
CS11 – Java

Fall 2009-2010

Lecture 1

Welcome!

- 8 Lectures
 - Slides posted on CS11 website
 - <http://www.cs.caltech.edu/courses/cs11>
 - 8 Lab Assignments
 - Made available on Tuesdays
 - Due one week later – Tuesday, 12 noon
 - CS Cluster Account
 - Software environment (Java 1.6)
 - Electronic submission
 - `~/cs11/java/lab1` etc.
-

Assignments and Grading

- Labs focus on lecture topics
 - ...and lectures cover tricky points in labs
 - Come to class! I give extra hints. 😊
 - Each lab will get a pass/fix, and feedback
 - If your code is broken, you will have to fix it.
 - If your code is sloppy, you will have to clean it up.
 - Must pass every assignment to pass CS11
 - Please turn in assignments on time
-

Course Texts

- No textbook is required
 - All necessary information is available online
 - Extensive Sun Java API documentation
 - Sun Java tutorials
 - If you *really* want a book:
The Java Programming Language, 4th ed.
Ken Arnold, James Gosling, David Holmes
-

A Brief History of Java

- Developed by Sun Microsystems, starting in late 1990
 - Intended for embedded-systems programming
 - Primary goal was improving on C++
 - Renamed to Java in 1994
 - Java 1.0 released in 1995
 - Numbering scheme changed with Java 5.0
 - (SDK/development version is still called 1.5)
 - Current version is Java 6 (aka 1.6)
-

A Brief History of Java (2)

- Language, and standard libraries, have expanded dramatically over the years
 - Java 6 released in late 2006 – introduced many new language features, new APIs
 - More language/library changes planned for Java 7
 - Java platform was made (mostly) open-source by Sun on May 2007
 - Facilitates more widespread adoption by software development teams
 - Allows Java platform to be ported to, and customized for, additional hardware platforms
-

Design Goals of Java Language

- Simple and familiar
 - Based on C/C++, but with many of the subtleties removed.
 - Object-oriented
 - Well suited to distributed systems
 - Architecture-neutral
 - Both source code and binaries are portable
 - Dynamic loading and binding
 - Minimizes recompilations, and facilitates modularity!
 - Secure
 - Class verification, code signing, permissions
 - Multithreaded
 - Language specifies platform-neutral threading support
-

How Java Does Its Thing

- Source code goes into `.java` files.
- One top-level class per file.
- Class' name dictates file name.

- Example: `HelloWorldApp.java`

```
// Display a message and then exit.  
public class HelloWorldApp {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!");  
    }  
}
```

How Java Does Its Thing (2)

- Java compiler takes `.java` files and compiles them into platform-independent `.class` files.
 - `javac HelloWorldApp.java`
 - ➔ produces `HelloWorldApp.class`
 - These class files contain “byte-codes” – instructions for the Java Virtual Machine (JVM).

- Byte-codes for our example:

```
public static void main(java.lang.String[])
0: getstatic      #2;      //Field java/lang/System.out
3: ldc           #3;      //String "Hello, world!"
5: invokevirtual #4;      //Method java/io/PrintStream.println
8: return
```

How Java Does Its Thing (3)

- Run the program with a Java Virtual Machine (JVM)
 - The JVM takes a class name, not the class' filename

```
> java HelloWorldApp
Hello, world!
```
 - The **java** program implements the JVM for a specific platform
 - Can run Java on any platform with a JVM implementation. (Windows, Linux, Solaris, MacOSX, ...)
 - Some JVMs improve performance by compiling Java byte-codes into native machine code
 - Called “just-in-time” compilation, or JIT for short
-

Java Comments

- Java comments are just like C++ comments

```
/*  
 * This method prints hello world.  
 */  
public static void main(String[] args) {  
    // This next part is tricky...  
    System.out.println("Hello, world!");    // phew!  
}
```

- Block comments can span multiple lines
 - Single-line comments extend to end of line
 - Nested `/* */` comments don't work!
 - Block comments end with *first* `*/` encountered
-

