Sets and Maps

CMSC 132

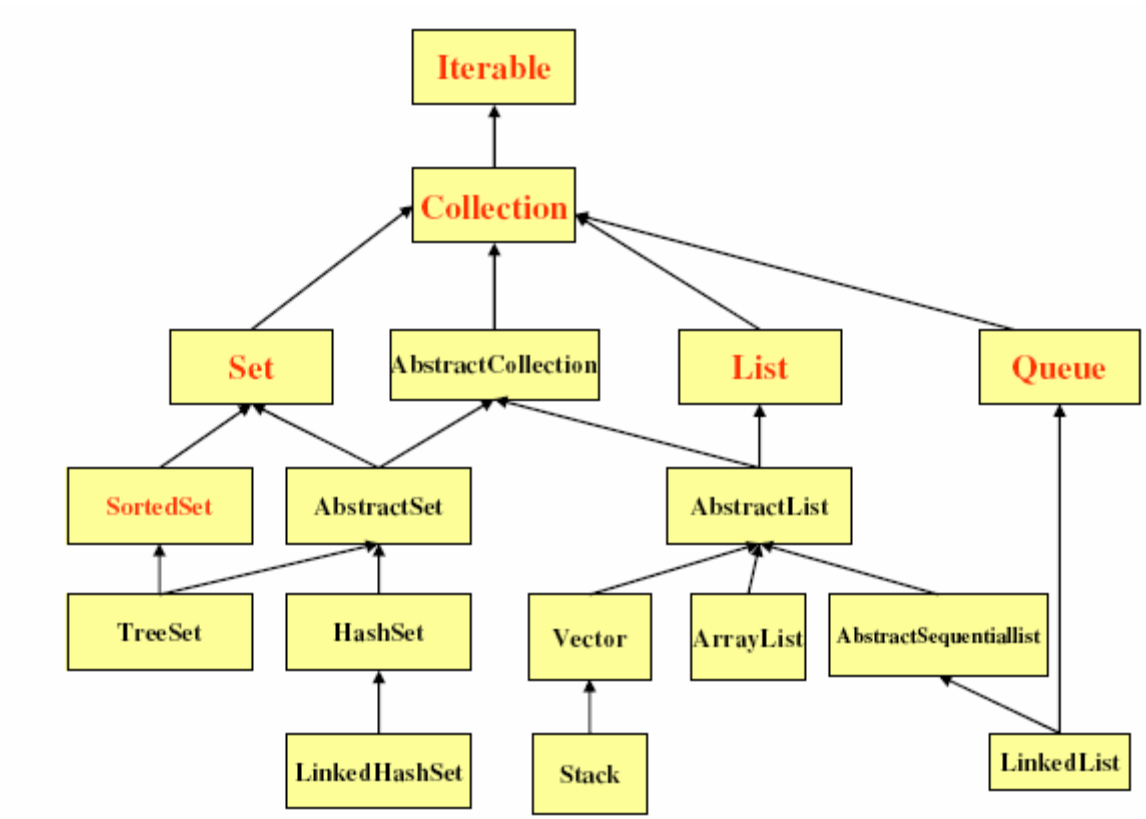
Department of Computer Science

University of Maryland, College Park

Overview

- Sets
- Fundamental Set Operations
- Set Concrete Classes
- How do sets work?
- hashCode()
- SortedSet
- Maps

Collection Hierarchy



•Interfaces – represented by red font
•Classes – represented by black font

Sets

- A Set interface represents a set
  - set - collection of elements without duplicates
- No front or back
- The order in which elements are added to a set doesn't matter
- But a set (generally) offers the ability to find or remove an element quickly
  - Without having to search through all the elements

Fundamental Set operations

- contains - does this set contain a value?
- remove - remove a value from the set (return true if it was found)
- add - add value to the set

Set Concrete Classes

- HashSet
  - Object added to the set most implement the hashCode method
  - Most common set implementation
- LinkedHashSet
  - Extends HashSet with a linked list implementation supporting ordering of elements
  - Elements can be retrieved in the order they were inserted
- TreeSet
  - Guarantees elements in the set are sorted
  - Elements can be added to the set as long as they can be compared (elements must implement the Comparable interface or an comparator must be specified)
- See examples in code distribution

How do sets work?

- **Finding a matching element is based on the equals method**
  - **If you want a collection of your own classes, you will need to define your own equals(Object) method**
  - **If you don't define an equals method with an Object parameter then you will have problems while using your object with classes expecting the equals(Object) method**
  - **If you don't override equals you have reference comparison (i.e., a.equals(b) → a == b)**
- **HashSet is most common Set implementation**
  - **Needs a hashCode() function**

hashCode()

- Function must return some int such that
  - if x.equals(y),
  - then we are guaranteed that
  - x.hashCode() == y.hashCode()
- Hashcode already defined for many classes
  - String, Integer, etc.
- The hashCode method allow us to implement a strategy called hashing. In this strategy you can store elements in a table (usually called the hash table) based on the value returned by the hashCode.
  - Example: We can use a hash function that places students in an array based on the student's id
- HashSet relies on the hashCode method to manipulate elements. If that method is not correct then your HashSet will not work as expected.

hashCode()

- For a Point class:
  - class Point {
    final int x,y; // notice they are final
    public int hashCode() { return x + 17*y; }
    public boolean equals() {…}
    }
- If you don't overwrite hashCode then usually the returned value is the conversion of the object address into an integer.
- If you overwrite equals you should overwrite hashCode.
- Potential problem if class is used where hashing is required
  - Suppose you overwrite equals correctly but don't overwrite hashCode.
  - Two objects that are considered "equal" can be mapped to different hash table entries.

