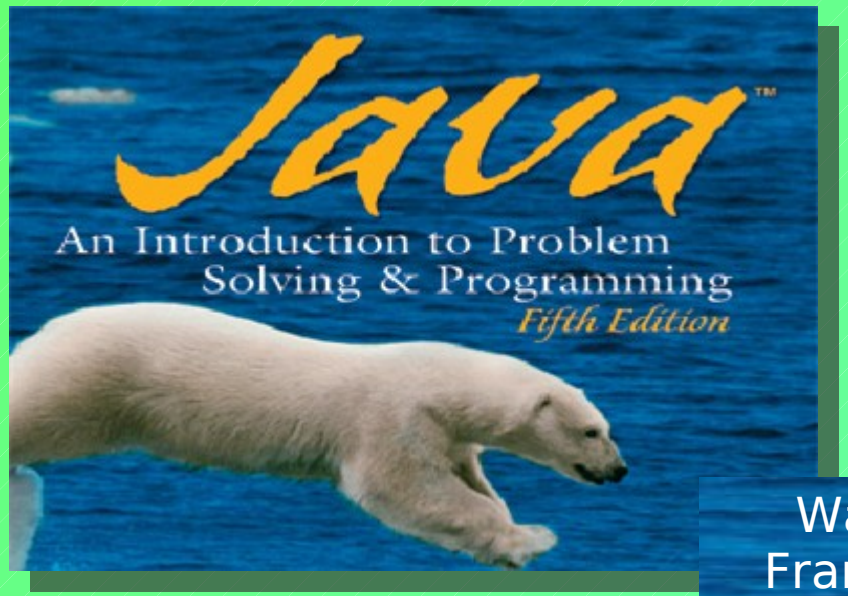


Walter Savitch
Frank M. Carrano

# Flow of Control

Chapter 3

*JAVA: An Introduction to Problem Solving & Programming, 5*th Ed. By Walter Savitch and Frank Carrano.
ISBN 0136130887 

# Outline

- The **`if-else`** statement
- The Type **`boolean`**
- The **`switch`** statement

# Flow of Control

- *Flow of control* is the order in which a program performs actions.
  - Up to this point, the order has been sequential.
- A *branching statement* chooses between two or more possible actions.

# The `if-else` Statement

- A branching statement that chooses between two possible actions.
- Syntax

```
if (Boolean_Expression)
{
  // do this if Boolean_Expression is true
}
else
{
 // do this if Boolean_Expression is false
}
```

# The `if-else` Statement

- Example

```
String message = “”;
balance = keyboard.nextDouble();
if (balance >= 0) {
  balance += (INTEREST_RATE * balance) / 12;
  message = “Interest has been added ”;
} else {
  balance -= OVERDRAWN_PENALTY;
  message = “Penalties have been deducted ”);
}
System.out.println(message + balance);
```

# Omitting the `else` Part

- Sometimes you don't need the `else` part:

```
int count = 0;
if (word.length() > 0)
{
   count++;
}
System.out.println("count: " + count);
```

# Introduction to Boolean Expressions

- The value of a *boolean expression* is either **`true`** or **`false`**.
- Examples

```
time < limit
balance <= 0
```

# Java Comparison Operators

- Figure 3.4 Java Comparison Operators

| Math Notation | Name | Java Notation | Java Examples |
|---|---|---|---|
| = | Equal to | == | `balance == 0`<br>`answer == 'y'` |
| ≠ | Not equal to | != | `income != tax`<br>`answer != 'y'` |
| > | Greater than | > | `expenses > income` |
| ≥ | Greater than or equal to | >= | `points >= 60` |
| < | Less than | < | `pressure < max` |
| ≤ | Less than or equal to | <= | `expenses <= income` |

# Java Logical Operators

| Name | Java Notation | Java Examples |
|---|---|---|
| Logical *and* | && | `(sum > min) && (sum < max)` |
| Logical *or* | \|\| | `(answer == 'y') \|\| (answer == 'Y')` |
| Logical *not* | ! | `!(number < 0)` |

# Compound Boolean Expressions

- Boolean expressions can be combined using the "and" **(&&)** operator.
- Example

```
if ((score > 0) && (score <= 100))
  ...
```

- Not allowed

```
if (0 < score <= 100) // NO!!
  ...
```

# Compound Boolean Expressions

- Boolean expressions can be combined using the "or" `(||)` operator.
- Example

```
if ((quantity > 5) || (cost < 10))
...
```

# Compound Boolean Expressions

- When using || the larger expression is true
  - When either of the smaller expressions is true
  - When both of the smaller expressions are true.
- The Java version of "or" is the *inclusive or* which allows either or both to be true.
- The *exclusive or* allows one or the other, but not both to be true.

