CSE-2102
Object Oriented Programming

Dr. Muhammad Ibrahim
Assistant Professor
Dept. of Computer Science and Engineering
University of Dhaka

Topics

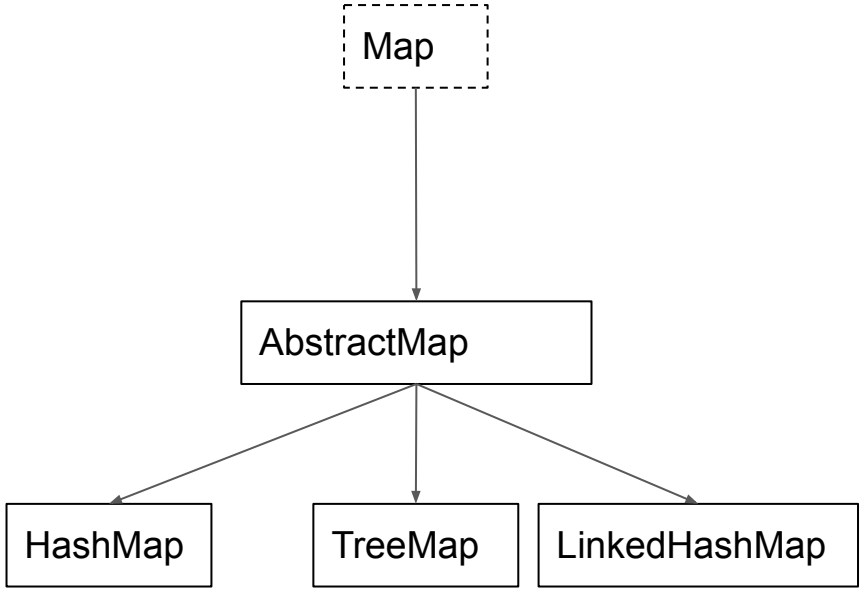
- Collections framework
  - Usage of classes
    - Treemap
    - LinkedHashMap
  - User-defined classes in collection
  - Iterator and for-each loop
  - Comparator interface
  - Collections algorithms
  - Arrays class

TreeMap

- Unlike HashMap, it keeps the elements in sorted (ascending) order of keys
- Declaration: `class TreeMap<K, V>`
- Constructors
  - `TreeMap( )`
  - `TreeMap(Comparator<? super K> comp)`
  - `TreeMap(Map<? extends K, ? extends V> m)`
  - `TreeMap(SortedMap<K, ? extends V> sm)`

LinkedHashMap

- It maintains a linked list of the entries in the map, in the order in which they were inserted. This allows insertion-order iteration over the map.
- Declaration: `class LinkedHashMap<K, V>`


Map
AbstractMap
HashMap
TreeMap
LinkedHashMap

User-Defined Classes in Collections

- Instead of using type wrappers (Integer, String, Double etc.) we can use user-defined objects for the classes of collection framework.
- Let us see an example with ArrayList…

for-each Loop and Iterator

- We’ve already examined the for-each style loop.
  - `for (int val : mylist ) {  System.out.println(val);  }`
- A more systematic way is to use `Iterator` interface.
- `Iterator` (and its group) provides some operations as well as traversing a collection.
  - E.g.: both way traversal, modifying an element using `ListIterator` interface.
- `ListIterator` is only available to those collections that implement `List` interface.
- The bottom line: if we don’t need to traverse a collection in reverse order or modify elements, for-each loop is more handy to use.

Comparator

- We've already learnt that both `TreeSet` and `TreeMap` keep the elements in sorted order.
- How the order is dictated? Java by default provides a "natural order".
  - "A" before "B", 1 before 2 and so on.
- We can explicitly tell how this sorting will be performed using `Comparator` interface.
  - E.g.: changing the sorting from ascending to descending
  - We can define the `Compare` method of `Comparator` interface that tells how to order two objects
  - If we specify `Comparator` during construction of a `TreeSet` or `TreeMap` or `Collections` class etc., Java will perform sorting using our defined `Compare` method.

Comparator

- Declaration:
  - `interface Comparator<T>`
    - T specifies the type of the object being compared
  - `int compare (T obj1, T obj2)`
    - Returns 0 if obj1 and obj2 are equal, a positive value if obj1 is greater than obj2, and a negative value otherwise
    - Now is is our job to override this method as required.
- Lets us examine the code named CompDemo.java that demonstrates how can we utilize `Comparator` interface (through compare method) to reverse the sorted order of a `TreeSet` collection.

Comparator

- Now that we know how to use `Comparator` interface, let us examine a more practical example: sorting the objects based on a particular field that contains different persons data …

The `Collections` Class

- Several algorithms are defined on collection that can be readily used for different types of collection data structures
- Let us examine two of them: shuffle and sort

```
Collections.shuffle(any_List)

Collections.sort(any_List)
```

The `Arrays` Class

- Provides several algorithms specific for arrays - for both primitive types and user-defined types
- Let us examine two of them: sort and search

```
Arrays.sort(any_array_of_Object);

Arrays.binarySearch(any_array_of_Object, an_Object);
```

End of Lecture 16.