Section 12: Java Collections

15 July 2021 19:23

Section 12: Java Collections

Use Java Collections:

```
Private Collection<Seat> seats = new ArrayList<>();
Private Collection<Seat> seats = new LinkedList<>();
Private Collection<Seat> seats = new HashSet<>();
Private Collection<Seat> seats = new LinkedHashSet<>();
Private Collection<Seat> seats = new ArrayList<>();
```

Shallow Copy : Taking a copy, a shallow copy of the same data. Both have same shared objects.

Deep Copy : As opposed to a shallow copy, a deep copy is a copy where the elements are not just references to the same element as in the original list, but are themselves copied.

Interfaces Java Docs : https://docs.oracle.com/javase/tutorial/collections/interfaces/index.html

Comparable and Comparator : The comparator interface defines a single method called compare. It's similar to comparable. The comparator interface defines a single method called compare. Unlike comparable the objects to be sorted don't have to implement comparator. Instead an object of type comparator can be created with a compare method that can sort the objects that we are interested in.

Maps : (Key, Value) Pair. For a particular key, there will be only one value and if we try to insert more it will be overridden.
https://docs.oracle.com/javase/tutorial/collections/interfaces/index.html

```
Map<String, String> languages = new HashMap<>();
Languages.put("first language", "hindi");
Languages.put("second language", "english");
```

To print the value :

```
System.out.println(languages.get("first language");
```

To check a key already exist or not :

```
if(languages.containsKey("first language"))
{ ............. }
```

There is another methos called put if absent and that’s only gonna add to the map if the key is not already present.

For printing :

```
for(String key: languages.keySet())
{
        System.out.println(key + " : " + languages.get(key));
}
```

There is no order in hashmap.

May more methods such as .remove, .replace, etc

Immutable Classes : are a great way to increase encapsulation.

1) Don't provide "setter" methods
2) Make all fields final and private.
3) Don't allow subclasses to override methods. The simplest way to do this is to declare the class as final.

https://docs.oracle.com/javase/tutorial/essential/concurrency/imstrat.html

Sets & HashSet : Set cannot defined ordering number, and can't contain duplicates.
https://docs.oracle.com/javase/8/docs/technotes/guides/vm/performance-enhancements-7.html

HashSet - equals() and hashCode()
https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals-java.lang.Object-

For overcoming object duplication in sets, we override the hashcode() method.

```
@Override
  public int hashCode() {
    System.out.println("hashcode called");
    return this.name.hashCode() + 57;    // 57 is just a random number.
  }
```

If we mark any method final, it means that method can't be overridden (in sub classes)

Sets - Symmetric & Asymmetric :
https://docs.oracle.com/javase/tutorial/collections/interfaces/set.html

In Set theory 2 differences are defined : symmetric difference and asymmetric difference.
Java doesn't have method for calculating the symmetric difference possibly that's use less often.

The symmetric difference is the element that appear in one set or the other but not both so it can therefore as the union minus intersection

Linked HashMap/TreeMap : Store Data in alphabet order (sorted).

UnmodifiableMap : It's unmodified Map.