Reading:
- Mitchell, Ch. 12

Objectives
1. Java design goals
   (a) be able to list the major design goals of Java
   (b) be able to describe some of the ways these goals impacted the design of the Java language and implementation
   (c) be able to compare and contrast Java’s design goals with those of C++ and SmallTalk and be able to give examples of how those differing goals are reflected in the design.
2. be able to compare and contrast the Java language mechanisms for inheritance and subtyping with inheritance and subtyping in C++
3. be able to explain the major cases when dynamic lookup is used (instance method invocation on an object reference typed by a class or interface) and not used (static method invocation) in Java.

Outline

<table>
<thead>
<tr>
<th></th>
<th>Portability</th>
<th>Safety</th>
<th>Simplicity</th>
<th>Efficiency</th>
<th>Extensibility</th>
<th>Concurrency</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM Platform</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Linking</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type Safety</td>
<td>+</td>
<td>+</td>
<td>+/-</td>
<td>+/-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mostly objects</td>
<td></td>
<td></td>
<td>+</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ref. Var.</td>
<td>+</td>
<td></td>
<td>+</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Garbage Collect</td>
<td></td>
<td></td>
<td>+</td>
<td>+</td>
<td>+/-</td>
<td></td>
</tr>
<tr>
<td>Memory Model</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Threads</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>

(a) less focus on efficiency, more focus on safety/simplicity (but less on expressivity?) than C++

(b) more on efficiency/safety, less on flexibility/extensibility than SmallTalk

(c) What about features left out? multiple inheritance, operator overloading, arbitrary pointer operations
(d) what about more recent features? nested classes, autoboxing, generics, etc.

2. abstraction
   (a) private, public, protected, default (package protected) access modifiers
   (b) nested classes
   (c) packages (sealed jar files)
   (d) class loaders

3. subtyping and inheritance
   (a) entities
      i. ‘interface’ = type and interface only; not instantiatable
      ii. concrete ‘class’ = type, interface, and implementation; instantiatable
      iii. ‘abstract class’ = type, interface, and implementation; not instantiatable
   (b) ‘implements:’ subtypes ‘interface’
   (c) ‘extends:’ inherits implementation from class and subtypes interface of class

4. method invocation and dynamic lookup
   (a) invokevirtual (normal call to instance method; dynamic lookup)
      i. method name in a call might get resolved:
         A. when class containing caller is loaded (eager)
         B. first invocation (lazy with patching)
         C. each invocation (lazy)
      ii. single inheritance
         A. fixed ordering based on prefixing superclasses
         B. lookup table similar to C++ virtual functions
         C. mtable
   (b) invokeinterface (receiving object has interface type; dynamic lookup)
      i. itable for each interface / implementing class pair
      ii. fixed offset in itable
      iii. list of itables in object representation
      iv. linear search of itable list to find correct interface
      v. more flexible, less efficient than invokevirtual
   (c) invokevirtual (invocation of a static method of a class)
i. no dynamic lookup, known callee
ii. more efficient, less flexible than invokevirtual

5. Java Dynamic Class Loading Process
   
   (a) requested by name
      i. implicitly, when some class-being-loaded refers to it
      ii. implicitly, when some field/method of it is used
      iii. explicitly, by a call to ClassLoader.loadClass(...)

   (b) class loader finds and "loads" the class
      i. maps fully-qualified class names to class files
      ii. built-in or application-defined (e.g., Eclipse, Tomcat)
      iii. class loader hierarchy
      iv. submits the class file to VM with call to ClassLoader.defineClass(...)

   (c) linking
      i. verify (type check bytecode)
      ii. prepare (initialize fields to null/0, create method tables, etc.)
      iii. eager resolution
         A. recursively load classes referred to by name in the class being loaded
         B. superclasses, interfaces, parameter types, types used in method body
         C. may be done during linking or done lazily as needed later

   (d) initialization
      i. static variable initialization
      ii. static initializers

   (e) newly loaded class can be used (instances can be created, static fields can be accessed, static methods called etc.)

vocabulary

reference variable, primitive type, virtual machine, garbage collection, access modifier, class, abstract class, interface [keyword/Java entity], single inheritance, mtable, itable, invokevirtual, invokeinterface, invokesstatic, class loader