# Negating a Boolean Expression

- A boolean expression can be negated using the "not" (`!`) operator.
- Examples

```
(!a || b)
(a && !b)
(a || b) && !(a && b) // exclusive or
```

# Exercise

- What is the output of the following code?

```
boolean x=true, y=false, z=true;
if ((x && y) || (x && z)) {
   System.out.println("First condition is " +
                      "true");
}
if ((y && z) || !x)) {
   System.out.println("Second condition is " +
                      "true");
}
```

# Use == with ints and chars

- == is appropriate for determining if two integers or characters have the same value.

```
if (num == 3)
if (firstChar == 'n')
```

# Don't use == with doubles

- == is **not** appropriate for determining if two floating point values are equal.
- Try this in the interactions pane:

```
double d1 = 63.27, d2 = 1.0;
d1 + d2
```

- Because of the way floating point values are stored, rounding errors can occur.

# Don't use == with doubles

- Use `<` and some appropriate tolerance instead.

  ```
  if (Math.abs(b - c) < epsilon)
  ```

  where `b`, `c`, and `epsilon` are floating point types
  - Translation: if the difference between b and c is very small, consider them equal.

# Don't use `==` with objects

- `==` is not appropriate for determining if two objects have the same value.
  - `if (s1 == s2),` where `s1` and `s2` refer to strings, determines only if s1 and s2 refer the a common memory location.
  - If `s1` and `s2` refer to strings with identical sequences of characters, but stored in different memory locations, `(s1 == s2)` is false.

# Don't use == with objects

- Try it out in the interactions pane:

```
String s1 = “hello”;
String s2 = “hello”;
System.out.println(s1 == s2);

String s3 = “bye”;
String s4 = s3;
System.out.println(s3 == s4);
```

# `equals` and `equalsIgnoreCase`

- To test the equality of objects of class String, use method **`equals`**.

  **`s1.equals(s2)`**

  or

  **`s2.equals(s1)`**

- To test for equality ignoring case, use method **`equalsIgnoreCase`**.

  **`"Hello".equalsIgnoreCase("hello")`**

# Summary: equals vs ==

| | "==" | equals | |
|---|---|---|---|
| **int** | yes | no | |
| **char** | yes | no | |
| **double** | no | no | if (Math.abs(d1 – d2) < epsilon)<br>// epsilon is some small number<br>// 0.00000000001 for example |
| **boolean** | no | no | if (done)<br>if (!done) |
| **object** | no | yes | if (s1 == s2) // compares memory locations<br>if (s1.equals(s2)) // compares contents |

# Lexicographic Order

- Lexicographic order is similar to alphabetical order, but is it based on the order of the characters in the Unicode character set.
  - All the digits come before all the letters.
  - All the uppercase letters come before all the lower case letters.

# Method `compareTo`

- Method `compareTo:`
  - `if (s1.compareTo(s2) < 0)`
    - `s1 < s2`
  - `if (s1.compareTo(s2) == 0)`
    - `s1` and `s2` are equal
  - `if (s1.compareTo(s2) > 0)`
    - `s1 > s2`

# Method `compareTo`

- Try it out in the interactions pane:

```
"alex".compareTo("zack")
"Book".compareTo("book")
"abcdef".compareTo("abbdef")
"long".compareTo("longest")
```

# Nested `if-else` Statements

- An `if-else` statement can contain any sort of statement within it.
- In particular, it can contain another `if-else` statement.
  - An `if-else` may be nested within the "if" part.
  - An `if-else` may be nested within the "else" part.
  - An `if-else` may be nested within both parts.

# Nested Statements

- Syntax

```
if (Boolean_Expression_1) {
    if (Boolean_Expression_2) {
        Statement(s)
    } else {
        Statement(s)
    }
} else {
    if (Boolean_Expression_3) {
        Statement(s)
    } else {
        Statement(s)
    }
}
```

# Nested Statements

- Each `else` is paired with the nearest unmatched `if`.
- **If used properly**, indentation communicates which `if` goes with which `else`.
- Always use curly braces even though they are not required when an `if` or `else` block contains only one statement
- Indent your code with your IDE to see how the `if`s and `else`s will be paired up

# Multibranch `if-else` Statements

- Syntax

```
if (Boolean_Expression_1) {
    Statement(s)
} else if (Boolean_Expression_2) {
    Statement(s)
} else if (Boolean_Expression_3) {
    Statement(s)
} else if …
} else {
    Default_Statement(s)
}
```

# Multibranch `if-else` Statements

- Download and open **`Grader.java`**
- Boolean expressions are evaluated in order
- The statements in the first branch that evaluate to **`true`** are executed
  - no further branches are evaluated

# `if-else` Exercise

Rewrite the `if-else` in `Grader.java` so that the first `if` sets grade = 'F' and the last `else` sets grade = 'A'

