Collection

- Java Collections is a framework with custom data structures (java’s own implementations) and rich set of java libraries which enable you do the common operations like searching , sorting , placing unique items in a list, organising data into key value pairs and many more.
- It takes a call on itself seeing the nature of data that which sort or search algorithm it should apply , it just needs the java user to invoke sort/search method.
- High abstraction is provided by the virtue of rich apis’ provided and helps focus more on the business logic implementation.

Collection hierarchy

- By this time we have already learnt about java interfaces , class and methods . We would be highly leveraging interfaces in java collections as we have previously also studied that it’s always a neat and standard way to program to interface (P2I) .
- Collection is the interface which is extended by List and Set interface.
- Classes like ArrayList , LinkedList and Vector implement List interface.
- Classes like HashSet , LinkedHashSet implement Set interface . SortedSet interface extend Set interface . TreeSet implements SortedSet interface.
- Classes like HashMap , LinkedHashMap implement Map interface . SortedMap interface extend Map interface.TreeMap implements SortedMap interface.
- Map interface is not extended by Collection, yet we study it in java Collection.

Generics

- Before generics the problem faced was the collection like ArrayList used to contain items as Object (parent class in java to all the classes). So even if we used to feed a fish , cat or dog object to the arrayList , it used to return it simply as Object . Then we had to typecast it to our domain object say fish or cat to get our domain object back.
- The process mentioned above is tedious and error prone at run time.There used to be a scope of error that the object while receiving could be typecasted to wrong domain object giving runtime exceptions.
- Then came generics which is a standard code provided by which the compiler stops us to put a cat object in a list of fish objects . This mistake is caught at run time and it’s always better to catch error at compile time rather at run time. This is what we call type-safety and is also a strong feature from java 1.5 onwards.

Generics - type safety and no typecasting

- Suppose you create an arrayList of fish objects using generics and then you want to insert a dog object into that list , it would be caught at the compile time.
- The following code snippet is to help you understand the concept

```
List<Fish> listOfFish = new ArrayList<Fish> ;  // creating a list of fish objects using generics ie. the code under
//angular brackets '<>'

listOfFish.add(dog) ; // it gives a compile error at that moment when we try to insert a dog object in list of fish
// objects.

Fish fish = listOfFish.get(0); // this gets us back the fish object from the list without any type casting hassles.
```

List , ArrayList

- List is the interface which has methods declared to keep items of similar or subtypes. It has basic methods declared to add and remove the items .
- It has three implementing classes ArrayList , LinkedList and Vectors.
- ArrayList is used primarily when we know that most of the data insertion is not at the middle . Its best suited when the data keeps getting added at the end . It internally leverages Array data structure which is subscript based (like array[0]), so when we insert some data at the start or middle , it shifts remaining all the data one block rightwards . This is highly computation intensive in that case.
- If we use arrayList for insertion at end , it happens very fast and also access is also very fast as it’s an index based access.

ArrayList , LinkedList

```
List<Integer> listOfIntegers = new ArrayList<Integer>; // sample code of use of arrayList
listOfIntegers.add(1); // adding of integer to the list
listOfIntegers.add(2);
```

- LinkedList is used and best preferred where the data insertion is in the middle of the list .
- Its because it internally leverages LinkedList data structure where the insertion in the middle of data happens by the pointer of one node pointing to the new node to be added and the pointer of the new node points to the next node before which it has to be added .
- Pointer contains the address of the node and each node had two parts - data and pointer.

Vector , Set , HashSet

- Vector is similarly like arrayList but it is thread safe and synchronized (these terms are already discussed in multithreading) . If an arrayList has to be concurrently used by many end clients who want to modify it at the same time , the Vector should be better used.
- Set is the interface which has api support for keeping only unique items.
- HashSet is the class implementing Set interface .

```
// sample code using HashSet and Set

Set<Integer> setOfIntegers = new HashSet<Integer>;

setOfIntegers.add(1);

setOfIntegers.add(2);// here if i would have added 1 again , it would be a compile time error.
```

