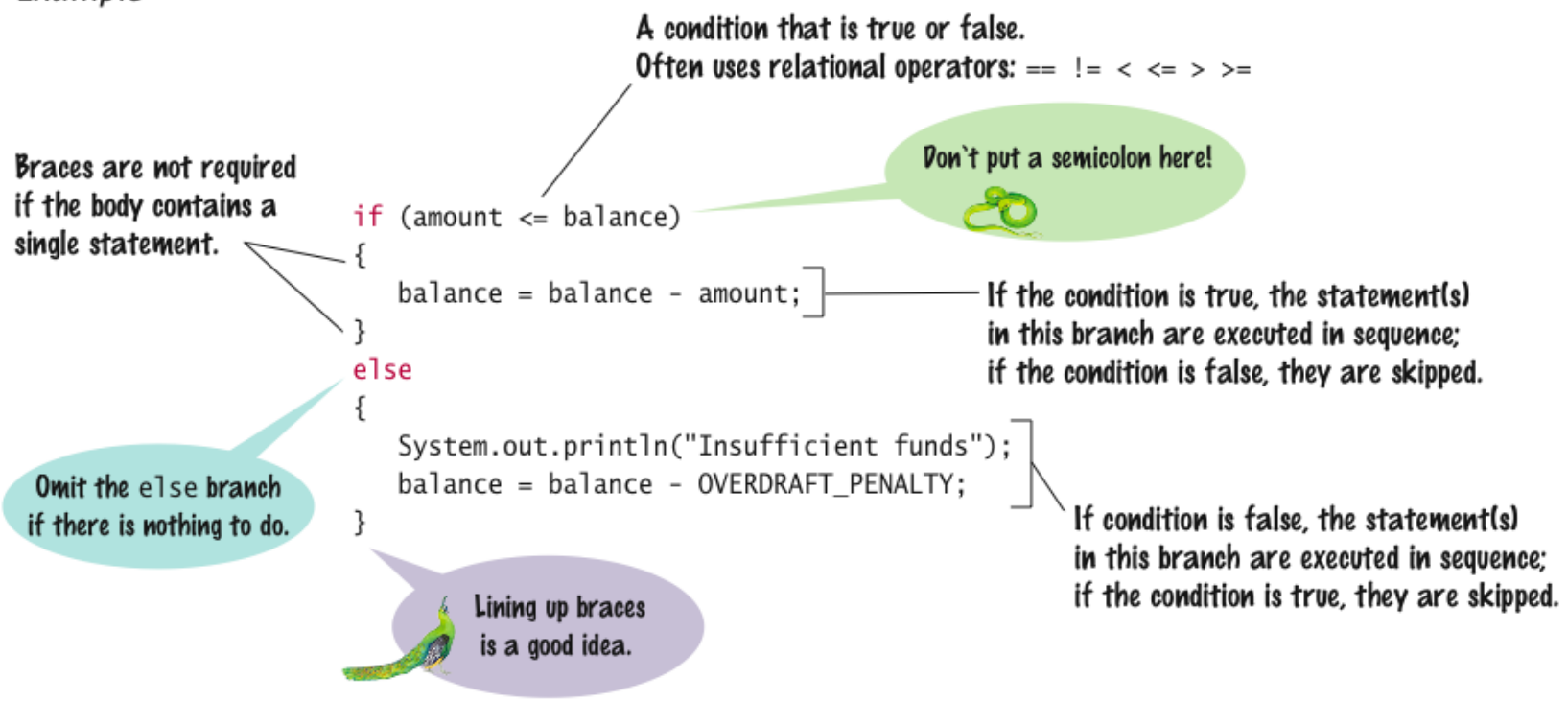
Syntax 5.1 The `if` Statement

Syntax

```
if (condition)
   statement
```

```
if (condition)
   statement1
else
   statement2
```

Example



Self Check 5.1

Why did we use the condition `amount <= balance` and not `amount < balance` in the example for the `if/else` statement?

Answer: If the withdrawal amount equals the balance, the result should be a zero balance and no penalty.

Self Check 5.2

What is logically wrong with the statement

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

and how do you fix it?

Answer: Only the first assignment statement is part of the `if` statement. Use braces to group both assignment statements into a block statement.

