Course Material Usage Rules

• **PowerPoint slides for use only in full-semester, for-credit courses at degree-granting institutions**
  – Slides *not* permitted for use in commercial training courses except when taught by coreservlets.com (see [http://courses.coreservlets.com](http://courses.coreservlets.com)).

• **Slides can be modified by instructor**
  – Please retain this notice and attribution to coreservlets.com

• **Instructor can give PDF or hardcopy to students, but should protect PowerPoint files**
  – *This slide is suppressed in Slide Show mode*
Basic Java Syntax

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/java5.html

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.
For live Java EE training, please see training courses at http://courses.coreservlets.com/.

Servlets, JSP, Struts, JSF 1.x, JSF 2.0, Ajax (with jQuery, Dojo, Prototype, Ext-JS, Google Closure, etc.), GWT 2.0 (with GXT), Java 5, Java 6, SOAP-based and RESTful Web Services, Spring, Hibernate/JPA, and customized combinations of topics.

Taught by the author of Core Servlets and JSP, More Servlets and JSP, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization. Contact hall@coreservlets.com for details.
Topics in This Section

• Basics
  – Creating, compiling, and executing simple Java programs
• Accessing arrays
• Looping
• Indenting Code
• Using if statements
• Comparing strings
• Building arrays
  – One-step process
  – Two-step process
  – Using multidimensional arrays
• Performing basic mathematical operations
• Reading command-line input
Basics

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.
Getting Started: Syntax

• Example

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world.");
    }
}
```

• Details
  – Processing starts in main
    • Routines usually called "methods," not "functions."
  – Printing is done with System.out.print...
    • System.out.println, System.out.print, System.out.printf
Getting Started: Execution

- **File:** HelloWorld.java
  
  ```java
  public class HelloWorld {
      public static void main(String[] args) {
          System.out.println("Hello, world.");
      }
  }
  ```

- **Compiling**
  
  > javac HelloWorld.java

- **Executing**
  
  > java HelloWorld
  Hello, world.
More Basics

• **Use + for string concatenation**

• **Arrays are accessed with [ ]**
  – Array indices are zero-based
  – The argument to `main` is an array of strings that correspond to the command line arguments
    • `args[0]` returns first command-line argument
    • `args[1]` returns second command-line argument, etc.
    • Error if you try to access more args than were supplied

• **The length field**
  – Gives the number of elements in any array
    • Thus, `args.length` gives the number of command-line arguments
    • Unlike in C/C++, the name of the program is not inserted into the command-line arguments
Command-line Arguments

- Useful for learning and testing
  - Command-line args are useful for practice
  > java Classname arg1 arg2 ...
Example

• **File: ShowTwoArgs.java (naïve version)**

```java
public class ShowTwoArgs {
    public static void main(String[] args) {
        System.out.println("First arg: "+
            args[0]);
        System.out.println("Second arg: "+
            args[1]);
    }
}
```

**Oops! Crashes if there are not at least two command-line arguments. The code should have checked the length field, like this:**

```java
if (args.length > 1) {
    doThePrintStatements();
} else {
    giveAnErrorMessage();
}
```
Example (Continued)

- Compiling
  > javac ShowTwoArgs.java

- Executing
  > java ShowTwoArgs Hello Class
  First args Hello
  Second arg: Class

> java ShowTwoArgs
[Error message]
Loops
 Looping Constructs

- **for/each**
  ```
  for(variable: collection) {
    body;
  }
  ```

- **for**
  ```
  for(init; continueTest; updateOp) {
    body;
  }
  ```

- **while**
  ```
  while (continueTest) {
    body;
  }
  ```

- **do**
  ```
  do {
    body;
  } while (continueTest);
  ```
public static void listEntries(String[] entries) {
    for(String entry: entries) {
        System.out.println(entry);
    }
}

• Result
String[] test = {"This", "is", "a", "test"};
listEntries(test);

This
is
a
test
For Loops

public static void listNums1(int max) {
    for(int i=0; i<max; i++) {
        System.out.println("Number: " + i);
    }
}

• Result
  listNums1(4);

  Number: 0
  Number: 1
  Number: 2
  Number: 3
While Loops

```java
public static void listNums2(int max) {
    int i = 0;
    while (i < max) {
        System.out.println("Number: " + i);
        i++;  // "++" means "add one"
    }
}
```

• Result

listNums2(5);

Number: 0
Number: 1
Number: 2
Number: 3
Number: 4
public static void listNums3(int max) {
    int i = 0;
    do {
        System.out.println("Number: "+i);
        i++;
    } while (i < max);
    // ^ Don’t forget semicolon
}

• Result
  listNums3(3);
  Number: 0
  Number: 1
  Number: 2
Class Structure and Formatting

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.
Defining Multiple Methods in Single Class

public class LoopTest {
    public static void main(String[] args) {
        String[] test =
            { "This", "is", "a", "test"};
        listEntries(test);
        listNums1(5);
        listNums2(6);
        listNums3(7);
    }

    public static void listEntries(String[] entries) {...}
    public static void listNums1(int max) {...}
    public static void listNums2(int max) {...}
    public static void listNums3(int max) {...}
}

These methods say “static” because they are called directly from “main”. In the next two sections on OOP, we will explain what “static” means and why most regular methods do not use “static”. But for now, just note that methods that are directly called by “main” must say “static”.
Indentation: blocks that are nested more should be indented more

- Yes

