Selections

CSE 114: Introduction to Object-Oriented Programming
Paul Fodor
Stony Brook University

http://www.cs.stonybrook.edu/~cse114
Contents

- Motivation for selection statements
- The Comparison Operators and the boolean Type
- One-way if Statements
- Two-way if Statement
  - Multiple Alternative if Statements
  - if … else matching
  - Common Errors
- Complex conditions with Logical Operators
  - Example: Determining Leap Year
  - The unconditional & and | Operators
- switch Statements
- The Conditional Operator
- Operator Precedence and Associativity
Computing the Area of a Circle:

```java
import java.util.Scanner;
public class ComputeArea {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a radius: ");
        double radius = input.nextDouble();
        double area = radius * radius * 3.14159;
        // Display results
        System.out.println("The area for the circle" + " of radius " + radius + " is " + area);
    }
}
```

What if the user enters a negative value (i.e., an invalid value)?

The area for a circle with a negative radius does not make sense.
Motivation

If the user entered a negative value for radius in ComputeArea.java, then you don't want the program to compute the area, but to inform the user that their input was incorrect.
Computing the Area of a Circle:

```java
import java.util.Scanner;

public class ComputeAreaNew {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a radius: ");
        double radius = input.nextDouble();
        if(radius >= 0){
            double area = radius * radius * 3.14159;
            System.out.println("The area for the circle of radius "+radius+" is "+area);
        } else {
            System.out.println("The radius is negative. The area cannot be computed.");
        }
    }
}
```
The Comparison Operators and boolean Type

- Often in a program you need to compare values: e.g., if x is greater than y then ...
- Java provides six comparison operators (also called relational operators) to compare two values: <, <=, >, >=, == and !=

  - The result of the comparison is a Boolean value: true or false. For example,

    boolean b = (1 > 2);

    b is false after the statement.
## Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
</tbody>
</table>

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
One-way if Statements

if (boolean-expression) {
    statement(s);
}

if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area for the circle of radius " + radius + " is " + area);
}

(A) Boolean Expression
true
Statement(s)
false

(B) (radius >= 0)
true
false
area = radius * radius * PI;
System.out.println("The area for the circle of " + "radius " + radius + " is " + area);

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
One-way if Statements

Condition containment is necessary!

```java
if i > 0 {
    System.out.println("i is positive");
}
```
(a) Wrong

```java
if (i > 0) {
    System.out.println("i is positive");
}
```
(b) Correct

Block containment is not necessary for a **single** statement!

```java
if (i > 0) {
    System.out.println("i is positive");
}
```
(a)

Equivalent

```java
if (i > 0) {
    System.out.println("i is positive");
}
```
(b)
Two-way if Statement

if (boolean-expression) {
    statement(s) - for-the-true-case;
} else {
    statement(s) - for-the-false-case;
}
Two-way if Example

```java
if (radius >= 0) {
    double area = radius * radius * 3.1415;
    System.out.println("The area for the" + " circle of radius " + radius + " is " + area);
} else
    System.out.println("Negative input");
```
Multiple Alternative if Statements

Indentation in Java is not required, but a good programming style.

if (score >= 90.0)
    grade = 'A';
else
    if (score >= 80.0)
        grade = 'B';
    else
        if (score >= 70.0)
            grade = 'C';
        else
            if (score >= 60.0)
                grade = 'D';
            else
                grade = 'F';

Equivalent

Indentation exception for cascading if-else-if statements:

if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
Suppose score is 70.0

The condition is false

```java
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```
Suppose score is 70.0

if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';

The condition is false
Suppose score is 70.0

if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';

The condition is true
Suppose score is 70.0

if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';

grade is C
Suppose score is 70.0

```java
if (score >= 90.0)
   grade = 'A';
else if (score >= 80.0)
   grade = 'B';
else if (score >= 70.0)
   grade = 'C';
else if (score >= 60.0)
   grade = 'D';
else
   grade = 'F';
```

Exit the if statement
Inconsistent indentation can get us confused, so the rule is that the `else` clause matches the most recent `if` clause in the same block.

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
else
  System.out.println("B");
```

Wrong indentation

This does not print anything!