Conditional (Boolean) Expressions

- Boolean literals: True, False
- Relational operations: <, <, <=, >=, !=
- Boolean methods: equals
- Logical operations: && (and), || (or) , ! (not)
- Boolean variables

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.

Comparing Values: Relational Operators

- Relational operators compare values

Java	Math Notation	Description
`>`	>	Greater than
`>=`	≥	Greater than or equal
`<`	<	Less than
`<=`	≤	Less than or equal
`==`	=	Equal
`!=`	≠	Not equal

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
```

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
```

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*: $|x - y| \leq \varepsilon$

```
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}

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))
```

Comparing Strings

- `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"`

Lexicographic Comparison

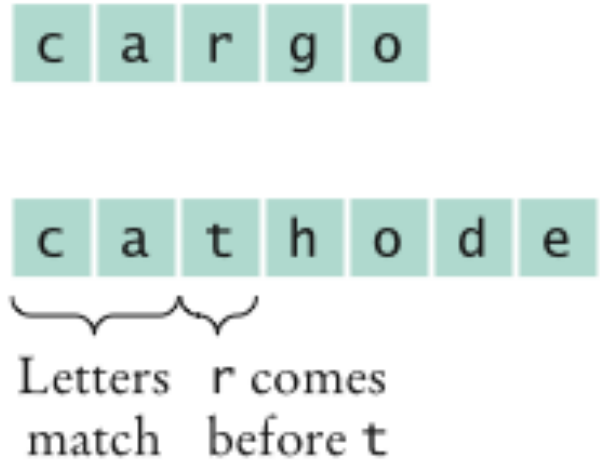
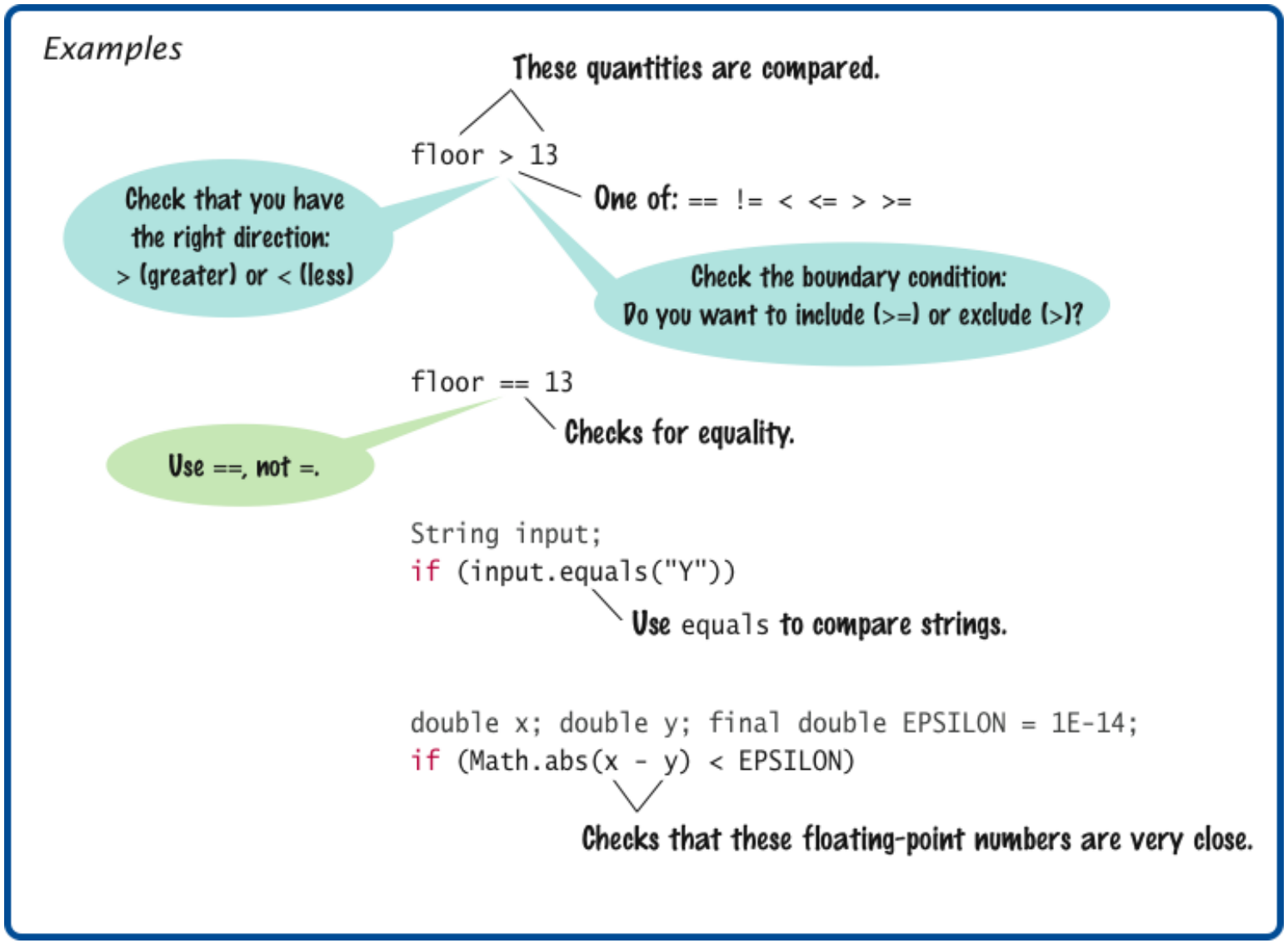


