Decisions

Advanced Programming

ICOM 4015

Lecture 5

Reading: Java Concepts Chapter 6
Lecture 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
The if Statement

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

```java
if (amount <= balance)
    balance = balance - amount;
```
The if Statement

Figure 1: Flowchart for an if statement

- amount ≤ balance?
  - False
  - balance = balance - amount
  - True
- amount ≤ balance?
The if/else Statement

if (amount <= balance)
    balance = balance - amount;
else
    balance = balance - OVERDRAFT_PENALTY;
The if/else Statement

Figure 2: Flowchart for an if/else statement

- True: amount ≤ balance?
  - balance = balance - amount
- False: balance = balance - OVERDRAFT_PENALTY
Statement Types

• Simple statement

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

• Compound statement

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

Also `while`, `for`, etc. (loop statements–Chapter 7)

Continued...
Statement Types

• Block statement

```java
{  
double newBalance = balance - amount;
balance = newBalance;
}
```
Syntax 6.1: The *if* Statement

```java
if (condition)
    statement

if (condition)
    statement1
else
    statement2
```

**Example:**
```java
if (amount <= balance)
    balance = balance - amount;

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

**Purpose:**
To execute a statement when a condition is true or false
Syntax 6.2: Block Statement

Example:
```
{ 
    statement_1
    statement_2
    ...
}
```

Purpose:
To group several statements together to form a single statement
Self-Check

1. Why did we use the condition \( amount \leq balance \) and not \( amount < balance \) in the example for the if/else statement?

2. What is logically wrong with the statement and how do you fix it?

```java
if (amount <= balance)
    newBalance = balance - amount; balance = newBalance;
```
1. If the withdrawal amount equals the balance, the result should be a zero balance and no penalty

2. 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

<table>
<thead>
<tr>
<th>Java</th>
<th>Math Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>≥</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>≤</td>
<td>Less than or equal</td>
</tr>
<tr>
<td>==</td>
<td>=</td>
<td>Equal</td>
</tr>
<tr>
<td>!=</td>
<td>≠</td>
<td>Not equal</td>
</tr>
</tbody>
</table>

- 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:

```java
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: $|x - y| \leq \varepsilon$

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

$\varepsilon$ is a small number such as $10^{-14}$
Comparing Strings

• Don't use `==` for strings!

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

• Use equals method:

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

== tests identity, equals tests equal contents

• Case insensitive test ("Y" or "y")

```java
if (input.equalsIgnoreCase("Y"))
```
Comparing Strings

- `s.compareTo(t) < 0` means `s` comes before `t` in the dictionary
- "car" comes before "cargo"
- All uppercase letters come before lowercase: "Hello" comes before "car"
Lexicographic Comparison

Figure 3:
Lexicographic Comparison
Fall 2006 Slides adapted from Java Concepts companion slides
Comparing Objects

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

```java
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

box1 = Rectangle
  x = 5
  y = 10
  width = 20
  height = 30

box2 = Rectangle
  x = 5
  y = 10
  width = 20
  height = 30

box3 = Rectangle
  x = 5
  y = 10
  width = 20
  height = 30
Testing for **null**

- **null** reference refers to no object

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

- Can be used in tests:

```java
if (middleInitial == null) {
    System.out.println(firstName + " " + lastName);
} else {
    System.out.println(firstName + " " + middleInitial + ". " + lastName);
}
```
Testing for null

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

Fall 2006

Slides adapted from Java Concepts companion slides
Self Check

1. What is the value of `s.length()` if `s` is
   1. the empty string ""?
   2. the string " " containing a space?
   3. `null`?

Self-Check

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

```java
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)
```
Answers

1. (a) 0; (b) 1; (c) an exception is thrown
2. Syntactically incorrect: e, g, h. Logically questionable: a, d, f
Multiple Alternatives: Sequences of Comparisons

- The first matching condition is executed
- Order matters

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

```java
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`

```java
if (richter >= 8.0)
    r = "Most structures fall";
if (richter >= 7.0) // omitted else--ERROR
    r = "Many buildings destroyed"
```
A class that describes the effects of an earthquake.

```java
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() {
        // Implementation
    }
}
```

Continued...
public String getDescription() {
    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;
}
File Earthquake.java

38:
39: private double richter;
40: }
File EarthquakeTester.java

```java
01: import java.util.Scanner;
02:
03: /**
04: * A class to test the Earthquake class.
05: */
06: public class EarthquakeTester
07: {
08:     public static void main(String[] args)
09:     {
10:         Scanner in = new Scanner(System.in);
11:         System.out.print("Enter a magnitude on the Richter scale: ");
12:         double magnitude = in.nextDouble();
13:         Earthquake quake = new Earthquake(magnitude);
14:         System.out.println(quake.getDescription());
15:     }
16: }
17: }
```
Multiple Alternatives: Nested Branches

- Branch inside another branch

```java
if (condition1)
{
    if (condition1a)
        statement1a;
    else
        statement1b;
}
else
    statement2;
```
## Tax Schedule

