Java Collections Framework

JCF 3 & 4

The Java Collections Framework

- A collection of interfaces and classes that implement useful data structures and algorithms
- The `Collection` interface specifies how objects can be added, removed, or accessed from a `Collection`

Objectives

- Create Tree Set and use its methods
- Create Hash Set and use its methods
- Create Hash Map and use its methods

Sets

- A Set is an unordered `Collection` of unique elements (i.e., no duplicate elements).
- The collections framework contains several `Set` implementations, including HashSet and TreeSet.
- `HashSet` stores its elements in a hash table, and `TreeSet` stores its elements in a tree.

Sets (cont.)

- The collections framework also includes the SortedSet interface (which extends `Set`) for sets that maintain their elements in sorted order.
- Class `TreeSet` implements `SortedSet`.
- TreeSet method headSet gets a subset of the `TreeSet` in which every element is less than the specified value.
- TreeSet method tailSet gets a subset in which each element is greater than or equal to the specified value.
- SortedSet methods first and last get the smallest and largest elements of the set, respectively.

SortedSet and TreeSet

- See code

```
sorted set: black green grey orange red tan white yellow
headSet ("orange"):  black green grey
tailSet ("orange"):  orange red tan white yellow
first: black
last : yellow
```

Fig. 20.17 | Using `SortedSet`s and `TreeSet`s. (Part 3 of 3.)

The Java Collections Framework

- Brief introduction to just two implementations, **`HashSet`** and **`HashMap`**

FIGURE 12.2 Selected Methods in the `Collection` Interface

`public boolean add(Base_TypenewElement)` Adds the specified element to the collection. Returns `true` if the collection is changed as a result of the call.
`public void clear()` Removes all of the elements from the collection.
`public boolean remove(Object o)` Removes a single instance of the specified element from the collection if it is present. Returns `true` if the collection is changed as a result of the call.
`public boolean contains(Object o)` Returns `true` if the specified element is a member of the collection.
`public boolean isEmpty()` Returns `true` if the collection is empty.
`public int size()` Returns the number of elements in the collection.
`public Object[] to Array()` Returns an array containing all of the elements in the collection. The array is of a type `Object` so each element may need to be typecast back into the original base type.

`HashSet` Class

- Used to store a set of objects
- Uses the same `<>` notation as an `ArrayList` to specify the data type
- View source code, listing 12.2
  `class HashSetDemo`
- A `Set` is a `Collection` of unique elements (i.e., no duplicate elements).

Hash Set Demonstration

- See Code

HashSet hashCode() method

- The **hashCode**() method of **HashSet** in Java is used to get the hashCode value for this instance of the HashSet. It returns an integer value which is the hashCode value for this instance of the HashSet.
- A hash code is an integer value that is associated with each object in java.

Sample Screen Output

```
The set contains:
2
3
7
The set contains:
2
7
Set contains 2: true
Set contains 3: false
```

```java
import java.util.*;

public class GFG {
    public static void main(String[] args)
    {
        // creating an HashSet
        HashSet<String> arr
            = new HashSet<String>();

        // using add() to initialize values
        // [Geeks, For, ForGeeks, GeeksForGeeks]
        arr.add("Geeks");
        arr.add("For");
        arr.add("ForGeeks");
        arr.add("GeeksForGeeks");

        // print HashSet
        System.out.println("HashSet: "
                           + arr);

        // Get the hashCode value
        // using hashCode() value
        System.out.println("HashCode value: "
                           + arr.hashCode());
    }
}
```

Output:

HashSet: [ForGeeks, Geeks, For, GeeksForGeeks]
HashCode value: -482506029

`HashMap` Class

- Used like a database to efficiently map from a **key** to an **object**
- Maps associate keys to values.
  - The keys in a `Map` must be unique, but the associated values need not be.
- Uses the same `<>` notation as an `ArrayList` to specify the data type of both the key and object
- View source code, listing 12.3
  `class HashMapDemo`

FIGURE 12.3 Selected Methods in the `Map` Interface

`public` *`Base_Type_Value`* `put(`*`Base_Type_Key`* `k,` *`Base_Type_Value`* `v)` Associates the value `v` with the key `k`. Returns the previous value for `k` or `null` if there was no previous mapping
`public` *`Base_Type_Value`* `get(Object k)` Returns the value mapped to the key `k` or `null` if no mapping exists.
`public void clear()` Removes all
`public` *`Base_Type_Value`* `remove(Object k)` Removes the mapping of key `k` from the map if present. Returns the previous value for the key `k` or `null` if there was no previous mapping.
`public boolean containsKey(Object k)` Returns `true` if the key `k` is a key in the map.
`public boolean containsValue(Object v)` Returns `true` if the value `v` is a value in the map.
`public boolean isEmpty()` Returns `true` if the map contains no mappings.
`public int size()` Returns the number of mappings in the map.
`public Set <`*`Base_Type_Key`*`> keySet()` Returns a set containing all of the keys in the map.
`public Collection <`*`Base_Type_Values`*`> values()` Returns a collection containing all of the values in the map.

Hash Map Demonstration

- See Code

Sample Screen Output

```
Map contains:
K2 --> 28251 feet.
Denali --> 20355 feet.
Kangchenjunga --> 28169 feet.
Everest --> 29029 feet.
Denali in the map: true
Changing height of Denali.
Map contains:
K2 --> 28251 feet.
Denali --> 20320 feet.
Kangchenjunga --> 28169 feet.
Everest --> 29029 feet.
Removing Kangchenjunga.
Map contains:
K2 --> 28251 feet.
Denali --> 20320 feet.
Everest --> 29029 feet.
```

TreeMap

TreeMap class implements Map interface similar to HashMap class.
The main difference between them is that **HashMap** is an unordered collection while TreeMap is **sorted** in the ascending order of its keys.

```
Import java.util.*;
public class Details {

    public static void main(String args[]) {

        /* This is how to declare TreeMap */
        TreeMap<Integer, String> tmap =
                new TreeMap<Integer, String>();

        /*Adding elements to TreeMap*/
        tmap.put(1, "Data1");
        tmap.put(23, "Data2");
        tmap.put(70, "Data3");
        tmap.put(4, "Data4");
        tmap.put(2, "Data5");

        System.out.println(tmap);
        }

    }
}
```

```
Output:
{1=Data1, 2=Data5, 4=Data4, 23=Data2, 70=Data3}
```

- We have inserted the data in random order however when we displayed the TreeMap content we got the sorted result in the ascending order of keys.

Summary

- Learned:
  - Tree Set
  - Hash Set
  - Hash Map
  - Tree Map