SortedSet , LinkedHashSet ,

- LinkedHashSet implements Set . HashSet is an unordered collection ie. the items are not in the HashSet in the order they were inserted . LinkedHashSet maintains the order and is an ordered HashSet where you can retrieve the items in the same order in which they were inserted.
- SortedSet is the interface extending Set and TreeSet is the class implementing SortedSet . The items in the TreeSet class are in a sorted order

Map , HashMap , LinkedHashMap

- Map in an interface which has api support to store and access the data in key, value paris. All keys are unique here while the values can be duplicated.
- HashMap is a class implementing Map interface

```
// sample code for HashMap
Map<String,String> map = new HashMap<String,String>();// making HashMap object
map.put('java','headFirst'); // putting java as key and headFirst as value
map.get('java'); // fetching the value by passing the key 'java'
```

- Difference between HashMap and LinkedHashMap is that HashMap is unordered while LinkedHashMap is ordered in terms of storage and access.

SortedMap and HashTable

- Sorted Map is the interface extending Map interface which has api support to keep the keys sorted . TreeMap is the class implementing this interface and does the functionality of keeping the keys unique and sorted also.
- HashTable is the class implementing Map interface but it is synchronized (thread safe). We need such a map when our map is being used by many clients concurrently for modification and access .

Sorting and Comparable interface

- Suppose we have an arrayList of integers and we want to sort it , we just need to call Collections.sort() and pass the arrayList object as the argument to the method.
- In case we have a scenario where we have an arrayList of song objects and the Song class has fields like artists , song name etc.. . So the question arises on which field the sort method should be called on . Here is where we use Comparable interface.
- The domain class (Song) needs to implement Comparable interface and thus necessarily override ‘ compareTo’ method where the particular fields as specified of the Song object can be compared.
- Then finally we call Collections.sort and pass list of songs as argument . Since the Song class is implemnting Comparable , it now knows on which field to sort.

Using comparators for sorting.

- At times we are told to sort a list of domain objects like songs on different fields like artists,song name varying time to time in the same code . Had we been using Comparable we had to re-override compareTo method and change the logic from song name to artist name say. This is again not so neat process.
- What we do that we use Comparators instead. We write an inner class in the main method where we are calling sort method on Collections . Inner class is a class within a class say a nested class. Sample comparator code is as follows :

```
 class ArtistCompare implements Comparator<Song> {
public int compare(Song one, Song two) {
return one.getArtist().compareTo(two.getArtist());
}
```

Comparators

- compareTo returns 0,1and -1 as per it the objects compared are equal , greater or leser.
- The ArtistCompare class that we just wrote can be duplicated for songNameCompare class also. Then we just need to instantiate the inner class and pass it’s object as second parameter to Collections.sort .
- The first argument will be the list of song objects and the second arguement will be the object of inner class implementing Comparator which shows which field of the song object should be sorted.
- The biggest advantage is the decoupled design that we follow so that we don’t have to keep rewriting the logic of compareTo method had we made the domain Class (Song) implement the Comparable interface

Packaging java classes

- We write all the java code so that it may run and to run we need .class files . But if we have 1000 java files , do we really need to export 1000 .class files for the same ?
- No , it’s not that complicated , we just need to export those java classes collectively as .jar file.
- A .jar file contains all the .class files and also a manifest.xml which tells the starting point of execution for all the .class files listed. We can extract a jar to find the .class files and manifest.xml . A .jar file is nothing but java archive.

HttpServlet and tomcat

- Servlets are java classes that are able to process http request or response.
- Servlets are the shorthands used for HttpServlet .
- HttpServlet are processed by the servlet containers inside webservers like apache tomcat. Webservers are meant to deliver / render webpages to the client.
- Client and the server communicate via a protocol called http (hyper text transfer protocol)
- Servlets have a service method where they receive http requests from the clients , process them in the server via the servlet container of the tomcat engine and then render back servlet response which are usually in terms of html pages , images and stylesheets.

RMI

- RMI stands for remote method invocation where the internal implementation is about making calls from one jvm instance to another jvm instance and passing of serialized java objects within them .
- This happens in distributed networks and happens jvm’s within a machine and also from jvm’s of another machine.

Exercise

- Go throuh all the codes inside com.musigma.javatraining
- Try to relate task instances running in map-reduce slots with task tracker inside a slave machine.

Mu Sigma

Mu Sigma

Mu Sigma