ICOM 4015: Advanced Programming

Lecture 5

Chapter Five: Decisions

Chapter Five: 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

The `if` Statement

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

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

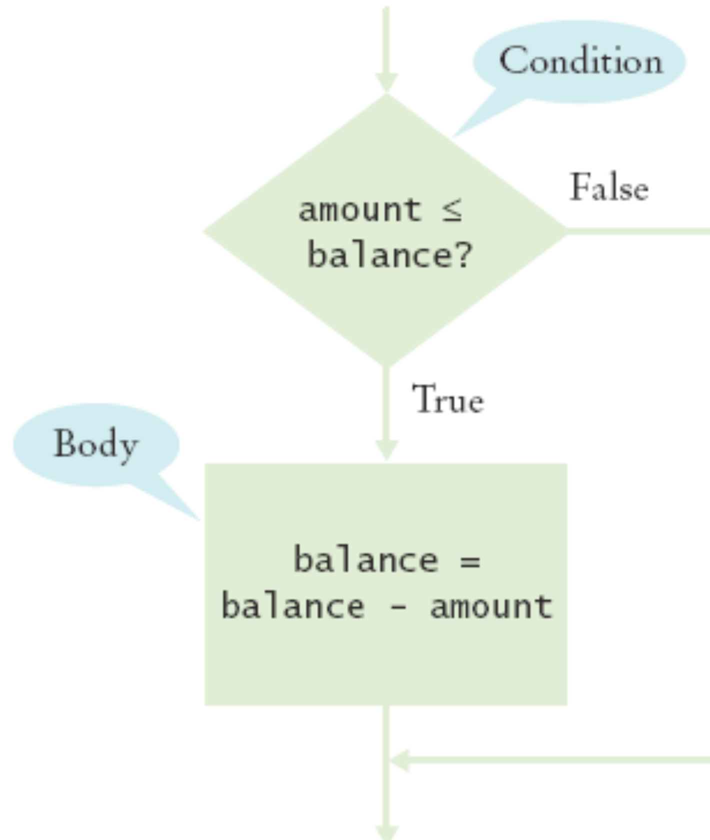

Figure 1
Flowchart for an `if` Statement

The `if/else` Statement

```
If (amount <= balance)
balance = balance – amount;
else
balance = balance – OVERDRAFT_PENALTY
```