<table>
<thead>
<tr>
<th>If your filing status is single</th>
<th>Tax Bracket</th>
<th>Percentage</th>
<th>If your filing status is married</th>
<th>Tax Bracket</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 \ldots $21,450</td>
<td>$0 \ldots $21,450</td>
<td>15%</td>
<td>$0 \ldots $35,800</td>
<td>15%</td>
<td></td>
</tr>
<tr>
<td>Amount over $21,451, up to $51,900</td>
<td>28%</td>
<td>$0 \ldots $35,800, up to $86,500</td>
<td>28%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amount over $51,900</td>
<td>31%</td>
<td>Amount over $86,500</td>
<td>31%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
Figure 5: Income Tax Computation Using 1992 Schedule
File TaxReturn.java

```java
public class TaxReturn {
    public TaxReturn(double anIncome, int aStatus) {
        income = anIncome;
        status = aStatus;
    }
}
```

Continued…
File TaxReturn.java

18:    public double getTax()
19:    {
20:        double tax = 0;
21:        
22:        if (status == SINGLE)
23:            {
24:                if (income <= SINGLE_BRACKET1)
25:                    tax = RATE1 * income;
26:                else if (income <= SINGLE_BRACKET2)
27:                    tax = RATE1 * SINGLE_BRACKET1
28:                        + RATE2 * (income - SINGLE_BRACKET1);
29:                else
30:                    tax = RATE1 * SINGLE_BRACKET1
31:                        + RATE2 * (SINGLE_BRACKET2 – SINGLE_BRACKET1)
32:                        + RATE3 * (income - SINGLE_BRACKET2);
33:            }
34:        
35:        return tax;
36:    }
37:    
38:    private static final double RATE1 = 0.15;
39:    private static final double RATE2 = 0.25;
40:    private static final double RATE3 = 0.30;
41:    private static final double SINGLE_BRACKET1 = 10000;
42:    private static final double SINGLE_BRACKET2 = 25000;
43:    
44:    public int getStatus()
45:    {
46:        return status;
47:    }
48:    
49:    public double getIncome()
50:    {
51:        return income;
52:    }
53:    
54:    public void setStatus(int status)
55:    {
56:        this.status = status;
57:    }
58:    
59:    public void setIncome(double income)
60:    {
61:        this.income = income;
62:    }
63:    
64:    private int status;
65:    private double income;
66:
67:    static{
68:        status = DEFAULT_STATUS;
69:        income = DEFAULT_INCOME;
70:    }
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);
}

return tax;

public static final int SINGLE = 1;
public static final int MARRIED = 2;
```java
class TaxReturn {
    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;
    }
```
import java.util.Scanner;

/**
 * A class to test the TaxReturn class.
 */
public class TaxReturnTester {
    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("Please enter S (single) or M (married): ");
        String input = in.next();
        int status = 0;
    }
}
if (input.equalsIgnoreCase("S"))
    status = TaxReturn.SINGLE;
else if (input.equalsIgnoreCase("M"))
    status = TaxReturn.MARRIED;
else
{
    System.out.println("Bad input.");
    return;
}
TaxReturn aTaxReturn = new TaxReturn(income, status);
System.out.println("The tax is "+ aTaxReturn.getTax());
File TaxReturnTester.java

Output

Please enter your income: 50000
Please enter S (single) or M (married): S
The tax is 11211.5
Self Check

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

2. 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?
Answers

1. Yes, if you also reverse the comparisons:

```java
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";
...
2. 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 \( \text{amount} < 1000 \) is true or false. 
- boolean type: one of these 2 truth values
Using Boolean Expressions: The `boolean` Type
Using Boolean Expressions: Predicate Method

• A predicate method returns a boolean value

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

• Use in conditions

```java
if (harrysChecking.isOverdrawn()) . . .
```
Using Boolean Expressions: Predicate Method

• Useful predicate methods in `Character` class:
  - `isDigit`
  - `isLetter`
  - `isUpperCase`
  - `isLowerCase`

• `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

```java
if (0 < amount && amount < 1000) . . .
```

```java
if (input.equals("S") || input.equals("M")) . . .
```
Operators

Figure 6:
Flowcharts for && and || Combinations
# Truth Tables

<table>
<thead>
<tr>
<th>A</th>
<th>! A</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A &amp; &amp; B</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>Any</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A</th>
<th></th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Any</td>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using Boolean Variables

- Set to truth value:
  
  ```java
  married = input.equals("M");
  ```

- Use in conditions:
  
  ```java
  if (married) . . . else . . .
  if (!married) . . .
  ```
Using Boolean Variables

- Also called *flag*
- It is considered gauche to write a test such as

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

  **Just use the simpler test**

  ```java
  if (married) ... 
  ```
Self Check

1. When does the statement

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

print `false`?

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

```java
if (Character.isDigit(ch) == false) . . .
```
Answers

1. When $x$ is zero

2. `if (!Character.isDigit(ch))`