Collections

- Amal Prasad

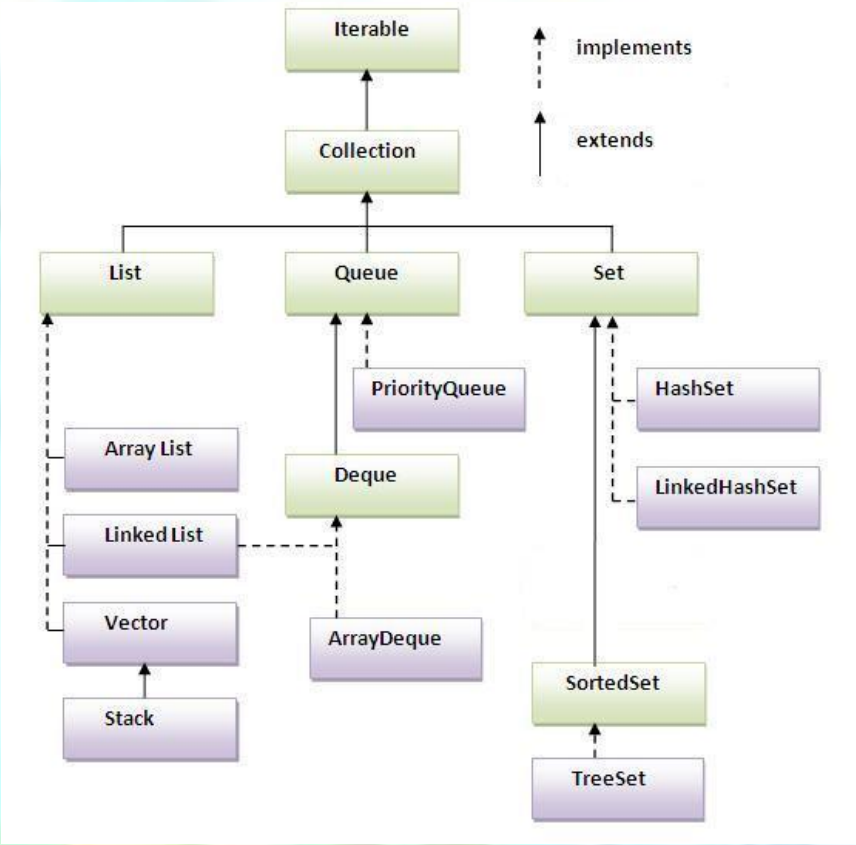
Collections in Java

- **Collections in java** is a framework that provides an architecture to store and manipulate the group of objects
- All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections
- Collection represents a single unit of objects i.e. a group
- **What is framework in java**
  - provides readymade architecture
  - represents set of classes and interface
  - is optional
- **What is Collection framework**
  - Interfaces and its implementations i.e. classes
  - Algorithm

Amal Prasad

Hierarchy of Collection Framework

- The **java.util** package contains all the classes and interfaces for Collection framework



Amal Prasad

ArrayList

- Uses a dynamic array for storing the elements
- Extends AbstractList class and implements List interface
- Can contain duplicate elements
- Maintains insertion order
- Allows random access because array works at the index basis
- Manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list

Non-generic Vs Generic Collection

- Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic
- Generic collection allows you to have only one type of object in collection
- ArrayList al=new ArrayList();//creating old non-generic arraylist
- ArrayList<String> al=new ArrayList<String>();//creating new generic arraylist
- In generic collection, we specify the type in angular braces. Now ArrayList is forced to have only specified type of objects in it
- If you try to add another type of object, it gives *compile time error*

Example of ArrayList

- Example1
- Example 1 -> Convert in for-each loop
- Example2 (Class object Array List)
- **addAll(Collection c) method**
- **removeAll() method**
- **retainAll() method**

Amal Prasad

LinkedList class

- Uses doubly linked list to store the elements
- Can contain duplicate elements
- Maintains insertion order
- Manipulation is fast because no shifting needs to be occurred
- Can be used as list, stack or queue