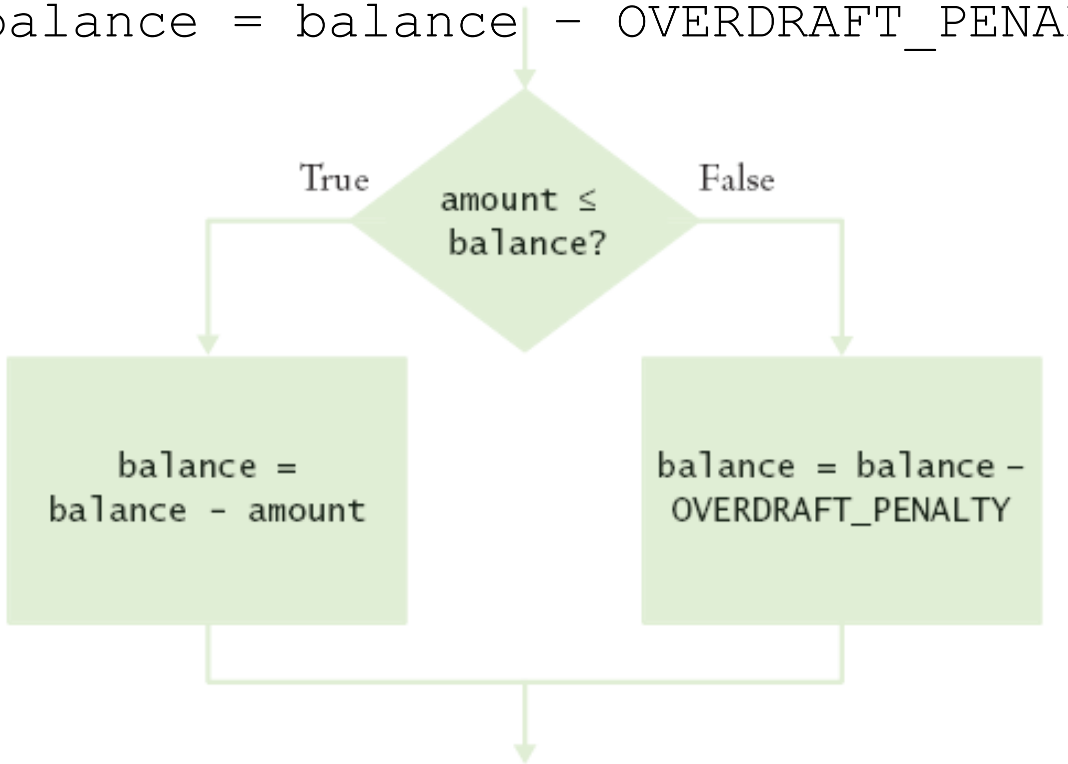

Figure 2
Flowchart for an `if/else` Statement

Statement Types

- Simple statement

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

- Compound statement

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

  Also

  `while`, `for`, etc. (loop statements – Chapter 6)

- Block statement

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

Syntax 5.1 The `if` Statement

```
if(condition)
   statement
if (condition)
   statement1
else
```

Example:

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

Purpose:

To execute a statement when a condition is true or false.

Syntax 5.2 Block Statement

```
{
   statement1
   statement2
   . . .
}
```

Example:

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

Purpose:

To group several statements together to form a single statement.

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.

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

- The `==` denotes equality testing

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

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*:

  $|x - y| \le \varepsilon$

  ```
  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

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

```
if (input == "Y") // WRONG!!!
```

- Use equals method:

```
if (input.equals("Y"))
```

- `==` tests identity, `equals` tests equal contents
- Case insensitive test ("Y" or "y")

