From C++ to Java

- Java history: Oak, toaster-ovens, internet language, panacea
  - Not really a standard language like C++
  - Arguably proprietary (and arguably not)
  - Precursor to C# ?

- What it is
  - O-O language, not a hybrid (like 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

Java on one slide

- All objects allocated on heap, via new, garbage collected
  - All objects subclass Object
  - All object variables are pointers aka references
    - Can we compare pointers for equality? Is this a problem?
  - Primitive types like int, double, boolean are not objects
- 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

Using Java

- **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

- **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

Java References in Practice

- Parameters passed by value, but Objects stored as references
  - Good news
    - No copy of object is made
    - Can modify object's state via method calls
  - Bad news
    - Does no good to change what object parameter refers to
    - Primitive types are always copied
  - Most of the time, exactly what you want
  - But consider, how to write swap

- Strings are immutable
  - No public methods modify state, again, usually what you want
  - Use StringBuffer when you need to change

```
String s = "hello";
s = s + " world "
s = (new StringBuffer()).append(s).append("hello").toString();
```

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)
  - Class can extend only one baseclass (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, not often used
  - Java uses packages: corresponds to directory hierarchy
  - We are using the default package (no name) later we will use packages
  - java.util, java.lang, java.io, … are all packages

- The import statement at the beginning of a program does not work like #include, it tells the Java compiler where to look to resolve names
  - Differences from #include/pre-processor?

Access modifiers aka public/private

- Related classes should be grouped together, may even share access to otherwise private methods/instance variables
  - A package corresponds to a directory just as class corresponds to file
  - Each method/instance variable must be declared as
    - public, accessible to all client code (other packages)
    - private, accessible only to class itself
    - protected, accessible to subclasses *and package classes*
    - (nothing), accessible to classes in same package
- One public class per file
  - Can have helper classes in the file, these will have package scope
  - Can have nested classes, very useful, also in C++

Factoring Out Common Code

- Common constructor code, one constructor calls another
  - `this(...)`
    - First code in a constructor, contrast to solution in C++
- Call super/parent class
  - `super(...)`
    - Calls inherited constructor, default is automatically called
    - Contrast to C++, must name class
    - Must be first line, cannot refer to `this` (not set up yet)
  - `super.method(...)`
    - Calls inherited method of the same name