Figure 3
Lexicographic Comparison

Syntax 5.2 Comparisons



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

Object Comparison

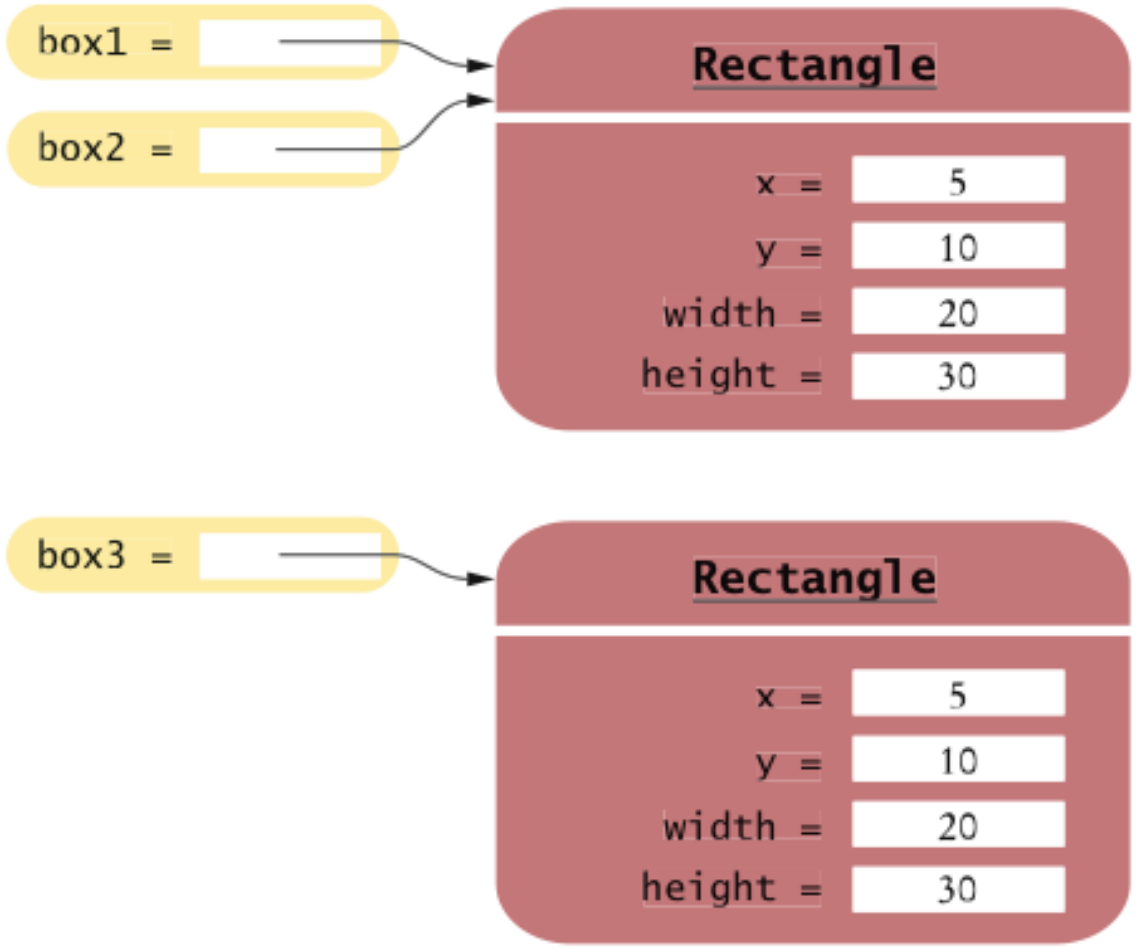


Figure 4
Comparing Object References

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 `""`

Relational Operator Examples

Table 1 Relational Operator Examples

Expression	Value	Comment
`3 <= 4`	`true`	3 is less than 4; <= tests for "less than or equal".
`3 =< 4`	**Error**	The "less than or equal" operator is <=, not =<, with the "less than" symbol first.
`3 > 4`	`false`	> is the opposite of <=.
`4 < 4`	`false`	The left-hand side must be strictly smaller than the right-hand side.
`4 <= 4`	`true`	Both sides are equal; <= tests for "less than or equal".
`3 == 5 - 2`	`true`	== tests for equality.
`3 != 5 - 1`	`true`	!= tests for inequality. It is true that 3 is not 5 – 1.
`3 = 6 / 2`	**Error**	Use == to test for equality.
`1.0 / 3.0 == 0.333333333`	`false`	Although the values are very close to one another, they are not exactly equal. See Common Error 4.3.
`"10" > 5`	**Error**	You cannot compare a string to a number.
`"Tomato".substring(0, 3).equals("Tom")`	`true`	Always use the `equals` method to check whether two strings have the same contents.
`"Tomato".substring(0, 3) == ("Tom")`	`false`	Never use == to compare strings; it only checks whether the strings are stored in the same location. See Common Error 5.2 on page 180.
`"Tom".equalsIgnoreCase("TOM")`	`true`	Use the `equalsIgnoreCase` method if you don't want to distinguish between uppercase and lowercase letters.

Self Check 5.3

What is the value of `s.length()` if `s` is

a. the empty string `""`?
b. the string `" "` containing a space?
c. `null`?

Answer: (a) 0; (b) 1; (c) an exception occurs.

Self Check 5.4

Which of the following comparisons are syntactically incorrect? Which of them are syntactically correct, but logically questionable?

```
String a = "1";
String b = "one";
double x = 1;
double y = 3 * (1.0 / 3);
```

a. a == "1"
b. a == null
c. a.equals("")
d. a == b
e. a == x
f. x == y
g. x - y == null
h. x.equals(y)

