Lecture Set #3: Conditional and Iterative Structures

Control Structures

- if branching
- if / else branching
- logical operators
- nesting of control structures
- proper indenting and spacing conventions
- java identifier naming conventions
- named constants
- while loop
- do-while loop
- for loop

Control Flow and Conditionals

Control flow: the order in which statements are executed

General rule: top to bottom

Several Control Structures that change that

Conditional statements: permit control flow to be dependent on (true/false) conditions

`if`

`if-else`

if and if-else

The if and if-else statements should have the following form:

if (*condition*) {
    statements;
}

tests the condition

if true statement is done; otherwise it is skipped

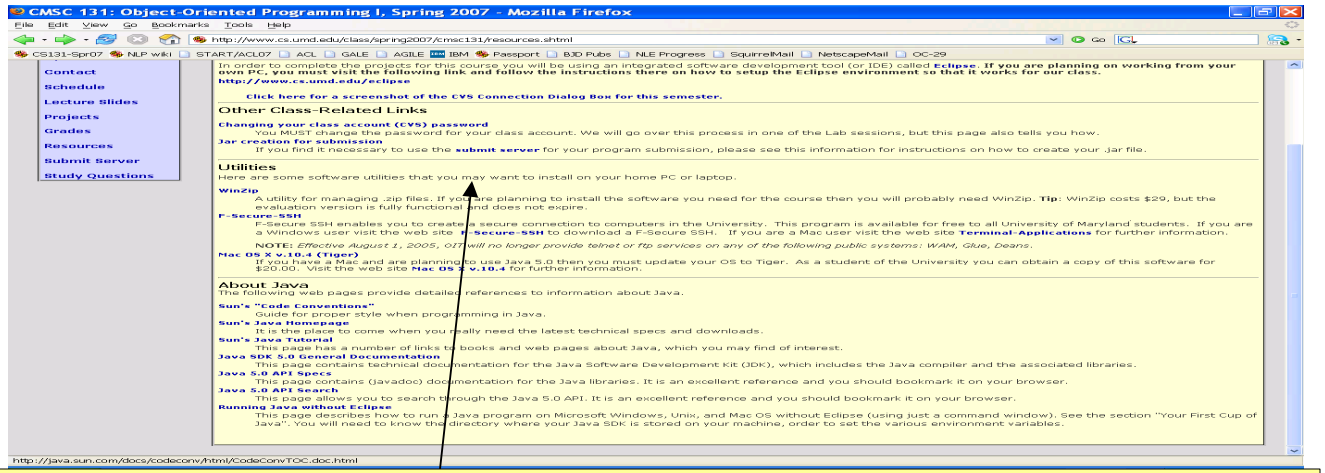
if (*condition*) {
    statements1;
} else {
    statements2;
}

tests the condition

if true, statements1 is done; otherwise statements2 is done

Java and White Space

You can add: carriage returns, spaces, tabs
wherever you want in Java
Properly used, this makes your program easier to read and understand



http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

Logical Operators

Used for forming more complex conditions.

"and" &&

```
    if ( temp >= 97 && temp <= 99 ) {
        System.out.println( "Patient is healthy" );
}
```

"or" ||

```
    if ( months >= 3 || miles >= 3000 ) {
        System.out.println( "Change your oil" );
}
```

"not": !

```
    if ( ! phone.equals( "301-555-1212" ) ) {
        System.out.println( "Sorry, wrong number" );
}
```

Blocks

What happens?

```
if (i > 10)
   i = 10;
   saturate = true;
```

Desired: both `i`, `saturate` are set only when `i` > 10
Actual: only the i=10 statement is dependant

Only one statement can be associated with `if`

The `saturate` assignment statement is not part of the `if`

Blocks solve this problem

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

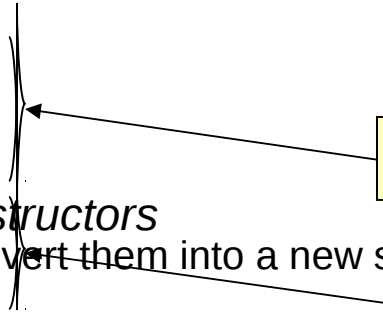
What Blocks Are

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

The “Dangling Else” Problem
    Java rule: else is associated with “innermost” possible if

Cascading Elses

WE WILL USE { … } FOR ALL IF, IF-ELSE, IF-ELSE-IF, STATEMENTS

In 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

- the indenting show the purpose in nesting
- with braces in the “Java determined” places with respect to the lines of code

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.

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

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);
```

Naming Rules and Conventions

What is legal for variable names?

Letters, digits, $, _

Can't start variable name with digit

Avoid reserved words

Avoid names starting or ending with $ or _

Use camelCase:

Variables & Methods use lower-case for first letter

Classes/Interfaces use upper-case for first letter

Naming Conventions: Standards developed over time.

Variables and methods: Start with lowercase, and use uppercase for each new word (including instance final variables):

dataList2 myFavoriteMartian showMeTheMoney

Class names: Start with uppercase and uppercase for each new word:

String JOptionPane MyFavoriteClass

Named class constants (static variables whose value never change): All uppercase with underscores between words:

MAX_LENGTH DAYS_PER_WEEK BOILING_POINT

Make variable names not too long, not too short

Bad: crtItm

Bad : theCurrentItemBeingProcessed

Good : currentItem

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

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

Loops in Java

So far our programs execute every program statement at most once

Often, we want to perform operations more than once:

- "Sum all numbers from 1 to 10"
- "Repeatedly prompt user for input"

Loops allow statements to be executed multiple times.

Loop types in Java:

- while
- do-while
- for

Called "iteration"

while and do-while Loops

while and **do-while** loops contain:

A statement, called the body

A boolean condition

Idea: the body is executed one more time as long as the condition is true

while-loop: The condition is tested before each body execution

- **while (condition) {**
- ** body**
- **}**

do-while-loop: The condition is tested after each body execution

- **do{**
- ** body**
- **} while (condition);**

Main difference: do-while loop bodies always executed at least once because it is "bottom tested" rather than "top tested"

Types of loops

indefinite iteration

usually tests something that is coming from outside the loop structure (e.g. input)

needs to eventually change from true to false

counted iteration

something that is controlled inside the loop

to start at some value and count up or down until some set ending point

for loop

for-loop: The counter is set, the condition is tested before each body execution, the update is performed at the end of each iteration

- **for (initialization; condition; update) {**
- **body**
- **}**

Usually used for counted loops, but any of the parts can be left empty.

Infinite Loops

Loops can run forever if condition never becomes false
Be careful when programming loops!

- Add statements for termination into loop body first
- Often these statements are at end of body
- e.g.

```
while (i <= 10) {
  System.out.println(i);
  i = i + 1;
}
```

Variables, Blocks and Scoping

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 when
    Global variables: scope is entire program

    Local variables: scope is a block

Nested Loops

while, do-while are statement constructors (like if and if-else: they use blocks)
Loops can thus be used inside other loops!

Nesting Example

```
public class NestedLoops {

  public static void main(String[] args) {

    int rowNumber = 1;
    while (rowNumber < 10) {
      int colNumber = 1;
      while (colNumber < 10) {
        System.out.print((rowNumber + colNumber) % 2);
        colNumber = colNumber + 1;
      }
      System.out.println();
      rowNumber = rowNumber + 1;
    }
  }
}
```