```plaintext
blah;
blah;
for(...) {
    blah;
    blah;
    for(...) {
        blah;
        blah;
    }
}
```

- No

```plaintext
blah;
blah;
for(...) {
    blah;
    blah;
    for(...) {
        blah;
        blah;
    }
}
```
Indentation: blocks that are nested the same should be indented the same

• Yes

```plaintext
blah;
blah;
for(...) {
  blah;
  blah;
  for(...) {
    blah;
    blah;
  }
}
```

• No

```plaintext
blah;
blah;
for(...) {
  blah;
  blah;
  for(...) {
    blah;
    blah;
  }
}
```
Indentation: Number of spaces and placement of braces is a matter of taste

• OK
  blah;
  blah;
  for(...) {
    blah;
    blah;
    for(...) {
      blah;
      blah;
    }
  }

• OK
  blah;
  blah;
  for(...) {
    blah;
    blah;
    for(...) {
      blah;
      blah;
    }
  }

• OK
  blah;
  blah;
  for(...) {
    blah;
    blah;
    for(...) {
      blah;
      blah;
    }
  }
Conditionals and Strings

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.
If Statements

• **Single Option**
  
  ```
  if (boolean-expression) {
    statement;
  }
  ```

• **Multiple Options**
  
  ```
  if (boolean-expression) {
    statement1;
  } else {
    statement2;
  }
  ```
Boolean Operators

- **==, !=**
  - Equality, inequality. In addition to comparing primitive types, `==` tests if two objects are identical (the same object), not just if they appear equal (have the same fields). More details when we introduce objects.

- **<, <=, >, >=**
  - Numeric less than, less than or equal to, greater than, greater than or equal to.

- **&&, ||**
  - Logical AND, OR. Both use short-circuit evaluation to more efficiently compute the results of complicated expressions.

- **!**
  - Logical negation.
Example: If Statements

```java
public static int max(int n1, int n2) {
    if (n1 >= n2) {
        return(n1);
    } else {
        return(n2);
    }
}
```
• **Basics**
  
  – String is a real class in Java, not an array of characters as in C and C++.
  
  – The String class has a shortcut method to create a new object: just use double quotes
    
    • This differs from normal objects, where you use the **new** construct to build an object

• **Use equals to compare strings**

  – **Never use** `==`

• **Many useful built-in methods**

  – contains, `startsWith`, `endsWith`, `indexOf`, `substring`, `split`, `replace`, `replaceAll`
    
    • Note: can use regular expressions, not just static strings
  
  – `toUpperCase`, `toLowerCase`, `equalsIgnoreCase`
Common String Error: Comparing with ==

```java
public static void main(String[] args) {
    String match = "Test";
    if (args.length == 0) {
        System.out.println("No args");
    } else if (args[0] == match) {
        System.out.println("Match");
    } else {
        System.out.println("No match");
    }
}
```

- Prints "No match" for all inputs
  - Fix:
    ```java
    if (args[0].equals(match))
    ```
String and ...

- String $\rightarrow$ Immutable

- StringBuffer $\rightarrow$ Mutable, Synchronized

- StringBuilder $\rightarrow$ Mutable, Unsynchronized
Arrays
Building Arrays: One-Step Process

- Declare and allocate array in one fell swoop

\[
\text{type}[] \ \text{var} = \{ \ \text{val1}, \ \text{val2}, \ldots, \ \text{valN} \};
\]

- Examples:

```java
int[] values = \{ 10, 100, 1000 \};
String[] names = \{ "Joe", "Jane", "Juan" \};
Point[] points = \{ \text{new Point}(0, 0), \text{new Point}(1, 2), \text{new Point}(3, 4) \};
```
Building Arrays: Two-Step Process

• **Step 1: allocate an array of references:**
  
  ```java
  type[] var = new type[size];
  ```
  
  – E.g.:  
  ```java
  int[] primes = new int[7];
  String[] names = new String[someArray.length];
  ```

• **Step 2: populate the array**
  ```java
  primes[0] = 2;                    names[0] = "Joe"
  primes[1] = 3;                    names[1] = "Jane"
  ```
  
  etc.

• **If you fail to populate an entry**
  – Default value is 0 for numeric arrays
  – Default value is `null` for object arrays
Array Performance Problems

- **For very large arrays, undue paging can occur**
  - Array of references (pointers) allocated first
  - Individual objects allocated next
  - Thus, for very large arrays of objects, reference and object can be on different pages, resulting in swapping for each array reference
  - Example
    ```java
    String[] names = new String[10000000];
    for(int i=0; i<names.length; i++) {
        names[i] = getNameFromSomewhere();
    }
    ```

- **Problem does not occur with arrays of primitives**
  - I.e., with arrays of `int`, `double`, and other types that start with lowercase letter
  - Because system stores values directly in arrays, rather than storing references (pointers) to the objects
Multidimensional Arrays

- Multidimensional arrays
  - Implemented as arrays of arrays

