Computer Science II

Professor Tom Ellman

Lecture 4
Conditionals

• Control in a Java program normally proceeds from one statement to the next.
• Conditional statements and expressions alter the normal flow of control.
• Conditionals include \texttt{if}, \texttt{if-else}, and \texttt{switch} statements.
• Conditionals also include the \texttt{? :} operator.
General forms of *if* Statement

if (<Boolean-Expression>)
<Statement>

if (<Boolean-Expression>)
{
  <Statement1>
  <Statement2>
  ...
  <StatementN>
}
Example of if Statement

```java
final int SUNDAY = 1, MONDAY = 2, TUESDAY = 3;
final int WEDNESDAY = 4, THURSDAY = 5, FRIDAY = 6;
final int SATURDAY = 7;
int time, day;
// ... Initialization of time and day Omitted ...
if ( (time >= 1200) && (time <= 1315) 
    && ((day == TUESDAY) || (day == THURSDAY)) )
{
    System.out.println("Go to 102 class.");
}
```
# Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Expression&gt; == &lt;Expression&gt;</code></td>
<td>Equality</td>
</tr>
<tr>
<td><code>&lt;Expression&gt; != &lt;Expression&gt;</code></td>
<td>Inequality</td>
</tr>
<tr>
<td><code>&lt;Number&gt; Op &lt;Number&gt;</code> (Op: <code>&lt;</code> <code>&lt;=</code> <code>&gt;</code> <code>&gt;=</code>)</td>
<td>Comparison</td>
</tr>
</tbody>
</table>
## Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Boolean&gt; &amp;&amp; &lt;Boolean&gt;</code></td>
<td>Logical conjunction. (AND)</td>
</tr>
<tr>
<td>`&lt;Boolean&gt;</td>
<td></td>
</tr>
<tr>
<td><code>! &lt;Boolean&gt;</code></td>
<td>Logical Negation (NOT)</td>
</tr>
</tbody>
</table>
General forms of \textbf{if–else} Statement

\begin{itemize}
\item if (<Boolean-Expression>)
  \begin{itemize}
  \item <Statement>
  \end{itemize}
\item else <Statement>
\end{itemize}

\begin{itemize}
\item if (<Boolean-Expression>)
  \begin{itemize}
  \item \{ <Statement1> \ldots <StatementN> \}
  \end{itemize}
\item else \{ <Statement1> \ldots <StatementM> \}
\end{itemize}
Example of \texttt{if-else} Statement

```java
final int SUNDAY = 1, MONDAY = 2, TUESDAY = 3;
final int WEDNESDAY = 4, THURSDAY = 5, FRIDAY = 6;
final int SATURDAY = 7;
int time, day;
// ... Initialization of time and day Omitted ...
if ( (time >= 1200) && (time <= 1315)
    && ((day == TUESDAY) || (day == THURSDAY)) )
{
    System.out.println("Go to 102 class.");
}
else {
    System.out.println("Don’t go to 102 class.");
}
```
Nested if-else Statements

```java
final int JANUARY = 1, FEBRUARY = 2, MARCH = 3, APRIL = 4;
final int MAY = 5, JUNE = 6, JULY = 7, AUGUST = 8;
final int SEPTEMBER = 9, OCTOBER = 10, NOVEMBER = 11;
final int DECEMBER = 12;

int month;

// ... Initialization of month Omitted ...

if (month==DECEMBER || month==JANUARY || month==FEBRUARY)
    System.out.println("It’s cold!");
else if (month==SEPTEMBER || month==OCTOBER || month==NOVEMBER ||
        month==MARCH || month==APRIL || month==MAY)
    System.out.println("Its cool or warm!");
else if (month==JUNE || month==JULY || month==AUGUST)
    System.out.println("Its hot!");
```
One Form of the `switch` Statement

```
switch (<Integer-Expression>)
{
    case <Integer-Constant>: <Statement1> . . . <StatementN>
        break;
    ...
    case <Integer-Constant>: <Statement1> . . . <StatementN>
        break;
}
```
Example of `switch` Statement

```java
final int GREEN = 1, YELLOW = 2, RED = 3;
int trafficSignal;

// ... Initialization of trafficSignal omitted ...

switch (trafficSignal) {
    case GREEN:    System.out.println("Go!");
                   break;
    case YELLOW:   System.out.println("Maybe stop, maybe go!");
                   break;
    case RED:      System.out.println("Stop!");
                   break;
}
```
Another Form of the `switch` Statement

switch (<Integer-Expression>)
{
    case <Integer-Constant>: <Statement1> . . . <StatementN>
    break;
    
