Collections and Maps in Java

March 19–20, 2012
CS 60: Principles of Computer Science

Assignment 7 due Monday, March 26: Unicalc, Part 1
## Overview

<table>
<thead>
<tr>
<th>Abstract Types</th>
<th>Specific Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>List&lt;T&gt;</td>
<td>ArrayList&lt;T&gt;, LinkedList&lt;T&gt;, ...</td>
</tr>
<tr>
<td>Set&lt;T&gt;</td>
<td>HashSet&lt;T&gt;, TreeSet&lt;T&gt;, ...</td>
</tr>
<tr>
<td>Map&lt;K,V&gt;</td>
<td>HashMap&lt;K,V&gt;, TreeMap&lt;K,V&gt;, ...</td>
</tr>
</tbody>
</table>

### Notes:

1. If you don’t care whether it’s a list or a set, you can be even more abstract and say `Collection<T>`
2. These must be imported from the `java.util` package.
Which are OK?

1. Set<String> myStrings = new HashSet<String>(); ✓
2. Set<String> myStrings = new TreeSet<String>(); ✓
3. Set<String> myStrings = new Set<String>(); ✗
4. HashSet<String> myStrings = new HashSet<String>(); ✓
5. HashSet<String> myStrings = new TreeSet<String>(); ✗
6. HashSet<String> myStrings = new Set<String>(); ✗
What can I do with a set?

✓ .add(…) a new element (imperative!)
✓ .addAll(…) elements from another collection (imperative!)
✓ .remove(…) an element if it’s present (imperative!)
✓ Check if it .contains(…) a particular element
✓ Check if it .isEmpty()
✓ Ask for its .size()
✓ Loop over the contents of the set.

See

http://docs.oracle.com/javase/6/docs/api/java/util/Set.html

for other operations
Looping through contents of a set, list, array (!), ... 

The “old” way: get an `Iterator` object from the collection:

✓ `.hasNext()` checks if there are more elements
✓ `.next()` returns the “next” element. (Depending on the collection, you may or may not get elements in order.)

```java
// Print each string in myStrings on its own line

Iterator<String> it = myStrings.iterator();

while (it.hasNext())
{
    System.out.println( it.next() );
}
```

What if we wanted to print each string twice?
The “new” (Java 5) way: specialized for (“for each”) loops:
(Under the hood, Java is creating the iterators for you!)

```java
// Print each string in myStrings on its own line
for (String s : myStrings) {
    System.out.println( s );
}
```

What if we wanted to print each string twice?
Cloning Collections

An easy way to make a copy of a collection (or map) is to use it as a constructor argument.

```java
Set<Integer> s1 = new HashSet<Integer>();
s1.add(3); // old-style: s1.add(new Integer(3));
s1.add(4); // old-style: s1.add(new Integer(4));

Set<Integer> s2 = new HashSet<Integer>(s1);
s2.add(3);
System.out.println(s1.size());
System.out.println(s2.size());

Set<Integer> s3 = new TreeSet<Integer>(s1); // !!
List<Integer> s4 = new ArrayList<Integer>(s1); // !!
Some Handy Functions

1. **Arrays.asList** creates a List<T> out of its arguments (which must all have the same type T):
   ```java
   List<String> names = Arrays.asList("Moe", "Larry", "Curly");
   List<Integer> ints = Arrays.asList(5,1,3,2,4);
   ```

2. **Collections.sort(...)** can be used to (imperatively!) sort any List whose elements are comparable:
   ```java
   Collections.sort(names);
   Collections.sort(ints);
   ```

Note: **Arrays and Collections** must be imported from `java.util`. 
What can I do with a map?

✓ `.put(...,...)` in a new key-value association (imperative!)
✓ Check if it `.containsKey(...)` a particular key
✓ `.get(...)` the value associated with a given key
✓ Ask for its `.size()`
✓ Ask for a `.keySet()` set containing all the keys in the map.
✓ Ask for a `.values()` set containing all the values in the map.

See

http://docs.oracle.com/javase/6/docs/api/java/util/Map.html

for other operations
Exercise: How can we print all key-value pairs in a given Map<String, Integer> m?

```java
for ( String k : m.keySet() )
    System.out.println( k + " -> " + m.get(k).toString() );
```
Java Programming Rule

Every object inherits an `.equals(…)` method.
✓ By default, it checks whether two objects are the same object in memory.
✓ This isn’t always way you want.

Every object inherits a `.hashCode()` method returning an `int`.
✓ Converts the object to an int
✓ By default, something simple like the object’s address

If you override `.equals(…), you must override `.hashCode()` as well!
✓ Equal objects must return the same hash code.
class Point extends Object {
    private double x;
    private double y;
    ...
    public boolean equals(Object other) {
        if (other instanceof Point) {
            Point otherp = (Point)other;
            return (this.x == otherp.x &&
                    this.y == otherp.y);
        } else {
            return false;
        }
    }
}

Set<Point> hs = new HashSet<Point>;

hs.add(new Point(3,4));

System.out.println(hs.contains(new Point(3,4)));
Writing computing **hashCode**s

1. *Never just give every object the same hash code!*
   
   ```java
   return 42;
   ```

2. **Convert each field to a suitable int and combine these numbers, e.g.,**
   
   ```java
   // Simple method of Joshua Bloch, "Effective Java"
   int result = 0;
   result = ...hash first field...;
   result = 37 * result + ...hash second field...;
   result = 37 * result + ...hash third field...;
   result = 37 * result + ...hash fourth field...;
   return result;
   ```

There is an art to constructing good **hashCode**s. *CS 70* discusses this more.
Constructing Individual Hash Codes

If you need a number for a field f:

✓ If f is an object, try f.hashCode()
✓ If f is an int, try f
✓ If f is a double, try new Double(f).hashCode()
✓ ...