Javadoc Comments

- Java provides tools for generating API docs from source-code.
 - Tool is called `javadoc`
 - Generated documentation is called “javadocs”
 - Java API documentation is all auto-generated
- Javadoc processes special comments:

```
/**
 * Prints "Hello World!" then exits.
 */
public class HelloWorldApp {
    /** Main function. */
    public static void main(String[] args) {
        ...
    }
}
```

More Java Data Types

- Reference Types

- Refers to an object (not a primitive type)
- Can be `null` if the reference refers to nothing
- Examples: **String**, **Integer**

- In Java, arrays are also reference types

```
int[] numArray;           // preferred!  
int numArray[];         // also works
```

- More on arrays in a few weeks!
-

Java Literals

- Boolean is simply **true** or **false**.
 - Integer values are straightforward
 - `int i = 17;`
 - Long values use “L” suffix:
 - `long secondsInYear = 31556926L;`
 - Avoid lower-case “l” – looks like 1 in many fonts...
 - Default type of a decimal value is double!
 - `double pi = 3.14159265358979323;`
 - Float literal uses “F” suffix:
 - `float goldenRatio = 1.618f;`
 - In this case, either “F” or “f” is fine.
-

More Java Literals

- Character literals can be single-quoted characters, or numbers between 0 and 65535

```
char capA = 'A'; // preferred
```

```
char capA = 65; // harder to maintain
```

- String literals are double-quoted

```
String sandwichType = "pastrami";
```

- Special characters must be escaped:

```
String msg = "He said, \"Java is neat!\"";
```

- Most useful special characters:

`\t` = tab

`\r` = carriage return

`\n` = new line

`\\` = backslash

`\'` = single quote

`\"` = double quote

Java Variables

- Variable declarations include a *type* and *name*

```
int i;  
boolean err, done;  
String name;           // Reference to String object
```

- Names must start with a letter, and can include only letters and digits.
 - In Java, `_` and `$` are considered to be letters
 - Don't use `$` though; reserved for use in auto-generated code
-

Initial Values

- Can specify an initial value:

```
int i = 0;  
boolean err = false, done = true;  
String name = "Donnie";
```

- Local variables don't have default initial values

```
int i;  
i = i + 1;
```

➔ Compile-time error:

variable i might not have been initialized

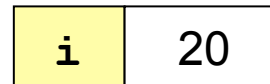
- This is an example of Java's focus on correctness
 - C or C++ would compile this code without errors
-

Primitive and Reference Variables

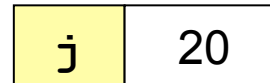
- Difference between primitive and reference types is where the value is actually stored

- Primitive variables:

```
int i = 20;
```



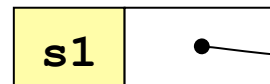
```
int j = i;
```



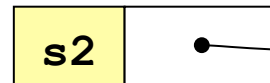
- Each variable stores its own value

- Reference variables:

```
String s1 = "Java!";
```

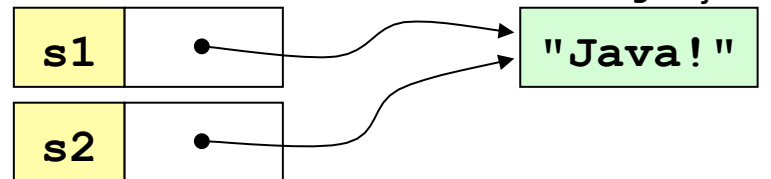


```
String s2 = s1;
```



String object

"Java!"



- Value of reference variables is stored in main memory
- Reference variables can refer to the same object

Java Operators

- Same set of operators as C and C++

- Operators have *precedence*

- Example: * has higher precedence than +

```
int ans = 3 + 5 * 2;    // Result = 13, not 16
```

- Override precedence rules with parentheses

```
int ans = (3 + 5) * 2;  // Result = 16, not 13
```

- Also use parentheses where precedence rules aren't obvious
-

Arithmetic Operators

- Same as C/C++: `+` `-` `*` `/` `%`
 - `%` is the remainder operator
- Operate on numeric types: `int`, `float`, etc.
- Compound assignment operators:

```
totalScore += pointsEarned;
```

```
recipeAmount *= scale;
```