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

Correct indentation
To force the else clause to match the first if clause, you must add a pair of braces:

```java
int i = 1;
int j = 2;
int k = 3;
if (i > j) {
    if (i > k)
        System.out.println("A");
} else
    System.out.println("B");
```

This code prints B.
• Adding a semicolon at the end of an if clause is a common mistake (often occurs when you use the next-line block style):

```java
if (radius >= 0);
{
    area = radius*radius*PI;
    System.out.println("The area for the circle of radius " + radius + " is " + area);
}
```

• It is not a compilation error and it is not a runtime error

• It is a logic error because ";" is a statement (the no-operation/no-op statement)
System.out.print("Enter your total cholesterol level: ");
int totalCholesterol = input.nextInt();

if (totalCholesterol >= 200)
    System.out.println("Your cholesterol is too high.");
    System.out.println("You need to lower that.");
else
    System.out.println("Good, eat away!");
System.out.print("Enter your total cholesterol level: ");
int totalCholesterol= input.nextInt();

if (totalCholesterol>= 200)
   System.out.println("Your cholesterol is too high.");
System.out.println("You need to lower that.");
else // SYNTAX ERROR HERE: this else does not match any if
   System.out.println("Good, eat away!");
What’s wrong here?

System.out.print("Enter your total cholesterol level: ");
int totalCholesterol= input.nextInt();

if (totalCholesterol>= 200) {
    // Now it is correct
    System.out.println("Your cholesterol is too high.");
    System.out.println("You need to lower that.");
} else
    System.out.println("Good, eat away!");
Why is this worse?

System.out.print("Enter your total cholesterol level: ");
int totalCholesterol = input.nextInt();

if (totalCholesterol>= 200)
    System.out.println("Your cholesterol is too high.");
    System.out.println("You need to lower that.");
System.out.print("Enter your total cholesterol level:");
int totalCholesterol = input.nextInt();

if (totalCholesterol >= 200)
    System.out.println("Your cholesterol is too high.");
System.out.println("You need to lower that.");

// NO SYNTAX ERROR
// NO RUNTIME ERROR
// It is a Bug/logical error because it says to lower
// the cholesterol even if it is fine.
Why is this worse?

System.out.print("Enter your total cholesterol level: ");
int totalCholesterol= input.nextInt();

if (totalCholesterol>= 200) { // correct
    System.out.println("Your cholesterol is too high.");
    System.out.println("You need to lower that.");
}

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
What about complex conditions?

- For example: Computing Taxes: the income tax is calculated based on the filing status and taxable income (so, we need multiple / complex logical conditions)

- There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household combined with earnings.

<table>
<thead>
<tr>
<th>Marginal Tax Rate</th>
<th>Single (0)</th>
<th>Married Filing Jointly or Qualified Widow(er) (1)</th>
<th>Married Filing Separately (2)</th>
<th>Head of Household (3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>$0 – $8,350</td>
<td>$0 – $16,700</td>
<td>$0 – $8,350</td>
<td>$0 – $11,950</td>
</tr>
<tr>
<td>15%</td>
<td>$8,351 – $33,950</td>
<td>$16,701 – $67,900</td>
<td>$8,351 – $33,950</td>
<td>$11,951 – $45,500</td>
</tr>
<tr>
<td>35%</td>
<td>$372,951+</td>
<td>$372,951+</td>
<td>$186,476+</td>
<td>$372,951+</td>
</tr>
</tbody>
</table>
# Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
</tr>
<tr>
<td>&amp; &amp;</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>exclusive or</td>
</tr>
</tbody>
</table>
Truth Table for Operator !