Good hashCode() functions

- A good hashCode function is one that tries to make it so that two different objects are likely to have different hashCodes
  - Can't make promises; for any hashCode function, there are unequal objects that have equal hashCodes
- But any valid hashCode function is better than an invalid one
  - public int hashCode() { return 42; }

We'll come back to hashCode

- We'll come back to hashCode and hash tables later
- Designing a good one, and implementing a HashSet is a little bit trickly
- But an important and valuable thing to understand and something that will be useful to you

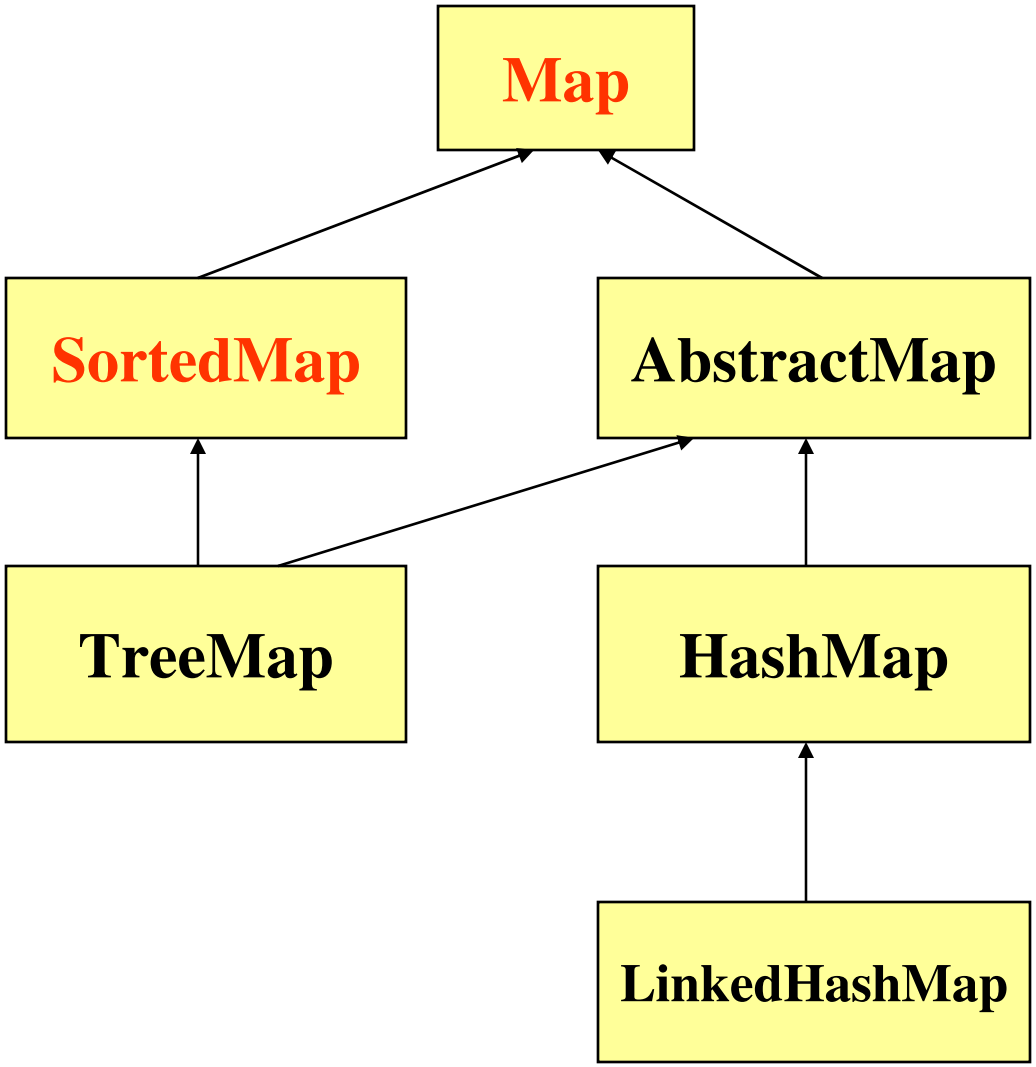
SortedSet

- A Set of ordered objects
  - For example, Integers or Strings
  - Allows you to ask for things such as the smallest value, or largest value
    - Can also ask for a headSet or tailSet
      - A headSet is the set of all values less than a supplied value
  - Can use objects that implement the Comparable interface
  - Or provide a Comparator object as a argument to the constructor

Map

- A dictionary
  - A set of keys
  - For each key, an associated value
- You can think of it as an array that can be indexed by any value
- Methods on Map
  - get(key)
  - put(key, value)
  - containsKey(key)
  - remove(key)
  - keySet()

Map


Map
SortedMap
AbstractMap
TreeMap
HashMap
LinkedHashMap

Map notes

- If you call get and the key isn't present, null is returned
- Map is not a collection
  - But you can get the keySet()
  - For more advanced usage, might also want entrySet() or values()
- HashMap is most common implementation
  - Also a TreeMap (which is a SortedMap)

Examples

- See map examples in code distribution