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.
  ```

&& and || Operators

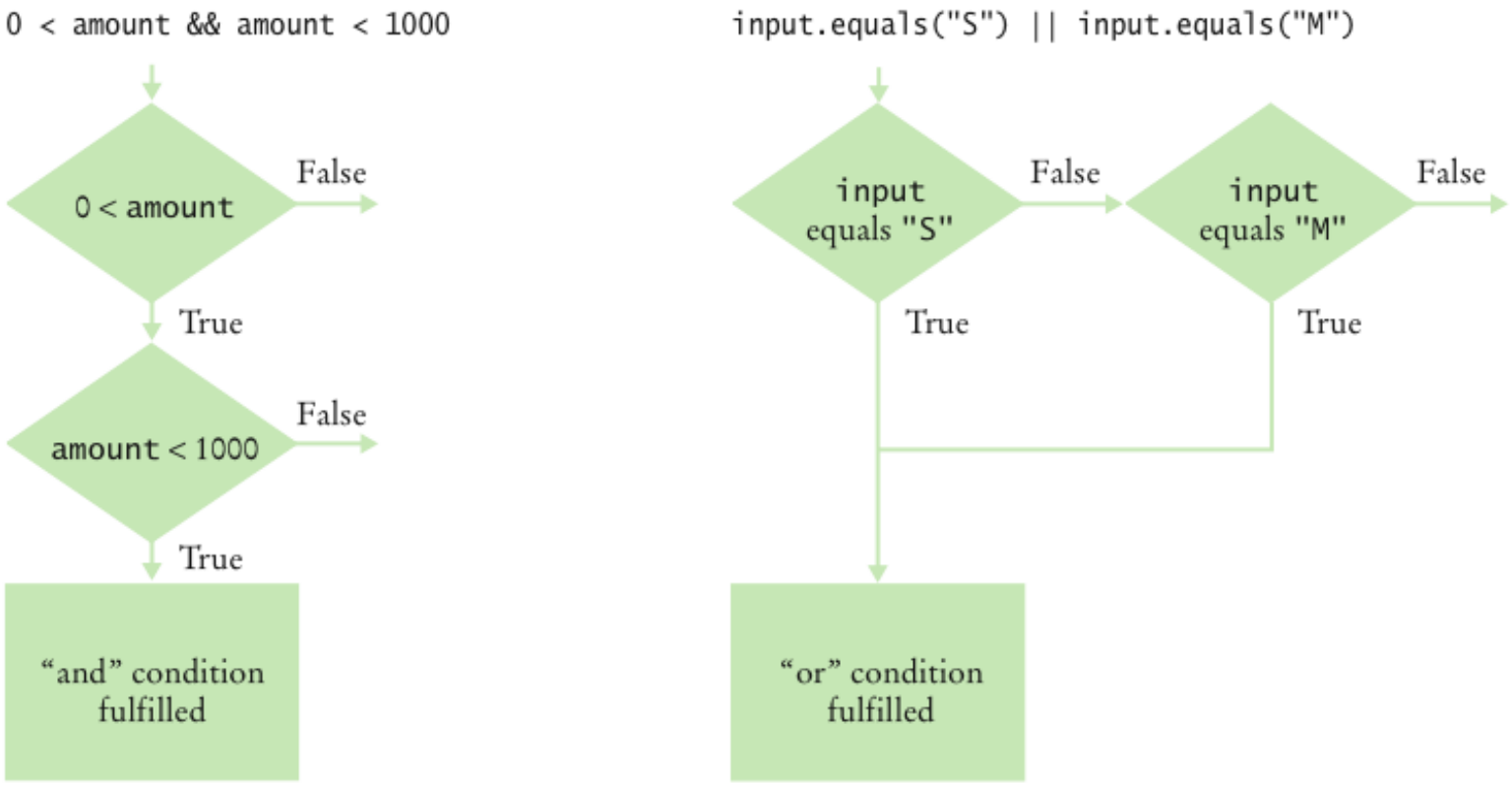


Figure 6 Flowcharts for && and || Combinations

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

Boolean Operators

Table 3 Boolean Operators

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

A	B	A || B
true	*Any*	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.

Self Check 5.8

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

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

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

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";
...
```
```

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";
```

ch05/quake/Earthquake.java

```
1  /**
2     A class that describes the effects of an earthquake.
3  */
4  public class Earthquake
5  {
6     private double richter;
7
8     /**
9        Constructs an Earthquake object.
10       @param magnitude the magnitude on the Richter scale
11    */
12    public Earthquake(double magnitude)
13    {
14       richter = magnitude;
15    }
16
```

Continued

ch05/quake/Earthquake.java (cont.)

```
17      /**
18         Gets a description of the effect of the earthquake.
19         @return the description of the effect
20      */
21      public String getDescription()
22      {
23         String r;
24         if (richter >= 8.0)
25            r = "Most structures fall";
26         else if (richter >= 7.0)
27            r = "Many buildings destroyed";
28         else if (richter >= 6.0)
29            r = "Many buildings considerably damaged, some collapse";
30         else if (richter >= 4.5)
31            r = "Damage to poorly constructed buildings";
32         else if (richter >= 3.5)
33            r = "Felt by many people, no destruction";
34         else if (richter >= 0)
35            r = "Generally not felt by people";
36         else
37            r = "Negative numbers are not valid";
38         return r;
39      }
40   }
```

ch05/quake/EarthquakeRunner.java

```
1  import java.util.Scanner;
2
3  /**
4     This program prints a description of an earthquake of a given magnitude.
5  */
6  public class EarthquakeRunner
7  {
8     public static void main(String[] args)
9     {
10       Scanner in = new Scanner(System.in);
11
12       System.out.print("Enter a magnitude on the Richter scale: ");
13       double magnitude = in.nextDouble();
14       Earthquake quake = new Earthquake(magnitude);
15       System.out.println(quake.getDescription());
16    }
17 }
```