```
if (input.equalsIgnoreCase("Y"))
```

- `s.compareTo(t) < 0` means:
  `s` comes before `t` in the dictionary

Continued

Comparing Strings (cont.)

- "car" comes before "cargo"
- All uppercase letters come before lowercase:
  "Hello" comes before "car"

Lexicographic Comparison

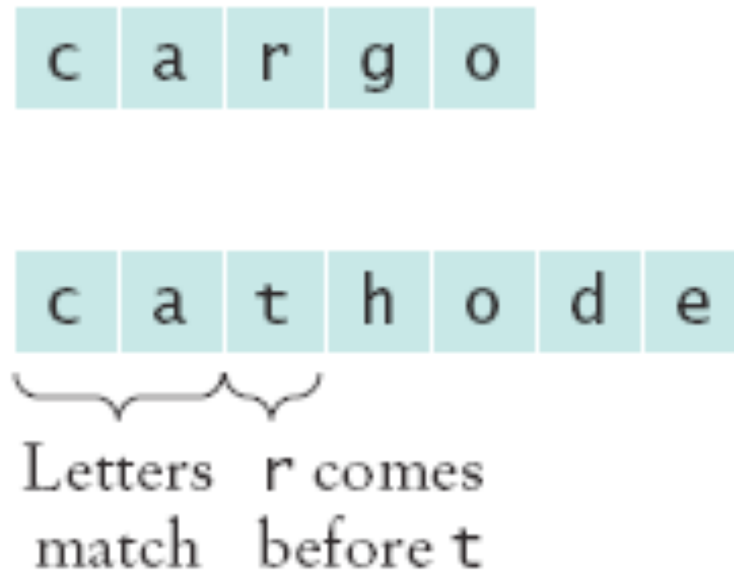

Figure 3
Lexicographic Comparison

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

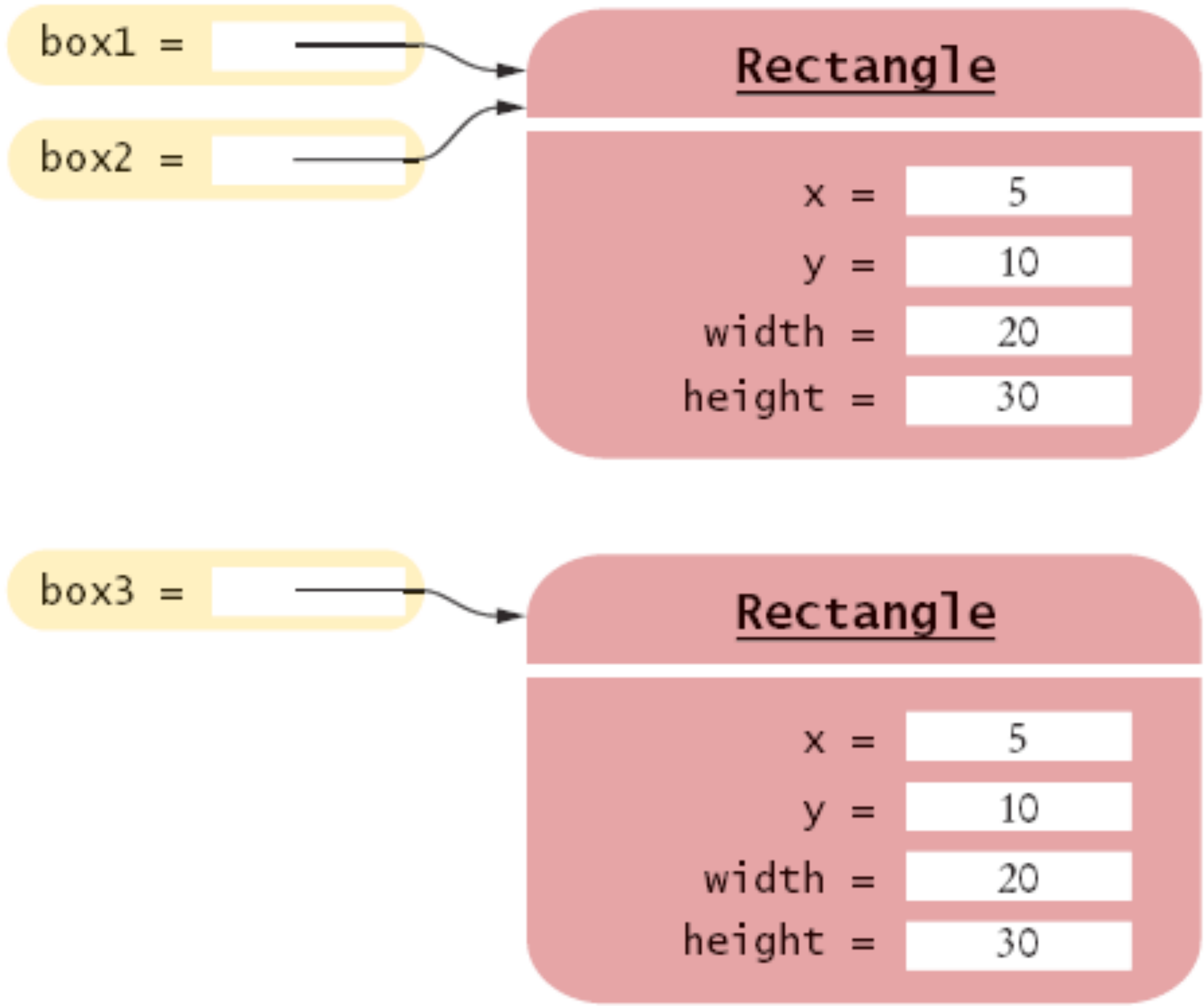

Figure 4 Comparing Object References

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 `""`

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 is thrown.

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)
```

Answer: Syntactically incorrect: e, g, h. Logically questionable: a, d, f.

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 (cont.)

