Java’s Map Classes

Java has two general purpose classes that implement maps:

TreeMap<K, V>

HashMap<K, V>

For example, to make a map with keys that are strings and values that ar
Integers we might have

```
TreeMap<String, Integer> map1 = new TreeMap<String, Integer>();
```

or

```
HashMap<String, Integer> map2 = new HashMap<String, Integer>();
```

The most important methods of the two classes have the same names:

- V put(K key, V value) This adds the (key, value) pair to the map. If the key was already a key of the map this returns the prior value that was associated with it; otherwise it returns null. You can usually just ignore the return value.
- V get(K key) This returns the value associated with the given key, or null if the key is not associated with a value. Note that Python's dictionaries crash if you try to look up the value associated with something that isn't a key; Java's maps just return null.
- Boolean containsKey(K key)
- Set<K> keySet( ) This returns a set of all of the keys in the map.

For example, if I wanted to print all of the information in map2, which is a TreeMap<String, Integer>, I could say

```
for (String k: map2.keySet() )
        System.out.printf(  “(%s, %d)\n”, k, map.get(k) );
```

If I needed to sort the keys before printing I could put them in a list and sort it:

```
ArrayList<String> L = new ArrayList<String>();
L.addAll(map2.keySet() );
Collections.sort(L);
for (String k: L )
        System.out.printf(  “(%s, %d)\n”, k, map.get(k) );
```

There is also a 2-argument constructor for HashMaps:

HashMap(int initialCapacity, float loadfactor)

The default loadfactor if you don't specify one is 0.75

If size/capacity ever becomes larger than the loadfactor the map is automatically rehashed to a table twice as large. This is time-consuming, so try to make your initial capacity large enough to hold all of your data.