Ch 18.3 Sets

The `Set` interface is-a `Collection`. How is Set different from List? (page 1077)

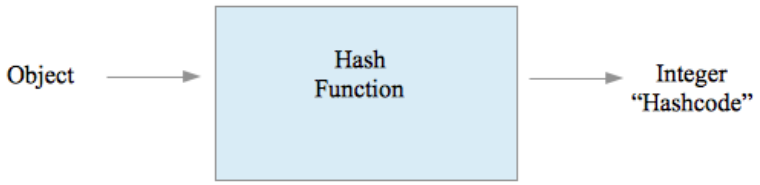
- No duplicate elements in a Set
- Set is unordered
- Use Set if “fast retrieval is important”
- “Useful when you have a large collection of data”

There are two flavors: `HashSet` and `TreeSet`.

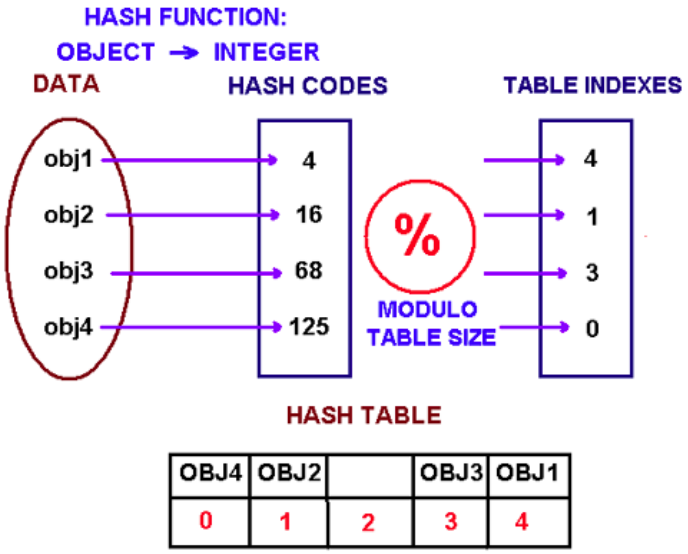
HashSet

`HashSet` implements `Set` using a hash table. The goal in this is to provide O(1) searching.

A hash table depends on the existence of a black box (a method) to turn an object into a semi-random large integer. The hash function must be reproducible and easy to calculate.



A *hash function* translates an object into an integer *hash code*. The hash code is then modulo the size of an array where objects are stored. This array is called a *hash table*.



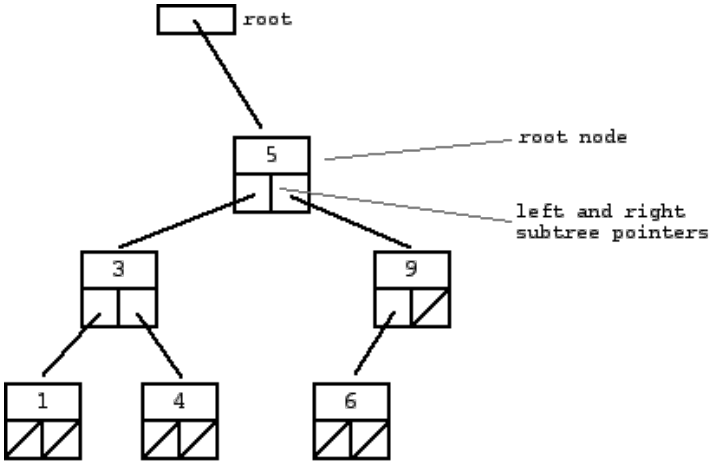
Every `Object` has a `hashCode()` method. The default method for String is commonly used. You can override `hashcode()` for your own class if you like.

`LinkedHashSet` maintains the order of insertion into a `HashSet` by storing this ordering in a list. So, if you need this order, then use `LinkedHashSet`. It costs extra time and memory.

Set/TreeSet

The TreeSet data structure is called a binary search tree (BST). In a BST,

- the left child of a tree node is less than the node
- the right child of a tree node is greater than the node



In a BST, data is inserted in sorted order.
The BST is O(log N) performance for searching for an item.
As we did previously for binary search and sort, you can use `Comparable` or a `Comparator` to order your TreeSet.
There are many, many flavors of BST.

Hash tables and trees are very important topic in CSC 210. But you can get a basic understanding from some great Javascript animations of these data structures here:

www.cs.usfca.edu/~galles/visualization/Algorithms.html