- 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

```java
/**
   A class that describes the effects of an earthquake.
*/
public class Earthquake
{
   /**
      Constructs an Earthquake object.
      @param magnitude the magnitude on the Richter scale
   */
   public Earthquake(double magnitude)
   {
      richter = magnitude;
   }

   /**
      Gets a description of the effect of the earthquake.
      @return the description of the effect
   */
   public String getDescription()
   {
```

Continued

ch05/quake/Earthquake.java (cont.)

```
      String r;
      if (richter >= 8.0)
         r = "Most structures fall";
      else if (richter >= 7.0)
         r = "Many buildings destroyed";
      else if (richter >= 6.0)
         r = "Many buildings considerably damaged, some collapse";
      else if (richter >= 4.5)
         r = "Damage to poorly constructed buildings";
      else if (richter >= 3.5)
         r = "Felt by many people, no destruction";
      else if (richter >= 0)
         r = "Generally not felt by people";
      else
         r = "Negative numbers are not valid";
      return r;
   }

   private double richter;
}
```

ch05/quake/EarthquakeRunner.java

```java
import java.util.Scanner;

/**
   This program prints a description of an earthquake of a given magnitude.
*/
public class EarthquakeRunner
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      System.out.print("Enter a magnitude on the Richter scale: ");
      double magnitude = in.nextDouble();
      Earthquake quake = new Earthquake(magnitude);
      System.out.println(quake.getDescription());
   }
}
```

Output:

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 . . . $21,450	15%	0 . . . $35,800	15%
Amount over $21,450, up to $51,900	28%	Amount over $35,800, up to $86,500	28%
Amount over $51,900	31%	Amount over $86,500	31%

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

Continued

Nested Branches (cont.)