Program Run:

```
Enter a magnitude on the Richter scale: 7.1
Many buildings destroyed
```

Multiple Alternatives: Nested Branches

- Branch inside another branch:

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

Tax Schedule

If your filing status is Single		If your filing status is Married	
Tax Bracket	**Percentage**	**Tax Bracket**	**Percentage**
$0 ... $32,000	10%	0 ... $64,000	10%
Amount over $32,000	25%	Amount over $64,000	25%

Nested Branches

- Compute taxes due, given filing status and income figure:
  1. *branch on the filing status*
  2. *for each filing status, branch on income level*
- The two-level decision process is reflected in two levels of `if` statements
- We say that the income test is *nested* inside the test for filing status

Nested Branches

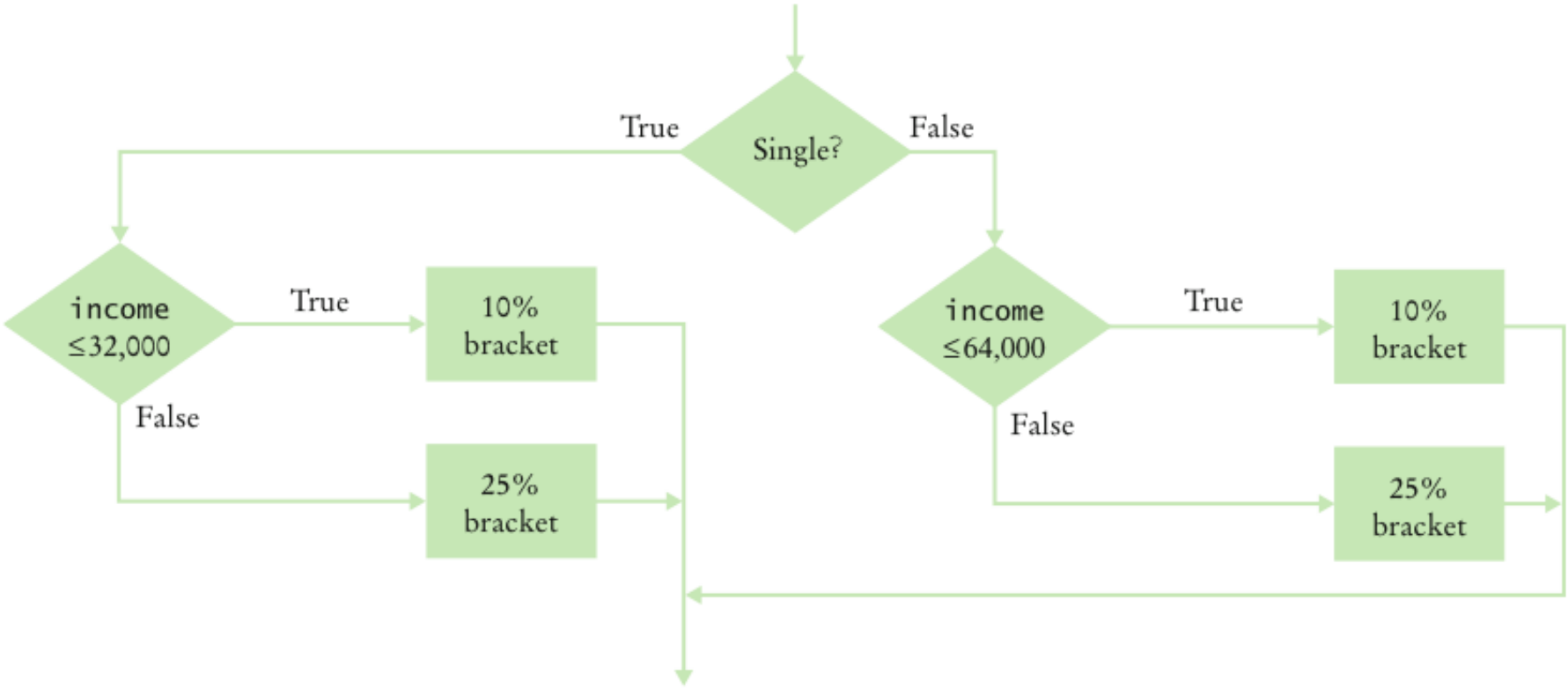


Figure 5 Income Tax Computation Using Simplified 2008 Schedule

ch05/tax/TaxReturn.java

```
1  /**
2     A tax return of a taxpayer in 2008.
3  */
4  public class TaxReturn
5  {
6     public static final int SINGLE = 1;
7     public static final int MARRIED = 2;
8
9     private static final double RATE1 = 0.10;
10    private static final double RATE2 = 0.25;
11    private static final double RATE1_SINGLE_LIMIT = 32000;
12    private static final double RATE1_MARRIED_LIMIT = 64000;
13
14    private double income;
15    private int status;
16
```

Continued

ch05/tax/TaxReturn.java (cont.)

```
17      /**
18         Constructs a TaxReturn object for a given income and
19         marital status.
20         @param anIncome the taxpayer income
21         @param aStatus either SINGLE or MARRIED
22      */
23      public TaxReturn(double anIncome, int aStatus)
24      {
25         income = anIncome;
26         status = aStatus;
27      }
28
29      public double getTax()
30      {
31         double tax1 = 0;
32         double tax2 = 0;
33
```

Continued

ch05/tax/TaxReturn.java (cont.)

```
34          if (status == SINGLE)
35          {
36             if (income <= RATE1_SINGLE_LIMIT)
37             {
38                tax1 = RATE1 * income;
39             }
40             else
41             {
42                tax1 = RATE1 * RATE1_SINGLE_LIMIT;
43                tax2 = RATE2 * (income - RATE1_SINGLE_LIMIT);
44             }
45          }
46          else
47          {
48             if (income <= RATE1_MARRIED_LIMIT)
49             {
50                tax1 = RATE1 * income;
