Java on one slide

- All objects allocated on heap, via new, garbage collected
  - Primitive types like int, double, boolean exempt
    - Everything else subclasses Object
  - All variables (non-primitive) are pointers aka references
    - Can we compare pointers for equality? Is this a problem?
- No free functions, everything in a class, inheritance by default
  - Functions and classes can be *final*, not inheritable from
  - Static functions like Math.sqrt are like free functions
  - Local variables *must* be assigned to, instance variables all initialized by default to 0, null
- Containers contain only non-primitive types
  - Conversion between int and Integer can be ugly
  - Use ArrayList and HashMap instead of Vector, Hashtable

Java on another slide

- Public class Foo must be in a file Foo.java
  - Compiled into Java bytecodes, stored in Foo.class
    - Bytecodes executed inside a JVM: Java Virtual Machine
    - JVM is architecture specific, often relies on native/C code
  - Helper/non-public classes can be in same file
    - Keep related/cohesive concepts together
    - Don't go overboard
- Execution starts with a static main function
  - Any class can have such a function, class invoked specifically via `java Foo` (runs Foo.main)
- The environment is important and essential
  - You need to understand CLASSPATH to leverage Java

From STL to Java

- In STL an iterator is a concept, there are refinements
  - Input, output, forward, bidirectional, random access
    - A forward iterator is an input iterator and an output iterator
    - The iterator may be immutable (or const)---read only
  - Refinements not implemented by inheritance, but by design, contract, and subsequently implementation
    - What happens if you try to implement an STL iterator?
- In Java *Iterator* is an interface (like a base class), similar to Tapestry iterators
  - Collection(s) are required to have iterators, these are used in some operations like max, min, construct vector, …
  - Related to STL as algorithm glue, but very different

WordCount.java, print strings, line #'s

```
public void print()
{
 Iterator allKeys = myMap.keySet().iterator(); // words

 while (allKeys.hasNext()) {
     String key = (String) allKeys.next();
     System.out.print(key + "\t");
     Iterator lines = ((Set) myMap.get(key)).iterator();
     while (lines.hasNext()) {
         System.out.print(lines.next() + " ");
     }
     System.out.println();
 }
}
```

- Differences between Java and Tapestry in practice?
  - Must store current element since `next()` does two things
  - Must cast since Collections store Objects

Java inheritance

- By default every class can be a base/parent class, every method is polymorphic. To inherit use *extends* keyword
  - Can change with final keyword (similar to const, but not)
  - A class can extend only one base class (but see interfaces)
  - Public, protected, private similar to C++, what's not?
- A class can be an abstract class, *public abstract class Foo*
  - Can't instantiate (no new Foo()), but can extend
  - A method can be abstract, like pure virtual in C++
- A class *implements* any number of *interfaces*
  - Like ABC, but function prototypes only, no state
  - Subclass must implement all methods of interface

Modules and Packages

- Java code/modules organized into packages
  - C++ has namespaces, required and now used
  - Java uses packages: corresponds to directory hierarchy
  - We're using the default package (no name) later we'll use packages
  - java.util, java.lang, java.io, … are all packages
- The import statement at the beginning of a program doesn't work like #include, it tells the Java compiler where to look to resolve names
  - Differences from #include/pre-processor?