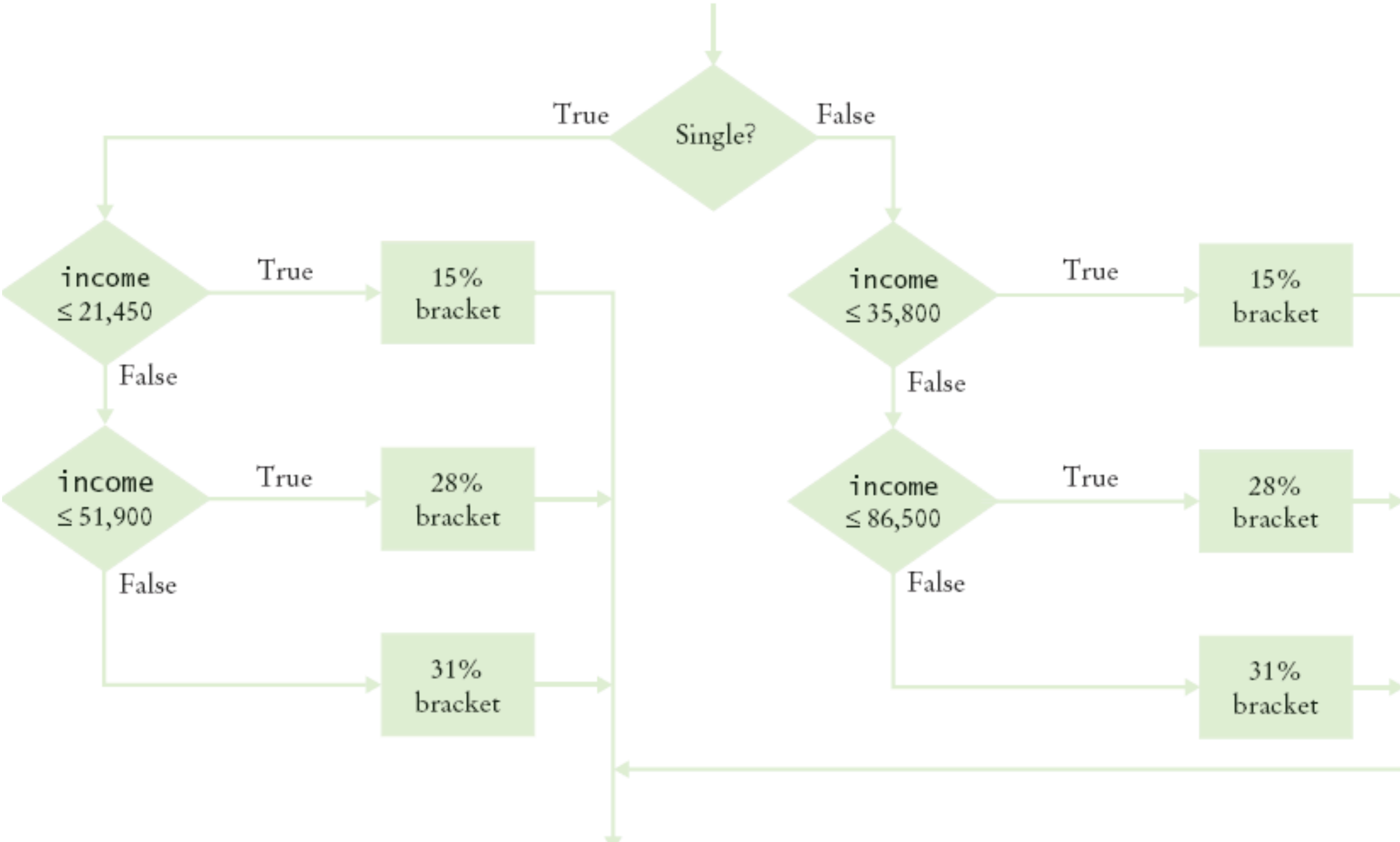

Figure 5 Income Tax Computation Using 1992 Schedule

ch05/tax/TaxReturn.java

```java
/**
   A tax return of a taxpayer in 1992.
*/
public class TaxReturn
{
   /**
      Constructs a TaxReturn object for a given income and
      marital status.
      @param anIncome the taxpayer income
      @param aStatus either SINGLE or MARRIED
   */
   public TaxReturn(double anIncome, int aStatus)
   {
      income = anIncome;
      status = aStatus;
   }

   public double getTax()
   {
      double tax = 0;

      if (status == SINGLE)
      {
```

Continued

ch05/tax/TaxReturn.java (cont.)

```java
         if (income <= SINGLE_BRACKET1)
            tax = RATE1 * income;
         else if (income <= SINGLE_BRACKET2)
            tax = RATE1 * SINGLE_BRACKET1
                  + RATE2 * (income - SINGLE_BRACKET1);
         else
            tax = RATE1 * SINGLE_BRACKET1
                  + RATE2 * (SINGLE_BRACKET2 - SINGLE_BRACKET1)
                  + RATE3 * (income - SINGLE_BRACKET2);
      }
      else
      {
         if (income <= MARRIED_BRACKET1)
            tax = RATE1 * income;
         else if (income <= MARRIED_BRACKET2)
            tax = RATE1 * MARRIED_BRACKET1
                  + RATE2 * (income - MARRIED_BRACKET1);
         else
            tax = RATE1 * MARRIED_BRACKET1
                  + RATE2 * (MARRIED_BRACKET2 - MARRIED_BRACKET1)
                  + RATE3 * (income - MARRIED_BRACKET2);
      }

```

Continued

ch05/tax/TaxReturn.java (cont.)

```
        return tax;
    }

    public static final int SINGLE = 1;
    public static final int MARRIED = 2;

    private static final double RATE1 = 0.15;
    private static final double RATE2 = 0.28;
    private static final double RATE3 = 0.31;

    private static final double SINGLE_BRACKET1 = 21450;
    private static final double SINGLE_BRACKET2 = 51900;

    private static final double MARRIED_BRACKET1 = 35800;
    private static final double MARRIED_BRACKET2 = 86500;

    private double income;
    private int status;
}
```

ch05/tax/TaxCalculator.java

```java
import java.util.Scanner;

/**
   This program calculates a simple tax return.
*/
public class TaxCalculator
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      System.out.print("Please enter your income: ");
      double income = in.nextDouble();

      System.out.print("Are you married? (Y/N) ");
      String input = in.next();
      int status;
      if (input.equalsIgnoreCase("Y"))
         status = TaxReturn.MARRIED;
      else
         status = TaxReturn.SINGLE;

```

Continued

ch05/tax/TaxCalculator.java (cont.)

```java
      TaxReturn aTaxReturn = new TaxReturn(income, status);

      System.out.println("Tax: "
            + aTaxReturn.getTax());
   }
}
```

Output:

```
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 $51,800. Should you try to get a $200 raise? Absolutely–you get to keep 72% of the first $100 and 69% of the next $100.

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

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
```

Continued

Using Boolean Expressions: Predicate Method (cont.)