51             }
52             else
53             {
54                tax1 = RATE1 * RATE1_MARRIED_LIMIT;
55                tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT);
56             }
57          }
58
59          return tax1 + tax2;
60       }
61    }
```

ch05/tax/TaxCalculator.java

```
 1  import java.util.Scanner;
 2
 3  /**
 4     This program calculates a simple tax return.
 5  */
 6  public class TaxCalculator
 7  {
 8     public static void main(String[] args)
 9     {
10        Scanner in = new Scanner(System.in);
11
12        System.out.print("Please enter your income: ");
13        double income = in.nextDouble();
14
15        System.out.print("Are you married? (Y/N) ");
16        String input = in.next();
17        int status;
18        if (input.equalsIgnoreCase("Y"))
19           status = TaxReturn.MARRIED;
20        else
21           status = TaxReturn.SINGLE;
22        TaxReturn aTaxReturn = new TaxReturn(income, status);
23
24        System.out.println("Tax: "
25              + aTaxReturn.getTax());
26     }
27  }
```

Continued

ch05/tax/TaxCalculator.java (cont.)

Program Run:

```
Please enter your income: 50000
Are you married? (Y/N) N
Tax: 11211.5
```

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";
...
```

Self Check 5.6

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.

Code Coverage

- **Black-box testing:** Test functionality without consideration of internal structure of implementation
- **White-box testing:** Take internal structure into account when designing tests
- **Test coverage:** Measure of how many parts of a program have been tested
- Make sure that each part of your program is exercised at least once by one test case
  E.g., make sure to execute each branch in at least one test case

Code Coverage

- Include boundary test cases: Legal values that lie at the boundary of the set of acceptable inputs
- Tip: Write first test cases before program is written completely → gives insight into what program should do

Self Check 5.9

How many test cases do you need to cover all branches of the `getDescription` method of the `Earthquake` class?

Answer: 7.

Self Check 5.10

Give a boundary test case for the `EarthquakeRunner` program. What output do you expect?

Answer: An input of 0 should yield an output of `"Generally not felt by people"`. (If the output is `"Negative numbers are not allowed"`, there is an error in the program.)