From C++ to Java

- **Java history**: Oak, toaster-ovens, internet language, panacea
- **What it is**
  - O-O language, not a hybrid (cf. C++)
  - compiled to byte-code, executed on JVM
  - byte-code is “highly-portable”, write once run “anywhere”
  - simple, object-oriented, portable, interpreted, robust, secure,
    architecture-neutral, distributed, dynamic, multithreaded,
    high performance

- **create both applets and applications**
  - current version 1.1 (1.1.5, 1.1.4 on acpub), 1.2b2 out now,
    also on acpub
  - browsers support 1.0.X, some support 1.1
Java Fundamentals

- **primitive types**: int, double, char, boolean (and some others that differ in size, e.g., float, long)
  - size of int is guaranteed to be 32 bits (unlike, say, C++)
  - these primitive types are different than all other types, do NOT belong to inheritance hierarchy, are NOT allocated using new, can be compared using ==, are copied, ...
- **All other types descend from base type `Object`, must be allocated using new**
  - no delete, garbage collection is automatic
  - all parameters are passed by value, no reference params
  - everything is a pointer (really a reference) --- can change value of parameter via a method call, not via assignment
  - no address operator, “safe pointers”
C++ and Java confusion

- `==` only works for primitive types
  
  ```java
  String s = "hello"; String t = "hello"; if (s == t) ...  
  ```
  ➤ `equal()` to check for semantic equality, `==` for pointers

- Assignment is shallow copy, no new values defined
  
  ```java
  Foo f = new Foo(123); Foo g = f; g.change();  
  ```
  ➤ What happens to `f`?
  ➤ `use clone()` function, from interface `Cloneable`

- No semi-colons after class declarations
- Repeat `public/private` each time needed, default is `package`
Java stuff: Strings and Arrays

● **String**
  ➤ immutable, once set cannot be changed (but make a new one), see also `StringBuffer`
  ➤ concatenation using +, this is the only non-arithmetic use of +, in Java no user-defined overloaded operators (+= also works, what about immutable?) any class can be concatenated if it implements `toString()`

● **array and Vector**
  ➤ array is typed, non-growable, random-access collection
  ➤ vector is non-typed, growable, random-access collection
    • casting out of function calls (e.g., container get) is required, casting is checked at runtime, therefore safe
      ```java
      v.setElementAt("hello", 1);
      String s = (String) v.getElementById(1);
      ```
Compiling/Executing Java Programs

- **class** `Foo` must be stored in `Foo.java`
  - file name corresponds to class name
  - directory hierarchy corresponds to package hierarchy
    - `java.lang.String` is in package `java.lang`, must be stored in path `/xxxxx/*/java/lang`
  - `package` is set of related classes
    - CLASSPATH specifies where packages are looked for

- **compile, run, document**
  - `javac`, compiler: `javac -deprecation Foo.java`
  - `java`, runtime: `java Foo`
  - `javadoc`: `javadoc -author *.java`

- **import java.lang.***
  - different from `#include` in C++, namespace device
Inheritance, Interfaces

● All classes in Java *extend* the class Object
  ➤ explicit extension/inheritance is possible, but only single inheritance
  ➤ possible to *implement* multiple *interfaces*

● An interface is like an abstract base class, all methods/member functions *must* be implemented
  ➤ example: Enumeration is an iterator interface
    ```java
    boolean hasMoreElements()
    Object nextElement()
    ```

● Possible to declare an object of type Enumeration, but cannot use `new Enumeration` (but see anonymous class exception)
  ➤ class Foo extends Widget implements Enumeration
Public, private, protected, package

- similar to use in C++
  - public methods/classes callable from client code
  - private methods/instance variables not accessible NO FRIENDs
  - protected limits access to subclasses
  - no designation is package access (this approximates friend in C++, but it’s both better and worse)

- Package is a module of related classes
  - package classes can access all package functions/data
  - can be used like friend functions --- belong to a package
  - directory hierarchy mimics package designation, CLASSPATH must be set properly
Java I/O and other non-pretty stuff

- I/O is not pretty
  - Reader base class: Reader supports reading characters only, no formatted input
  - use a BufferedReader constructed from another Reader
  - formatted I/O: use Integer, Double, etc., see ConsoleInput

- Integer: an int wrapped in a class
  - static Integer valueOf(String) -- returns an Integer
  - int intValue() -- corresponding int
  - static int parseInt(String) -- returns an int

- Double:
  - no parseInt(), but valueOf() and doubleValue()