    . . .
    
    case <Integer-Constant>: <Statement1> . . . <StatementN>
    break;

    default: <Statement1> . . . <StatementN>
    break;

}
Example of `switch` Statement

```java
final int VEGETARIAN = 1, KOSHER = 2;

int mealType;

// ... Initialization of mealType omitted ...

switch (mealType)
{

    case VEGETARIAN:  System.out.println("Salad.");
                       break;

    case KOSHER:      System.out.println("Salmon.");
                       break;

    default:          System.out.println("Chicken.");
                       break;

}
```
Another Form of the `switch` Statement

switch (<Integer-Expression>)
{
    case <Integer-Constant>:
    case <Integer-Constant>:
    case <Integer-Constant>:

        case <Integer-Constant>: <Statement1> . . . <StatementN>
        break;

        ....

    }
Example of `switch` Statement

```java
final int JANUARY = 1, FEBRUARY = 2, MARCH = 3, APRIL = 4;
final int MAY = 5, JUNE = 6, JULY = 7, AUGUST = 8;
final int SEPTEMBER = 9, OCTOBER = 10, NOVEMBER = 11;
final int DECEMBER = 12;

int month;

// ... Initialization of month omitted ...

switch (month)
{
    case SEPTEMBER: case APRIL: case JUNE: case NOVEMBER:
        System.out.println("30 Days"); break;
    case JANUARY: case MARCH: case MAY: case JULY:
    case AUGUST: case OCTOBER:
        System.out.println("31 Days"); break;
    case FEBRUARY:
        System.out.println("28 Or 29 Days"); break;
}
Comments on **switch** Statement

- The "**break;**" statement may be included or omitted.
- If omitted, then control falls through to the next case.
- A sequence of **case** clauses may all be handled by a single statement.
General form and Example of Expression using the `?:` Operator

```
<Boolean-Expression> ? <Type-Expression> : <Type-Expression>

int maximum(int x, int y) {
    return x >= y ? x : y;
}

int minimum(int x, int y) {
    return x <= y ? x : y;
}
```
Blocks

• A portion of the body of a method enclosed by { … } braces.
• Groups a sequence of statements together to form a single, composite statement.
• May be used in bodies of conditional statements and other control-flow constructs.
• May be used to impose a hierarchical structure on a method definition.
Hierarchic Block Structure
Visibility and Scope

- A variable may be declared anywhere within a block.
- The variable may be referenced anywhere from the point of declaration, to the end of the block.
- Within that region, the variable is said to “visible” and “in scope”.
- Outside of that region, the variable is said to be “not visible” and “out of scope”.
Scope in a Block Hierarchy

{
  // Variable "name" is out of scope here.
  {
    // Variable "name" is out of scope here.
    String name = "FooBar";
    // Variable "name" is in scope here.
    {
      // Variable "name" is in scope here.
    }
    // Variable "name" is in scope here.
  }
  // Variable "name" is in scope here.
}
// Variable "name" is in scope here.

// Variable "name" is out of scope here.
// Variable "name" is out of scope here.
package blockdemo1;

public class BlockDemo1 {
    public static void main(String[] args) {
        
        String name = "Bar";
        System.out.println("Inner reference to \"name\": " + name);
        System.out.println("Outer reference to \"name\": " + name);
    }
}

//cannot find symbol
//symbol : variable name
//class blockdemo1.BlockDemo1
package blockdemo2;

public class BlockDemo2 {

    public static void main(String[] args) {

        String name = "Bar";
        System.out.println("Inner reference to " + name);
    }

    String name = "Foo";
    System.out.println("Outer reference to " + name);

}

Program Output:    Inner reference to "name": Bar
                  Outer reference to "name": Foo
package blockdemo3;

public class BlockDemo3 {
    public static void main(String[] args) {
        String name = "Foo";
        System.out.println("Outer reference to \"name\": " + name);
        {
            String name = "Bar";
            System.out.println("Inner reference to \"name\": " + name);
        }
    }
}

//name is already defined in main(java.lang.String[])
package blockdemo4;

public class BlockDemo4 {
    public static void main(String[] args) {
        String name = "Foo";
        System.out.println("Outer reference to \"name\": " + name);
        {
            name = "Bar";
            System.out.println("Inner reference to \"name\": " + name);
        }
    }
}

Program Output: Outer reference to "name": Foo
                Inner reference to "name": Bar