<table>
<thead>
<tr>
<th>p</th>
<th>!p</th>
<th>Example (assume age = 24, gender = 'F')</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
<td>!(age &gt; 18) is false, because (age &gt; 18) is true.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>!(gender != 'F') is true, because (grade != 'F') is false.</td>
</tr>
</tbody>
</table>
### Truth Table for Operator `&&`

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p1 &amp;&amp; p2</th>
<th>Example (assume age = 24, gender = 'F')</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 18) &amp;&amp; (gender == 'F') is true, because (age &gt; 18) and (gender == 'F') are both true.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 18) &amp;&amp; (gender != 'F') is false, because (gender != 'F') is false.</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>
### Truth Table for Operator ||

| p1  | p2  | p1 || p2 |
|-----|-----|------|
| false | false | false |
| false | true  | true  |
| true  | false | true  |
| true  | true  | true  |

Example (assume age = 24, gender = 'F')

- \((age > 34) \| (gender == 'F')\) is true, because \((gender == 'F')\) is true.
- \((age > 34) \| (gender == 'M')\) is false, because \((age > 34)\) and \((gender == 'M')\) are both false.
Truth Table for Operator $^\wedge$

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p1 $^\wedge$ p2</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

Example (assume age = 24, gender = 'F')

$(age > 34) \wedge (gender == 'F')$ is true, because $(age > 34)$ is false but $(gender == 'F')$ is true.

$(age > 34) \wedge (gender == 'M')$ is false, because $(age > 34)$ and $(gender == 'M')$ are both false.

$p1 \wedge p2 = p1 \neq p2$
What is the result?

```java
boolean result;
result = (5 <= 9); // true
result = !(5 <= 9); // false
result = (3.9 > 3.19); // true
result = ('a' == 'A'); // false
result = (5 <= 9 && 8 > 9); // false
result = (5 <= 9 || 8 > 9); // true
```
Logical Operators Examples

```java
System.out.println("Is " + number + " divisible by 2 and 3? " + ((number % 2 == 0) && (number % 3 == 0)));

System.out.println("Is " + number + " divisible by 2 or 3? " + ((number % 2 == 0) || (number % 3 == 0)));

System.out.println("Is " + number + " divisible by 2 or 3, but not both? " + ((number % 2 == 0) ^ (number % 3 == 0)));
```
Determining Leap Year

Consider a program that prompts the user to enter a year as an int value and checks if it is a leap year.

A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.

\[(\text{year} \mod 4 == 0 && \text{year} \mod 100 != 0)\]
\[|| \text{year} \mod 400 == 0\]
Determining Leap Year

2000: leap?
(true && false) || true = false || true = true

1900: leap?
(true && false) || false = false || false = false

2026: leap?
(false && NA) || false = false || false = false

2020: leap? (true && true) || NA= true || NA= true
The unconditional & and | Operators

- false && p2 (it does not execute p2) = false
- && is called short-cut operator
  - if the first operand is false, the conjunction is immediately false (skips the evaluation of the second operand)
- sometimes is what we want, e.g.:
  ref!="null" && ref.property==constant
The unconditional & and | Operators

- \( \text{true} \ | \ | \ p2 \) (it does not execute \( p2 \)) = true
- | | is called short-cut operator
  - if the first operand is true, the disjunction is immediately true (skips the evaluation of the second operand)
The unconditional & and | Operators

If x is 1, what is x after these expressions?

\[(x > 1) \&\& (x++ < 10)\]  
false and NA = false

\[(x > 1) \& (x++ < 10)\]  

\[(1 == x) \mid\mid (10 > x++)?\]  
true or NA = true

\[(1 == x) \mid (10 > x++)?\]  

1  
2
switch Statements

switch (var) {
    case 0:   ....;
              break;
    case 1:   ....;
              break;
    case 2:   ....;
              break;
    case 3:   ....;
              break;
    default:  ....;
}

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
switch Statements

Good for menus:
   Enter an option:
   A = burger
   B = fries
   C = exit

switch(option) {
    case 'A': System.out.println("burger"); break;
    case 'B': System.out.println("fries"); break;
    case 'C': System.out.println("exit"); break;
}
switch Statement Flow Chart

```
var is 0
break
Compute tax for married file separately
break
var is 1
break
var is 2
break
var is 3
break
default
Next Statement
```
switch Statement Rules

char, byte, short, int, String

statement(s1);
break;

case value2:
statement(s2);
break;

case valueN:
statement(sN);
break;

default:
statement(s);

value1, ..., and valueN are constant expressions of the same data type as the value of the switch-expression.

