



CMSC 131

Object-Oriented Programming I

Conditionals, Block Statements, Style

**Dept of Computer Science
University of Maryland College Park**

This material is based on material provided by Ben Bederson, Bonnie Dorr,
Fawzi Emad, David Mount, Jan Plane

Overview

- ▶ Conditionals
- ▶ Block Statements
- ▶ Style

Blocks

- ▶ What happens?

```
if (i > 10)
    i = 10;
    saturate = true;
else
    k = 100;
```

- ▶ Desired → both `i`, `saturate` are set only when `i > 10`
- ▶ Actual → syntax error
 - Only one statement can be associated with `if`
 - The `saturate` assignment statement is not part of the `if`
 - The `else` can't find the `if` it belongs to
- ▶ **Blocks** solve this problem also

What Blocks Are

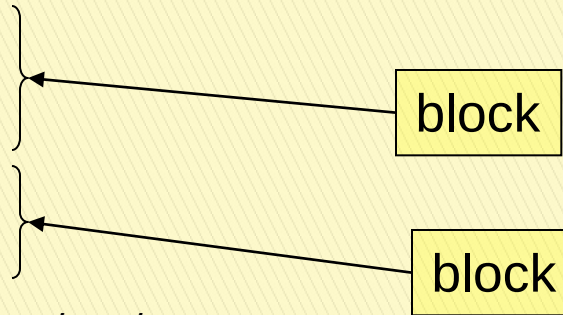
- ▶ Blocks are sequences of statements “glued together” into one

- ▶ Form:

```
{  
  <statement 1>;  
  <statement 2>;  
  ...  
}
```

- ▶ Example:

```
if (i > 10) {  
    i = 10;  
    saturate = true;  
} else {  
    i = i+1;  
}
```



- ▶ if, if-else, {...} are *statement constructors*
 - They take statement(s) and convert them into a new statement
 - Implications: if statements, etc. can also appear inside (“**be nested within**”) one another

Issues with if-else

- ▶ Nested If/Elses can be Ugly and Confusing!
 - Indent and block carefully
 - **Example:** NestingExample.java
- ▶ The “Dangling Else” Problem
 - Java rule → else is associated with “innermost” possible if
 - **Example:** BadExample.java (example of bad indenting)
- ▶ Cascading Elses
 - **Example:** Cascading1.java, Cascading2.java (Improved version of Cascading1.java), Cascading3.java
- ▶ WE WILL USE { ... } FOR ALL IF, IF-ELSE, IF-ELSE-IF, STATEMENTS

Scope rules/Initialization

- ▶ Variables can be declared anywhere in a Java program
- ▶ When are the declarations active?
 - After they are executed
 - *Only inside the block in which they are declared*
- ▶ **Scope rules** formalize which variable declaration are active
 - **Global variables**: scope is entire program
 - **Local variables**: scope is a block
- ▶ **Example:** Initialization1.java (problem),
Initialization2.java
- ▶ **Example:** Initialization3.java

Named Constants

- ▶ If same value should be used in several places, how to ensure consistency?
 - i.e. Check on temperature may be performed more than once
 - i.e. Same prompt might be printed in several places
- ▶ `final int MAX_OK_TEMP = 99;`
 - Just like a regular variable declaration/initialization, except...
 - Special term `final`
 - Necessity of initial value
 - Any valid variable name will work, but convention is to use all capitals
- ▶ If you are not using constants then you have “magic numbers”
- ▶ Difference from non-final variables: assignment attempt leads to error!
- ▶ **literals** (= named values)

e.g.

```
if (temp >= 212 || temp <= 32) ...  
if (temp >= BOILING || temp <= FREEZING)
```

e.g.

```
System.out.print ("Enter integer: ");  
System.out.print (PROMPT);
```

Style for Projects

- ▶ You must use **meaningful variable names**
 - It must tell the purpose of that variable → what it is meant to hold
 - It can not have so much abbreviation that only you can read it
- ▶ You must use Java convention indenting and brace placement
- ▶ Java convention of capitalization of identifiers
 - Variables and methods start with lower case
 - Classes and interfaces start with upper case
 - Variables, methods, classes and interface use camelCase
 - Constants are all uppercase with underscores between words
- ▶ You must have “Fully Blocked” if statements and looping structures
- ▶ You must have all lines less than or equal to 80 columns of text
- ▶ You must use "**named constants**" for any literal values that will not change during program execution

Meaningful Variable Names

- ▶ Choose names for your variables to reflect their purpose not their type
- ▶ Make it readable to someone else
- ▶ Help prevent mistakes in order of the relational operators
- ▶ Avoid names that start with \$ (usually reserved for “system level” variables)

Bad	Good
<code>typedValue == 5</code>	<code>menuOption == 5</code>
<code>integer > 13</code>	<code>age > 13</code>
<code>input1 > 45 && input2 > 100</code>	<code>height > 45 && weight > 100</code>
<code>val1 < 100 val1 > 999</code>	<code>non3dgt < 100 non3dgt > 999</code>

Suggestions for Writing Programs

- ▶ **Design** → Make sure you first come up with a plan/design for your code (e.g., using pseudocode)
- ▶ **Do not wait until the last minute** → Code implementation can be unpredictable
- ▶ **Incremental code development** → Fundamental principle in computer programming. Write a little bit of code, and make sure it works before moving forward
- ▶ **Don't make assumptions** → If you are not clear about a language construct, write a little program to familiarize yourself with the construct
- ▶ **Good Indentation** → From the get-go use good indentation as it will allow you to understand your code better

Suggestions For Writing Programs

- ▶ **Good variable names** → Use good variable names from the beginning (do not use x and y and then change them to meaningful names before submitting the project)
- ▶ **Testing**
 - Test your code with simple cases first
 - Test as you develop your code
- ▶ **Keep backups** → As you make significant progress in your development, make the appropriate backups
- ▶ Trace your code
- ▶ Use a debugger or specialized tools
 - Findbugs → <http://findbugs.sourceforge.net/>
- ▶ **Take breaks** → If you cannot find a bug take a break and come back later