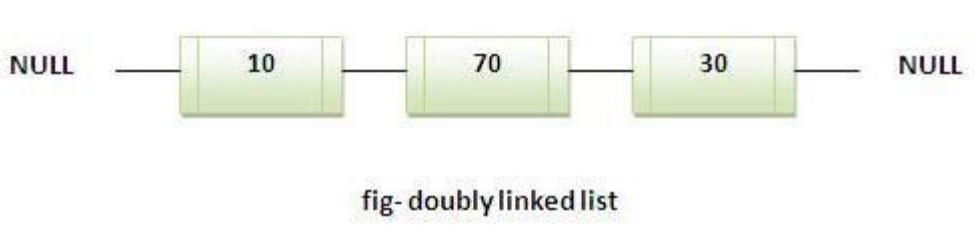


fig- doubly linked list

- Example LinkedList

Amal Prasad

ArrayList vs LinkedList

ArrayList

1) ArrayList internally uses **dynamic array** to store the elements.

2) Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.

3) ArrayList class can **act as a list** only because it implements List only.

4) ArrayList is **better for storing and accessing** data.

LinkedList

LinkedList internally uses **doubly linked list** to store the elements.

Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory.

LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.

LinkedList is **better for manipulating** data.

Amal Prasad

HashSet class

- Uses Hashtable to store the elements
- Contains unique elements only
- **Difference between List and Set**
  - List can contain duplicate elements whereas Set contains unique elements only
- Example

LinkedHashSet class

- Contains unique elements only like HashSe
- Maintains insertion order
- **Example of LinkedHashSet class**

Amal Prasad

TreeSet class

- Contains unique elements only like HashSet
- Maintains ascending order
- **Example of TreeSet class**

Amal Prasad

Map Interface

- A map contains values based on the key i.e. key and value pair
- Each pair is known as an entry
- Map contains only unique elements

HashMap class

- Contains values based on the key
- Contains only unique elements
- It may have one null key and multiple null values
- It maintains no order
- **Example of HashMap class**

Amal Prasad

LinkedHashMap class

- Contains values based on the key
- It contains only unique elements
- It may have one null key and multiple null values
- It is same as HashMap instead maintains insertion order
- **Example of LinkedHashMap class**

Amal Prasad

TreeMap class

- Contains values based on the key
- Contains only unique elements
- It cannot have null key but can have multiple null values
- It is same as HashMap instead maintains ascending order
- **Example of TreeMap class**

Amal Prasad

HashMap vs TreeMap

1) HashMap is can contain one null key.	TreeMap can not contain any null key.
2) HashMap maintains no order.	TreeMap maintains ascending order.

Amal Prasad

Comparable interface

- Used to order the objects of user-defined class.
- java.lang package
- Only one method named compareTo(Object)
- It provide only single sorting sequence i.e. you can sort the elements on based on single datamember only
  - For instance it may be either rollno,name,age or anything else
- We can sort the elements of:
  - String objects
  - Wrapper class objects
  - User-defined class objects
- **String class and Wrapper classes implements the Comparable interface. So if you store the objects of string or wrapper classes, it will be Comparable**
- **Example of Sorting the elements of List that contains user-defined class objects on age basis**

Comparator interface

- **Comparator interface** is used to order the objects of user-defined class
- Present in java.util package and contains 2 methods
  - compare(Object obj1,Object obj2)
  - equals(Object element)
- It provides multiple sorting sequence i.e. you can sort the elements based on any data member.
- Example of Comparator

Comparable vs Comparator

Comparable	Comparator
1) Comparable provides **single sorting sequence**. In other words, we can sort the collection on the basis of single element such as id or name or price etc.	Comparator provides **multiple sorting sequence**. In other words, we can sort the collection on the basis of multiple elements such as id, name and price etc.
2) Comparable **affects the original class** i.e. actual class is modified.	Comparator **doesn't affect the original class** i.e. actual class is not modified.
3) Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.
4) Comparable is found in **java.lang** package.	Comparator is found in **java.util** package.
5) We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list elements of Comparator type by **Collections.sort(List,Comparator)** method.

Amal Prasad

Thanks…

Amal Prasad