Sets and Maps

David Caggiano, Andy Miller, and Bryan Rose

What is a Set?

The Set interface is a part of Java's Collections framework. The most important feature of Sets is that they cannot contain duplicate elements. The add(E element) method of a Set will return **true** if the element was successfully added to the ~~set~~. If the Set already contained the element, it will return **false**.

There are 3 default implementations of the Set interface in Java:

- The **HashSet** implementation stores its elements in a hash table. It is the fastest Set to iterate through and to complete operations on; however, a HashSet has no "logical" ordering of its elements. It is unknown what order a HashSet will iterate through its elements. A HashSet's add, remove, and contains method are all O(1).
- The **LinkedHashSet** is the second fastest implementation. The trade-off to being slower than a regular HashSet is that the elements are linked in the order they are added to the Set. A LinkedHashSet will iterate through its elements in the order they are added.
- The **TreeSet** implementation is the slowest of the three. It is the slowest because the elements are kept in their "natural ordering." In other words, the elements in a TreeSet are automatically sorted from "least" to "greatest" (using the compareTo() method). For example, a list of Integers will be ordered numerically and a Set of strings will order the Set alphabetically. Traversing through a TreeSet is much slower than through a HashSet, but the iteration will follow a logical order.

An enhanced for loop or an iterator must be used to iterate through a Set (because the elements are not indexed). Common methods used for a Set include: add(E element), remove(E element), contains(E element), and size().

What is a Map?

The Map interface is also a part of the Collections framework. A Map will link a Key to a Value. To better understand Maps, think of an array as mapping an index to a value. If in the String array arr, arr[0] = "Bob", then in a Map the Key = 0 and the Value = "Bob". The important thing about Maps is that the Key does not have to be an integer. A Map can link any Object with any other Object. For example, the Key could be "Bob" and the Value could be "fat".

Example of instantiating a Map with String keys and String values:

```
Map<String, String> stringMap = new HashMap<String, String>();
```

The order of the types in the brackets is <Key, Value>. To add a <Key, Value> pair to a map, use the put(Key, Value) method. A Map contains its keys inside a Set. This means there can only be one of a certain key in a map. However, multiple keys can map to the same value.

There are three default implementation of the Map interface in Java, and they correspond to the three default Set implementations:

- A **HashMap** has a key set that is a HashSet.
- A **LinkedHashMap** has a key set that is a LinkedHashSet.
- A **TreeMap** has a key set that is a TreeSet. Relative to each other, these have the same speed and functionality trade-offs that the Set implementations have.

There are three ways to iterate through the elements of a Map: iterate through the key Set, iterate through the Collection of values, or iterate through a Set of <Key, Value> pairs. The Set of keys is obtained using the keySet() method, the Collection of values is obtained using values(), and the <Key, Value> pairs are obtained using entrySet(). Common methods used for a Set include: put(Key k, Value v), get(Key k), remove(Key k), containsKey(Key k), containsValue(Value v), size(), keySet(), values(), entrySet().

Sources:

http://docs.oracle.com/javase/tutorial/collections/interfaces/set.html
http://docs.oracle.com/javase/tutorial/collections/interfaces/map.html