Bill D'Attilio
Eric Taylor

Sets and Maps

First off there is the Set<E> interface. Basically a set is a collection with no duplicates, so no objects in the set can be equal. Another major characteristic of a set is that the order of its elements does not matter. The fundamental operations of the set are adding, removing, testing, and listing the elements in the set, and one cannot access a specific object with an address like an array. There are two types of sets, HashSets and TreeSets, which implement the set interface. These are going to be explained later.

Then there is the Map<K,V> interface where there is a set of keys that point to values specific to them. The map cannot have duplicate keys and a key can only point to one value. However multiple keys can have the same value. Some of the commands to manipulate a map are put(), get(), containsValue(), containsKey(), and remove() which are all demonstrated below in the example. Maps also have two implementations in the Java Library, HashMap and TreeMaps.

TreeSets and TreeMaps are implementations of SortedSet and Map .These are Sets and Maps that have ordered elements, so many times it is easier to use a tree to go through a set or map in an organized fashion. The objects being stored have to implement the comparable interface so the program can put the objects in the tree in a sorted order. These types of sets and maps store information in trees called binary trees. Then the computer uses a binary search to find the elements, and this is why it is faster to use this in cases where you need to go through a sorted list rather than a list with no order. This is like a linked list; however, each element points to two elements after it and not just one. One of these child nodes has to be less than the object and one has to be greater than it. Adding and removing elements is also very similar to linked lists.

HashSets and HashMaps are implementations of Set and Map. These are more efficient than trees if one just wants to add, remove, and locate elements. These use something called Hash Tables to access the different elements without using a liner search. Each object has a specific hash code, however two objects could have the same code, and this is called collision. This hash code points to an object like an index does in an array. Each object has a method for generating this Hash code called a hash function, and a good function can help avoid these collisions. This is why it is easy to add, remove, and check if an object is in a hashset or hashmap.

So which one should I use? If one has a good hash function for the stored objects then hashsets/maps are faster than tree based structures. However the tree-based structures have guaranteed performance. So if one does not want to make a good hash function then the tree set is a good way to go. Tree sets also have another advantage that iterators go through it in order where as in a hash set it goes through it randomly.

For example let's say we have a set of student ids. The following is how Maps can be instantiated and manipulated.

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

//Adds the keys with the corresponding values to the Map
studentIds.put(123, "John");
studentIds.put(213, "Amanda");
studentIds.put(313, "Smith");

//This prints the student with 123 as his or her id
System.out.print(studentIds.get("123"));

//Checks to see if there is a student with 315 as his or her id
if(studentIds.containsKey("315")){
     System.out.print("Student 315 Exists"));
}

//Checks if one of the values (students) is in the map.
if(studentIds.containsValue("John")){
     System.out.print("John Exists"));
}

//Removes key 313 from the list
studentIds.remove(313);
```