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

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

Using Boolean Expressions: The Boolean Operators

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

&& and || Operators

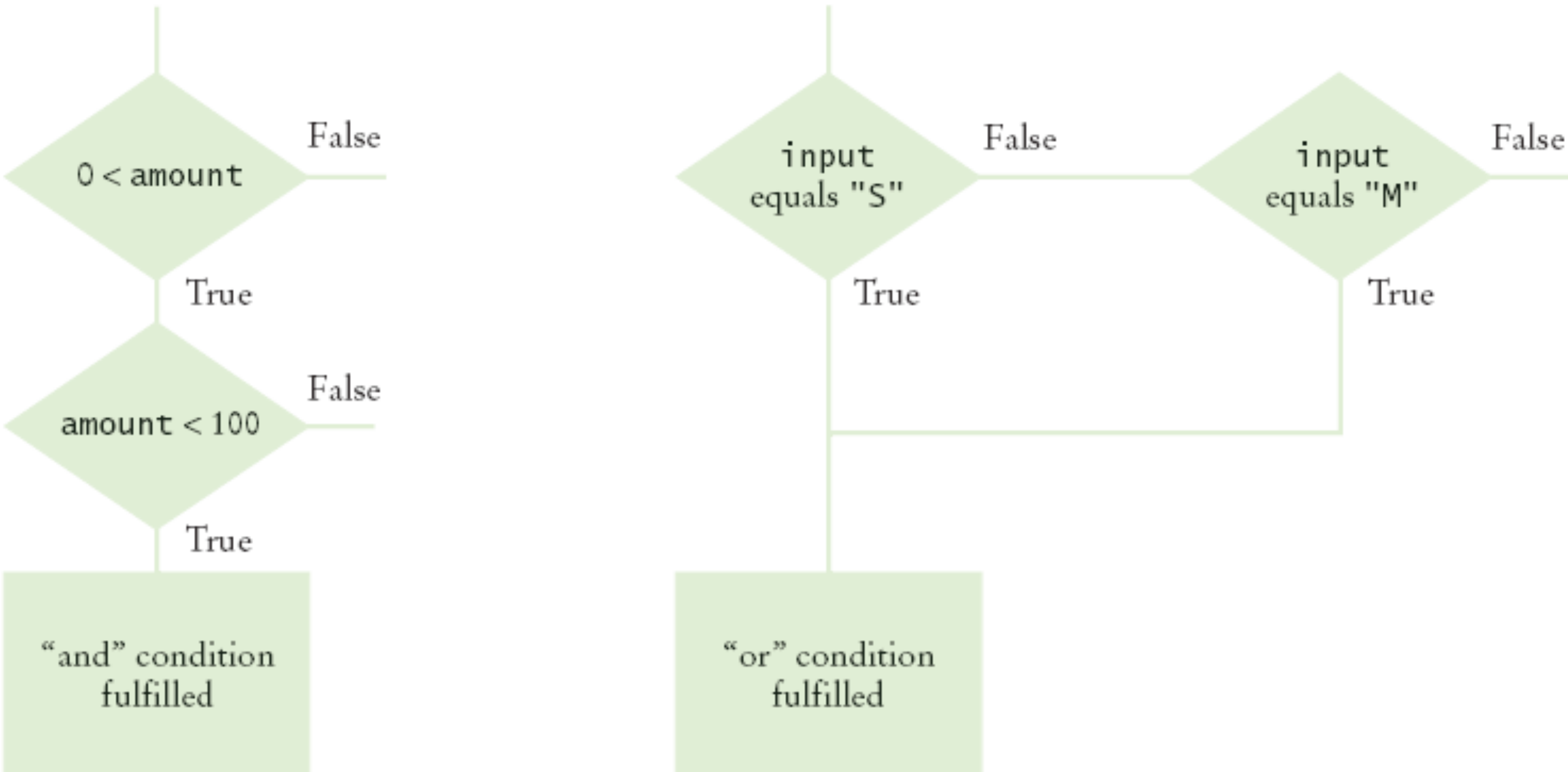

Figure 6 Flowcharts for && and || Combinations

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

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) . . .
  ```

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)) . . .`

Test 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

Continued

Test Coverage (cont.)

- 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.)