- Preincrement and postincrement operators:

```
int i = 5;
```

```
int j = ++i;           // j = 6, i = 6
```

```
int k = i++;          // k = 6, i = 7
```

Comparison Operators

- Compare two values, produce a **boolean** result
 - Standard set: < > <= >= == !=
 - Note that **boolean** and **int** are *not* equivalent!
 - ❑ **Wrong:** `int res = (x < 64); // Won't compile!`
 - ❑ **Correct:** `boolean res = (x < 64);`
 - In Java, no type can be cast to **boolean**
 - **boolean** also cannot be cast to any other type
-

Logical Boolean Operators

- Again, same as C/C++: `&&` `||` `!`
 - Logical AND, logical OR, and logical NOT.
 - These operators *require* **boolean** values, and *produce* **boolean** values.
 - Lazy evaluation:
 - For example: `name != null && name.equals("Donnie")`
 - `name.equals(...)` only evaluated if `name != null`
 - Conversely: `name == null || !name.equals("Donnie")`
 - Precedence order: `!` `&&` `||`
-

String Operators

- String concatenation also uses + operator

```
public static void main(String[] args) {  
    String name = "Donnie";  
    System.out.println("Hello " + name);  
}
```

- At least one operand must be a String for + to do string-concatenation.

```
int i = 5;  
int j = 4;  
System.out.println("i = " + i); // Prints "i = 5"  
System.out.println(i + j); // Prints "9"  
System.out.println("i + j = " + i + j); // "i + j = 54"  
System.out.println(i + j + " = i + j"); // "9 = i + j"
```

- + operator is evaluated left-to-right
-

Flow Control in Java

- If statements are nearly identical to C and C++

```
if (cond)
    statement;
else if (cond)
    statement;
else
    statement;
```

- Difference: *cond* must produce **boolean** value!
- Blocks of statements are enclosed with curly-braces

```
{ }, just like in C/C++
if (cond) {
    statement1;
    statement2;
}
```

Looping in Java

- While-loops:

```
while (condition)  
    statement;
```

```
while (condition) {  
    statement1;  
    ...  
}
```

- Tests condition *before* each iteration.

- Iterates zero or more times.

- Example: Sum the numbers 1..10

```
int i = 1;  
while (i <= 10) {  
    sum += i;  
    i++;  
}
```

- Iterates 10 times. Sum is 55 at end of loop.
-

Looping in Java (2)

- Do-loops:

```
do
    statement;
while (condition);
```

```
do {
    statement1;
    ...
} while (condition);
```

- Tests condition *after* each iteration.

- Iterates one or more times.

- Example:

```
// Convert positive integer into string version
do {
    strBuf.append(num % 10); // get lowest digit
    num /= 10; // then remove it
} while (num != 0);
```

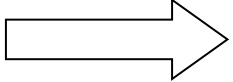
For-Loops

- Combine the basic operations into one syntax:

```
for (init; condition; update) statement;  
for (init; condition; update) {  
    statement1;  
    ...  
}
```

- Equivalent to **while** loops, but more compact.
- We can rewrite our previous while loop:

```
int i = 1;  
while (i <= 10) {  
    sum += i;  
    i++;  
}
```



```
Initialization      Condition      Update  
    ↓                ↓                ↓  
for (i = 1; i <= 10; i++)  
    sum += i;  
    ↑  
Statement
```

More For-Loops

- Can specify multiple initial values:

```
int i, sum;
for (i = 1, sum = 0; i <= 10; i++)
    sum += i;
```

- Can declare loop variables in for-loop:

```
int sum = 0;
for (int i = 1; i <= 10; i++)
    sum += i;
```

- In this example, **i** is only visible within the for-loop
 - The *scope* of **i** is within the for-loop.
-

Even More For-Loops

- Can specify multiple update operations:

```
int sum = 0;
for (int i = 1; i <= 10; sum += i, i++) /*nothing*/;
```