  ```java
  int[][] twoD = new int[64][32];
  
  String[][] cats = {{"Caesar", "blue-point"},
                     {"Heather", "seal-point"},
                     {"Ted", "red-point"}};
  ```

- Note:
  - Number of elements in each row need not be equal

  ```java
  int[][] irregular = {{1},
                      {2, 3, 4},
                      {5},
                      {6, 7}};
  ```
public class TriangleArray {
    public static void main(String[] args) {

        int[][] triangle = new int[10][];

        for(int i=0; i<triangle.length; i++) {
            triangle[i] = new int[i+1];
        }

        for (int i=0; i<triangle.length; i++) {
            for(int j=0; j<triangle[i].length; j++) {
                System.out.print(triangle[i][j]);
            }
            System.out.println();
        }
    }
}
TriangleArray: Result

> java TriangleArray

0
00
000
0000
00000
000000
0000000
00000000
000000000
0000000000
00000000000
Math and Input
Basic Mathematical Routines

• Very simplest routines use builtin operators
  – +, -, *, /, ^, %
  – Be careful with / on int and long variables

• Static methods in the Math class
  – So you call Math.cos(...), Math.random(), etc.
    • Most operate on double precision floating point numbers
  – Simple operations: Math.pow(), etc.
    • pow (x^y), sqrt (√x), cbrt, exp (e^x), log (log_e), log10
  – Trig functions: Math.sin(), etc.
    • sin, cos, tan, asin, acos, atan
      – Args are in radians, not degrees, (see toDegrees and toRadians)
  – Rounding and comparison: Math.round(), etc.
    • round/rint, floor, ceiling, abs, min, max
  – Random numbers: Math.random()
    • random (Math.random() returns from 0 inclusive to 1 exclusive).
    • See Random class for more control over randomization.
More Mathematical Routines

• **Special constants**
  – `Double.POSITIVE_INFINITY`
  – `Double.NEGATIVE_INFINITY`
  – `Double.NAN`
  – `Double.MAX_VALUE`
  – `Double.MIN_VALUE`

• **Unlimited precision libraries**
  – `BigInteger`, `BigDecimal`
    • Contain the basic operations, plus `BigInteger` has `isPrime`
Reading Simple Input

• **For simple testing, use standard input**
  - If you want strings, just use args[0], args[1], as before
    • To avoid errors, check args.length first
  - Convert if you want numbers. Two main options:
    • Use Scanner class
      - Note that you need import statement. See next slide!
    
    ```java
    Scanner inputScanner = new Scanner(System.in);
    int i = inputScanner.nextInt();
    double d = inputScanner.nextDouble();
    ```
    • Convert explicitly (Integer.parseInt, Double.parseDouble)
    
    ```java
    String seven = "7";
    int i = Integer.parseInt(seven);
    ```

• **In real applications, use a GUI**
  - Collect input with textfields, sliders, combo boxes, etc.
    • Convert to numeric types with Integer.parseInt, Double.parseDouble, etc.
import java.util.*;

public class RandomNums {
    public static void main(String[] args) {
        System.out.print("How many random nums? ");
        Scanner inputScanner = new Scanner(System.in);
        int n = inputScanner.nextInt();
        for(int i=0; i<n; i++) {
            System.out.println("Random num "+i+
                    " is "+Math.random());
        }
    }
}

How many random nums? 5
Random num 0 is 0.22686369670835704
Random num 1 is 0.0783768527137797
Random num 2 is 0.17918121951887145
Random num 3 is 0.3441924454634313
Random num 4 is 0.6131053203170818
Wrap-Up
Summary

• Basics
  – Loops, conditional statements, and array access is similar to C and C++
    • But new for loop: for(String s: someStrings) { … }
  – Indent your code for readability
  – String is a real class in Java
    • Use equals, not ==, to compare strings

• Allocate arrays in one step or in two steps
  – If two steps, loop down array and supply values

• Use Math.blah() for simple math operations

• Simple input from command window
  – Use command line for strings supplied at program startup
  – Use Scanner to read values after prompts or to turn simple input into numbers