- constant = they cannot contain variables in the expression, such as x+y
The `switch` statement is used to select one of several blocks of code to execute. It evaluates a `switch` expression and matches it against a series of `case` expressions. If a match is found, the block of code associated with that case is executed. If no match is found, the `default` block is executed (if present).

**Statement Rules**

- `break` is optional, but it terminates the remainder of the `switch` statement.
- `default` is optional - executed when none of the specified cases matches the `switch`-expression.

```java
switch (switch-expression) {
    case value1:  
        statement(s)1;
        break;
    case value2:  
        statement(s)2;
        break;
    ...
    case valueN:  
        statement(s)N;
        break;
    default:      
        statement(s);
}
```

Execution is sequential, and control flows from one case to the next until a match is found or the `default` is reached.
Suppose ch is 'a':

```java
switch (ch) {
    case 'a': System.out.println(ch);
              break;
    case 'b': System.out.println(ch);
              break;
    case 'c': System.out.println(ch);
}
```
Trace switch statement 1

```java
switch (ch) {
    case 'a':
        System.out.println(ch);
        break;
    case 'b':
        System.out.println(ch);
        break;
    case 'c':
        System.out.println(ch);
}
```

ch is 'a':
Trace switch statement 1

Execute this line

```java
switch (ch) {
    case 'a':
        System.out.println(ch);
        break;
    case 'b':
        System.out.println(ch);
        break;
    case 'c':
        System.out.println(ch);
}
```

a
Trace switch statement 1

```java
switch (ch) {
    case 'a':    System.out.println(ch);
                break;
    case 'b':    System.out.println(ch);
                break;
    case 'c':    System.out.println(ch);
}
```

Execute this line

```
a
```
Trace switch statement 1

```java
switch (ch) {
    case 'a':
        System.out.println(ch);
        break;
    case 'b':
        System.out.println(ch);
        break;
    case 'c':
        System.out.println(ch);
}
```

`a`
Trace switch statement 2

Suppose ch is 'a':

```java
switch (ch) {
    case 'a':    System.out.println(ch);
    case 'b':    System.out.println(ch);
    case 'c':    System.out.println(ch);
}
```
switch (ch) {
    case 'a':
        System.out.println(ch);
    case 'b':
        System.out.println(ch);
    case 'c':
        System.out.println(ch);
}

ch is 'a':
Trace switch statement 2

```java
switch (ch) {
    case 'a':
        System.out.println(ch);
    case 'b':
        System.out.println(ch);
    case 'c':
        System.out.println(ch);
}
```

Execute this line

a
Trace switch statement 2

```java
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
}
```

aa
```java
switch (ch) {
    case 'a':    System.out.println(ch);
    case 'b':    System.out.println(ch);
    case 'c':    System.out.println(ch);
}
```

Execute this line

aaa
switch (ch) {
    case 'a':    System.out.println(ch);
    case 'b':    System.out.println(ch);
    case 'c':    System.out.println(ch);
}

aaa
Trace switch statement 3

Suppose ch is 'b':

```java
switch (ch) {
    case 'a':    System.out.println(ch);
    case 'b':    System.out.println(ch);
    case 'c':    System.out.println(ch);
}
```
Trace switch statement 3

```
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
}
```

ch is 'b':

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
Trace switch statement 3

```java
switch (ch) {
    case 'a':    System.out.println(ch);
    case 'b':    System.out.println(ch);
    case 'c':    System.out.println(ch);
}
```