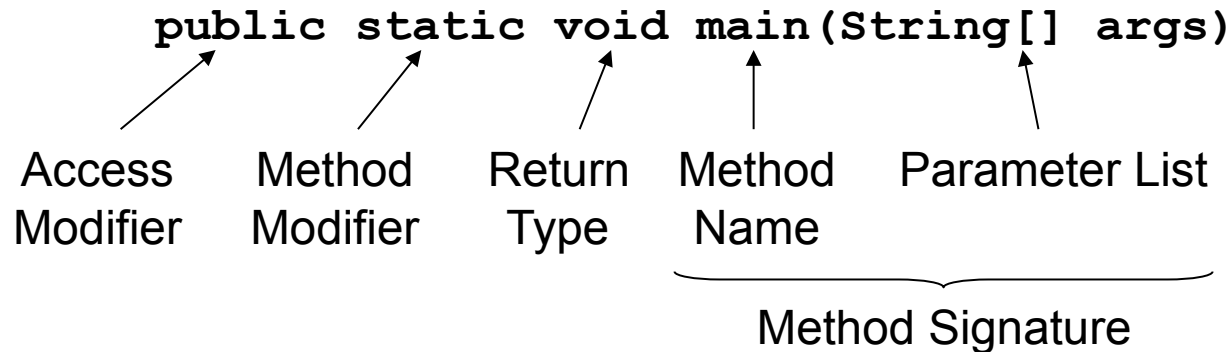
- Document that the for-loop doesn't need a body!

- Even *more* compact:

```
int sum = 0;
for (int i = 1; i <= 10; sum += i++) /* nothing */ ;
```

- Difficult to maintain! Best to be avoided.
-

Java Methods – A Brief Overview



- Methods return a value of the specified type.
 - Or they return *no value*, indicated by `void` keyword.
 - Methods can accept any number of arguments.
 - “No arguments” is indicated with empty parens `()`, not `void`.
 - A method’s signature includes its name and its parameter-list.

 - Modifiers will be covered in future classes.
-

Printing in Java

- `System.out.println("Hello!");`

- Many flavors:

 - `System.out.println(String x)`

 - `System.out.println(boolean x)`

 - `System.out.println(char x)`

 - `System.out.println(float x)`

 - `System.out.println(int x)`

 - `System.out.println(Object x)`

 - `System.out.println()`

 - and a few more...

- These are overloaded methods.

 - Same name, but different signature.

More Printing Options

- `System.out` is the standard output stream
 - `System.err` is the standard error stream
 - Use this to report errors when bad things happen.
 - `System.in` is the standard input stream
 - We will use this next week.
 - `System.out.println(...)` goes to next line
 - Use `System.out.print(...)` to stay on same line
-

A Note About Class Names

- Java classes can be grouped into packages
 - This is optional, but typically very helpful!
 - Packages form a hierarchy
 - **package1.package2.ClassName**
 - Package names are typically all lower-case
 - Naming rules are same as variable names.
 - Example: `java.awt.event.MouseEvent`
 - More details on this later!
-

Java Coding Style!

- Capitalization is *very important* in Java coding style
 - Fields and methods should follow **camelCase** naming convention
 - Classes and interfaces should follow **UpperCamelCase** naming convention
 - Package names should be *all lowercase*
 - Java has a number of industry-wide conventions like this.
 - Definitely want to learn them and follow them...
 - You must follow them in CS11 Java.
-

The Java API Documentation

- Complete API docs for the entire Java platform
 - Extremely useful, once you learn how to use it!
 - Auto-generated from Java library source-code
 - Lists all classes and interfaces
 - How to use them
 - What features they provide
 - Their relationships with each other
 - <http://java.sun.com/javase/6/docs/api/>
 - So useful, you might even want a local copy!
-

Other Useful Java Documentation

- The Java Tutorial
 - Different “trails” cover different topics
 - Very helpful resource for learning new features!
 - Java Development Kit (JDK) Documentation
 - Feature-changes and new features
 - Tool documentation
 - The Java Language Specification
 - The Java VM Specification
-

This Week's Homework

- Create your first Java program.
 - Use some of the looping constructs we discussed.
 - Learn how to compile and run your program.

 - Read about object-oriented programming in Java.

 - Learn to navigate the Java API Documentation
-

Next Time!

- Java classes and objects
 - Read about this in your homework
 - Fun with math.
(Or, why computers can't do basic arithmetic.)
-