CS2136: Paradigms of Computation
Class 17:
Java:
Containers
Enumerations & Iterators

Copyright 2001, 2002, 2003, Michael J. Ciaraldi and David Finkel
**But First...**

- Remember that objects are active.
- Generally...
  - You don’t do things to them.
  - You ask them to do things to themselves.
But Second...

- A little more on polymorphism, overloading, and overriding.

Remember:

- Overloading means same operation name.
  - Same or different classes.

- Overriding means same operation signature.
  - Different classes.
  - Polymorphism only applies to overriding.
PolyOver2.java

Art
- method1(aa : Art) : void
- method1(dd : Drawing) : void
- method2(aa : Art) : void
- method2(dd : Drawing) : void
- method3(aa : Art) : void
- method4(dd : Drawing) : void

Drawing
- method1(aa : Art) : void
- method1(dd : Drawing) : void
- method3(dd : Drawing) : void
- method4(aa : Art) : void

Cartoon
Containers
Containers

z Objects which hold objects
  y What they hold are of class `Object`.
  y Not any specific subclass.

z Categories
  y Collection
    x Set
    x List
  y Map
Containers

In Java 1.3, Collections and Maps are Interfaces.

- Need to actually use concrete classes which implement these interfaces.
- Or create your own.

Containers were completely revamped in Java 2 (as implemented in JDK 1.2).

- Old ways, e.g. Vector, are de-emphasized.
Collections vs. Maps

- Collections hold individual elements.
- Maps hold pairs.
  - Key and value.
Collections

z Hold individual elements

z Types (subinterfaces):
  y A **Set** cannot have duplicate elements
  y A **List** holds elements in a particular sequence.

z Prints in square brackets, separated by commas.
Maps

A **Map** holds a group of **pairs** of objects.
Each pair is a key and a value.
a.k.a. associative array.
Acts like a simple database.
No duplicate keys.
Prints in curly braces, with pairs shown as key=value.
Collections

Methods common to all:
- boolean add(Object o)
  - Returns true if collection has changed.
- boolean remove(Object o)
  - Returns true if collection has changed.
- void clear()
List

- Holds elements in a particular sequence.
- Automatically expands
- You should use one of the concrete classes which implement this interface:
  - ArrayList
  - LinkedList
ArrayList

Creating:
  - new ArrayList()
  - new ArrayList(int initialCapacity)

Measuring:
  - int size()
ArrayList

z Storing:
  y boolean add(Object o)
  y boolean add(int index, Object element)
  y Object set(int index, Object element)

z Retrieving:
  y Object get(int index)
  y Object remove(int index)
ArrayList

z Testing:
  y boolean isEmpty()
  y boolean contains(Object elem)

z Finding (failure = -1):
  y int indexOf(Object elem)
  y int lastIndexOf(Object elem)
Set

- Holds elements in no particular sequence.
- No duplicates allowed.
- Automatically expands.
- You should use one of the concrete classes which implement this interface:
  - HashSet
  - TreeSet
Map

- Stores values and keys.
- Duplicate keys not allowed.
- Can be tricky to set up.
- Concrete implementations:
  - Attributes
  - HashMap
  - Hashtable
Map

z Storing:
  y Object put(Object key, Object value)
  x Note: Both must be objects, not primitives.

z Retrieving:
  y Object get(Object key)
  y Object remove(Object key)
Map

z Testing:
  y boolean containsKey(Object key)
  y boolean containsValue(Object value)
Problems With Map

- Matching keys
- Overriding equals() and hashCode().
HashMap

- Implements a hash table.
- Efficient, constant-time access.
- Two functions must be properly implemented for the class used as the key:
  - boolean equals()
  - int hashCode()
Overriding equals()

The default method compares object addresses: no good because you want to compare contents.

Return true if contents are equal, by whatever criteria are important in your case.
Overriding `hashCode()`

- The `hashCode()` function must generate an int derived from the key.
- The default method uses object address: no good.
- Two objects with the same contents must return the same hash. Vice versa not required.
Enumerations & Iterators: A Brief Look
Enumerations & Iterators

z Objects used to step through a container.
  y Available for some standard container classes.
  y Work properly even if the container changes.
  y Order might or might not be significant.

z Java has two variations:
  y Enumeration (old: since JDK 1.0)
  y Iterator (new: since JDK 1.2)
Enumerations

z To get an Enumeration e for container v:
  y Enumeration e = v.elements();
  y This is initialized at the start of the list.

z To get the first and subsequent elements:
  y someObject = e.nextElement()

z To test if all have been accessed:
  y e.hasMoreElements()
Enumeration Example

```java
for (Enumeration e = v.elements();
     e.hasMoreElements(); ) {
    System.out.println(
        e.nextElement());
}
```
Iterators

z To get an Iterator i for container v:
y Iterator i = v.iterator();
y This is initialized at the start of the list.
z To get the first and subsequent elements:
y someObject = i.next()
z To test if all have been accessed:
y i.hasNext()
Iterator Example

```java
for (Iterator i = v.iterator();
     i.hasNext(); ) {
    System.out.println(i.next());
}
```
Next Time

- Exceptions
- Numbers