Selections

CSE 114, Computer Science 1
Stony Brook University

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

If you assigned a negative value for radius in ComputeArea.java, then you don't want the program to compute the area.
The boolean Type and Operators

- Often in a program you need to compare values: if \( x \) is greater than \( y \)

- Java provides six comparison operators (relational operators) to compare two values: \(<, \leq, >, \geq, ==\) and \(!=\)

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

```
boolean b = (1 > 2);
```
## 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>
One-way `if` Statements

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

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

(A) Boolean Expression

(B) (radius >= 0)

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
### Condition containment necessary!

(a) Wrong

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

(b) Correct

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

### No block containment necessary!

(a)

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

(b)

```java
Equivalent

if (i > 0)
    System.out.println("i is positive");
```
The Two-way if Statement

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

```java
if (radius >= 0) {
    area = radius * radius * 3.14159;
    System.out.println("The area for the " + "circle of radius " + radius + " is " + area);
} else {
    System.out.println("Negative input");
}
```
Multiple Alternative 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';
```

Equivalent

```
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';
```
Trace if-else statement

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';
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

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
Note

The else clause matches the most recent if clause in the same block.

```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");
```

Equivalent

```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");
```
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 statement prints B.
Common Errors

- Adding a semicolon at the end of an if clause is a common mistake.

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

- It is not a compilation error or a runtime error, it is a logic error.
- This error often occurs when you use the next-line block style.
if (number % 2 == 0)
    even = true;
else
    even = false;

Equivalent

boolean even = number % 2 == 0;
Problem: Computing Taxes

The US federal personal income tax is calculated based on the filing status and taxable income.

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

<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>
Problem: Computing Taxes, cont.

if (status == 0) {
    // Compute tax for single filers
    ...
} else if (status == 1) {
    // Compute tax for married file jointly
    ...
} else if (status == 2) {
    // Compute tax for married file separately
    ...
} else if (status == 3) {
    // Compute tax for head of household
    ...
} else {
    // Display wrong status
    ...
}
## 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 `^`

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p1 ^ 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; 34) ^ (gender == 'F') is true, because (age &gt; 34) is false but (gender == 'F') is true.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>(age &gt; 34)</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>
Examples

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)));

The unconditional & and | Operators

- The & operator works exactly the same as the && operator, and the |
  operator works exactly the same as the || operator with one exception:
- the & and | operators always evaluate both operands
The unconditional & and | Operators

If x is 1, what is x after this expression?

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

If x is 1, what is x after this expression?

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

How about:

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

\[(1 == x) \| (10 > x++)?\]
This program first 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.

\[(\text{year} \mod 4 == 0 && \text{year} \mod 100 != 0) || (\text{year} \mod 400 == 0)\]
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)
The `switch` statement flow chart is shown in the image. It illustrates the decision-making process for different cases based on the value of `var`. Here is the description of the flow chart:

1. If `var` is 0, it goes to the `break` statement.
2. If `var` is 1, it goes to the `break` statement.
3. If `var` is 2, it goes to the `break` statement.
4. If `var` is 3, it goes to the `break` statement.
5. If `var` is not any of the above, it goes to the `default` statement.

After the `default` statement, it goes to the `Next Statement`. This chart demonstrates how the `switch` statement in programming allows for multiple conditions to be evaluated simultaneously, choosing the appropriate block of code to execute based on the value of the variable `var`.
char, byte, short, or int String

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

switch (switch-expression) {
    case value1:    statement(s)1;
        break;
    case value2:    statement(s)2;
        break;
    ...
    case valueN:    statement(s)N;
        break;
    default:        statement(s);
}
**switch 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 in sequential order
Trace switch statement

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);
}
```
Trace switch statement

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

ch is 'a':

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

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

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

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

```java
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
}
```
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 2

```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
Trace switch statement 2

```
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
```java
switch (ch) {
    case 'a': System.out.println(ch);
              break;
    case 'b': System.out.println(ch);
              break;
    case 'c': System.out.println(ch);
}
```
if (x > 0)
    y = 1
else
    y = -1;

is equivalent to

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

(\text{boolean-expression}) \ ? \ \text{expression1} : \ \text{expression2}

```
System.out.println(
    (num % 2 == 0)? num + "is even" : num + "is odd" );
```
Operator Precedence

- `var++`, `var--`
- `+`, `-` (Unary plus and minus), `++var`, `--var`
- (type) Casting
- `!!` (Not)
- `*`, `/`, `%` (Multiplication, division, and remainder)
- `+`, `-` (Binary addition and subtraction)
- `<`, `<=`, `>`, `>=` (Comparison)
- `==`, `!=`; (Equality)
- `^` (Exclusive OR)
- `&&` (Conditional AND) Short-circuit AND
- `||` (Conditional OR) Short-circuit OR
- `=`, `+=`, `-=`, `*=`, `/=`, `%=` (Assignment operator)

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

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

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

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

The assignment operators are right-associative:

\[ a = b += c = 5 \text{ is equivalent to } a = (b += (c = 5)) \]
Selection extras

- What is the result?

```java
boolean result;
result = (5 <= 9);
result = (3.9 > 3.19);
result = ('a' == 'A');
```
Selection extras

- What is the result?

```java
boolean result;
result = (5 <= 9);  // result is true
result = (3.9 > 3.19);  // result is true
result = ('a' == 'A');  // result is false
```
• What is the result?

```java
boolean result;
result = !(5<=9);
result = (5<=9 && 8>9);
result = (5<=9 || 8>9);
```
What is the result?

```java
boolean result;
result = !(5<=9); // result is false
result = (5<=9 && 8>9); // result is false
result = (5<=9 || 8>9); // result is true
```
PrintStream out = System.out;
out.print("Enter your total cholesterol level: ");
int totalCholesterol = keyboard.nextInt();

if (totalCholesterol >= 200)
    out.println("Your cholesterol is too high.");
    out.println("You need to lower that.");
else
    out.println("Good, eat away!");
What’s wrong here?

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

if (totalCholesterol >= 200)
    out.println("Your cholesterol is too high.");
    out.println("You need to lower that.");
else
    out.println("Good, eat away!");
What’s wrong here?

```java
PrintStream out = System.out;
out.print("Enter your total cholesterol level: ");
int totalCholesterol = keyboard.nextInt();

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

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

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

```java
PrintStream out = System.out;
out.print("Enter your total cholesterol level: ");
int totalCholesterol = keyboard.nextInt();

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

// No compiler error
// Bug, logical error
```
PrintStream out = System.out;
out.print("Enter your total cholesterol level: ");
int totalCholesterol = keyboard.nextInt();

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

• Don’t use ‘==’ to compare Strings
  • it compares their memory addresses and not actual strings (character sequences)
• Instead use the equals method supplied by the String class
Comparing Strings

```java
String word1 = new String("Hello");
String word2 = new String("Hello");
if (word1 == word2){
    System.out.println(true);
} else {
    System.out.println(false);
}
```

Result?
Comparing Strings

```java
String word1 = new String("Hello");
String word2 = new String("Hello");
if (word1 == word2) {
    System.out.println(true);
} else {
    System.out.println(false);
}
```

- Two different addresses
  
  false
Comparing Strings

String word1 = new String("Hello");
String word2 = new String("Hello");
if (word1.equals(word2)){
    System.out.println(true);
} else {
    System.out.println(false);
}
Comparing Strings

String word1 = "Hello";
String word2 = "Hello";
if (word1 == word2){
    System.out.println(true);
} else {
    System.out.println(false);
}

true

- Only one instance of “Hello” is stored
  - word1 and word2 will have the same address
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");
}
• 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
}
```