# `if-else` Exercise

```
if (score < 60)
    grade = 'F';
else if (score < 70)
    grade = 'D';
else if (score < 80)
    grade = 'C';
else if (score < 90)
    grade = 'B';
else
    grade = 'A';
```

# The Conditional Operator

```
if (n1 > n2)
   max = n1;
else
   max = n2;
```

can be written as

```
max = (n1 > n2) ? n1 : n2;
```

- The `?` and `:` together are called the *conditional operator* or *ternary operator.*

# The Conditional Operator

- The conditional operator is useful with print and println statements.

```
System.out.print("You worked " +
  hours +
  ((hours > 1) ? " hours" : " hour"));
```

# The `exit` Method

- Sometimes a situation arises that makes continuing the program pointless.
- A program can be terminated with

```
System.exit(0);
```

# The `exit` Method

- Example

```
if (numberOfWinners == 0) {
    System.out.println("Error: Dividing by zero.");
    System.exit(0);
} else {
    oneShare = payoff / numberOfWinners;
    System.out.println("Each winner will receive $"
                       + oneShare);
}
```

# Boolean Expressions and Variables

- Variables, constants, and expressions of type **`boolean`** all evaluate to either **`true`** or **`false`**.
- A boolean variable can be given the value of a boolean expression by using an assignment operator.

```
boolean isPositive = (number > 0);
...
if (isPositive) ...
```

# Naming Boolean Variables

- Choose names that sound good in an `if` statement
- Examples:
  - if (`isNoun`)
  - if (`isPositive`)
  - if (`!done`)

# Input and Output of Boolean Values

- Example – try it out:

```
boolean boolVar = false;
Scanner keyboard = new Scanner(System.in);

System.out.println(boolVar);
System.out.println("Enter a boolean value:");
boolVar = keyboard.nextBoolean();
System.out.println("You entered " + boolVar);
```

# Short-circuit Evaluation

- Sometimes only part of a boolean expression needs to be evaluated to determine the value of the entire expression.
  - If the first operand associated with an `||` is `true`, the expression is `true`.
  - If the first operand associated with an `&&` is `false`, the expression is `false`.
- This is called *short-circuit* or *lazy* evaluation.

# Short-circuit Evaluation

- Short-circuit evaluation is not only efficient, sometimes it is essential!
- A run-time error can result, for example, from an attempt to divide by zero.

```
if ((number != 0) && (sum/number > 5))
```

- A run-time error can also result from an attempt to call a method on a null object.

```
if ((name != null) && (name.length() > 0))
```

# The `switch` Statement

- The `switch` statement is a multiway branch that makes a decision based on an *integral* (integer or character) expression.
- The action associated with a matching case label is executed.
- If no match is found, the case labeled `default` is executed.

# The `switch` Statement

- Syntax

```
switch (Controlling_Expression)
{
   case Case_Label:
        Statement(s);
        break;
   case Case_Label:
   …
   default:
   …
}
```

# The `switch` Statement

- Download **`MultipleBirths.java`**
- The action for each case typically ends with the word **`break`**.
- The optional **`break`** statement prevents the consideration of other cases.
- The controlling expression can be anything that evaluates to an integral type (int or char).

# `switch` Exercise

- Write a program that determines if a word entered by the user starts with a vowel ('a', 'A', 'e', 'E', 'i', 'I', 'o', 'O', 'u', 'U'). Use a switch statement.

# `switch` Exercise

```
import java.util.*;
public class VowelStuff {

    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        String word;
        char firstChar;
        boolean isVowel;

        System.out.println("Enter a word:");
        word = keyboard.next();
        firstChar = word.charAt(0);

        switch (firstChar) {
            case 'a':
            case 'A':
            case 'e':
            case 'E':
            case 'i':
            case 'I':
            case 'o':
            case 'O':
            case 'u':
            case 'U':
                isVowel = true;
                break;

            default:
                isVowel = false;
        }

        if (isVowel) {
            System.out.println(word + " starts with a vowel");
        } else {
            System.out.println(word + " starts with a consonant");
        }
    }
}
```

# Enumerations

- Consider a need to restrict contents of a variable to certain values
- An enumeration lists the values a variable can have
- Example:

```
enum Language {ENGLISH, GERMAN, FRENCH, SPANISH}
Language lang;  // initially null
lang = Language.ENGLISH;

System.out.println(lang);  // prints ENGLISH
```

# Enumerations

- Now possible to use in a **`switch`** statement

```
switch (lang)
{
      case GERMAN:
           System.out.println("Wie geht's?");
           break;
      case FRENCH:
           System.out.println("Comment vas-tu?");
           break;
      case SPANISH:
           System.out.println("¿Cómo estás?");
           break;
      case ENGLISH:
      default:
           System.out.println("How are you?");
}
```