Execute this line

b
Trace switch statement 3

```java
switch (ch) {
    case 'a':  System.out.println(ch);
    case 'b':  System.out.println(ch);
    case 'c':  System.out.println(ch);
}
```

bb
Trace switch statement 3

```java
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
}
```

bb
Trace switch statement 4

Suppose ch is 'c':

```
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
}
```
Trace switch statement 4

```
switch (ch) {
    case 'a':    System.out.println(ch);
    case 'b':    System.out.println(ch);
    case 'c':    System.out.println(ch);
}
```

ch is 'c':
Trace switch statement 4

Execute this line

```java
switch (ch) {
    case 'a':    System.out.println(ch);
    case 'b':    System.out.println(ch);
    case 'c':    System.out.println(ch);
}
```

C
```
switch (ch) {
    case 'a':    System.out.println(ch);
    case 'b':    System.out.println(ch);
    case 'c':    System.out.println(ch);
}
```

`c`
Conditional Operator

\[
\begin{align*}
\text{if } (x > 0) & \quad y = 1; \\
\text{else} & \quad y = -1; \\
\end{align*}
\]

is equivalent to

\[
y = (x > 0) \ ? \ 1 : \ -1;
\]

- Conditional Operator expression form:
  \((\text{boolean-expression}) \ ? \ \text{expression1} : \ \text{expression2}\)
System.out.println(
(num % 2 == 0)? num + " is even" : num + " is odd");
Operator Precedence

1. var++, var--
2. +, - (Unary plus and minus), ++var,--var
3. (type) Casting
4. ! (Not)
5. *, /, % (Multiplication, division, and remainder)
6. +, - (Binary addition and subtraction)
7. <, <=, >, >= (Comparison)
8. ==, !=; (Equality)
9. ^ (Exclusive OR)
10. && (Conditional AND) Short-circuit AND
11. || (Conditional OR) Short-circuit OR
12. =, +=, -=, *=, /=, %= (Assignment operator)
Example

Applying the operator precedence and associativity rule, the expression $3 + 4 * 4 > 5 * (4 + 3) - 1$ is evaluated as follows:

1. Inside parentheses first
2. Multiplication
3. Multiplication
4. Addition
5. Subtraction
6. Greater than

\[
3 + 4 * 4 > 5 * (4 + 3) - 1 \\
3 + 4 * 4 > 5 * 7 - 1 \\
3 + 16 > 5 * 7 - 1 \\
3 + 16 > 35 - 1 \\
19 > 35 - 1 \\
19 > 34 \\
\text{false}
\]
Operator Associativity

All binary operators except assignment operators are *left-associative*.

Example:

\[ 10 - 5 - 4 = (10 - 5) - 4 = 5 - 4 = 1 \]

The assignment operators are *right-associative*.

Example:

\[ a = b += c = 5; \]

is equivalent to \( a = (b += (c = 5)) \);
ChangeMaker Example Revisited

```java
System.out.print("Input change amount (1-99):");
originalAmount = scanner.readInt();
if (originalAmount< 1 || originalAmount> 99)
    System.out.println("ERROR: Out of range.");
else{
    numQuarters = originalAmount/ 25;
    remainder = originalAmount% 25;
    numDimes = remainder / 10;
    remainder = remainder % 10;
    numNickels = remainder / 5;
    numPennies = remainder % 5;
    if (numQuarters!= 0) // Do not print if zero
        System.out.println(numQuarters+ " quarters");
    if (numDimes!= 0)// Do not print if zero
        System.out.println(numDimes+ " dimes");
    if (numNickels!= 0)// Do not print if zero
        System.out.println(numNickels+ " nickels");
    if (numPennies!= 0)// Do not print if zero
        System.out.println(numPennies+ " pennies");
}
```

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
Nested ifs:

```java
if (numQuarters != 0) {  // Do not print if zero
    System.out.print(numQuarters + " quarter");
    if (numQuarters == 1) {  // Do not print s if one
        System.out.println();
    } else
        System.out.println("s");  // print s if more
}
```