HashMaps vs HashTables vs HashSets

In Java, there are three main collection objects that make use of hashing to implement their internal structure. This document provides a brief rundown on what sets them apart, for an extensive look at their implementation and all their available functions see the documentation section at the bottom of the document. See the *hashcollections* Java project for coding examples of basic usage.

HashSet

This implements a structure via the Set interface, found within the Collections API. It doesn't use key-value pairs in the traditional sense, (i.e. an integer *key* matches with a String *value*). Instead it calculates a hash value from the object that it stores which serves as the key for that object. This in turn allows for constant (O(1)) access, addition and removal of elements. It however, doesn't keep track of the order in which the elements are stored.

To check if a set contains an element we must provide an identical element to retrieve it. This means that HashSets are good for quickly searching for the existence of an element within a set but cannot be used to retrieve additional information regarding the given element. Like other Set objects, the HashSet cannot contain duplicate elements.

The HashSet permits the use of the NULL element.

HashMap

HashMaps implement a structure via the Map interface, found within the Collections API. They are more akin to the hashtables that we have already seen. Unlike their set counterpart, HashMaps use key-value pairs to store data. For example, a unique String *key* can be mapped to an Integer *value.* Keys need to be unique, but it is not a requirement for values. The HashMap allows for constant (O(1)) access, addition and removal of elements. It isn't however guaranteed to keep track of the order in which elements are stored.

The HashMap permits the use of NULL for both keys and values.

Hashtable

Very much a legacy option, used primarily for backwards compatibility. Generally, you want to avoid this. Its functionality is very similar to that of a HashMap, which should be used in its place.

Its unique feature is that it is synchronized meaning that it can handle modification and access from two different sources at the same time, but this can be achieved with both the HashMaps and HashSets through proper implementation. This will unlikely be of any concern to you at this time, but if you are interested, the HashMap documentation provides a manner to achieve synchronised usage of HashMaps.

The Hashtable doesn't permit the use of NULL keys or values.

Documentation

https://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html

https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html

https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html