Pg1

Thursday, June 4, 2020   7:58 AM

<u>Why do we need collection Framework in java ?</u>

To store multiple objects or group of objects together we can generally use arrays. But arrays has some limitations

Limitations of Array :

1. The size of the array is fixed, we cannot reduce or increase dynamically during the execution of the program.
2. Array is a collection of homogeneous elements.
3. Array manipulation such as :
    1. removing an element from an array.
    2. adding the element in between the array etc…

    Requires complex logic to solve.

Therefore, to avoid the limitations of the array we can store the group of objects or elements using different data structures such as :

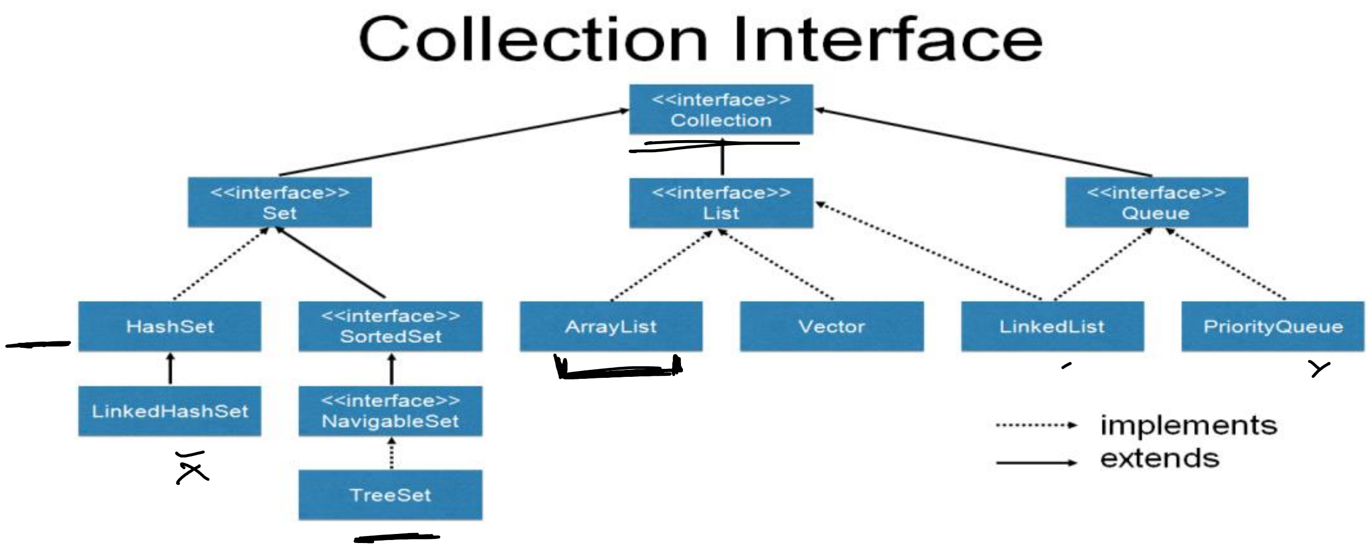
1. List
2. Set
3. Queue
4. maps / dictionaries

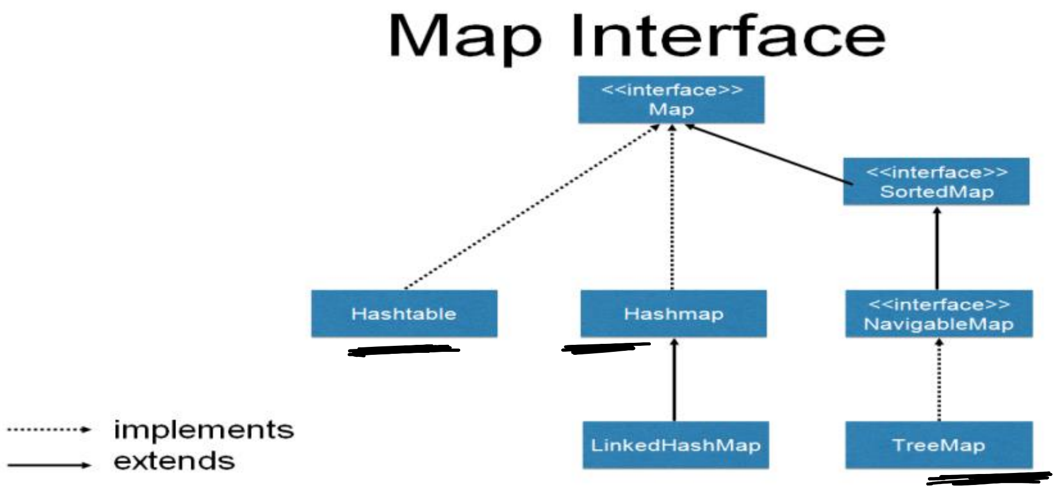
Def : Collection Framework is set of classes and interfaces (hierarchies), which provides mechanism to store group of objects (elements) together.

It also provides mechanism to perform actions such as :
(CRUD - operations)
1. create and add an element
2. access the elements
3. remove / delete the elements
4. search elements
5. update elements
6. sort the elements

Note : 2 important hierarchies of collection frame Work is :

1. Collection hierarchy
2. Map hierarchy





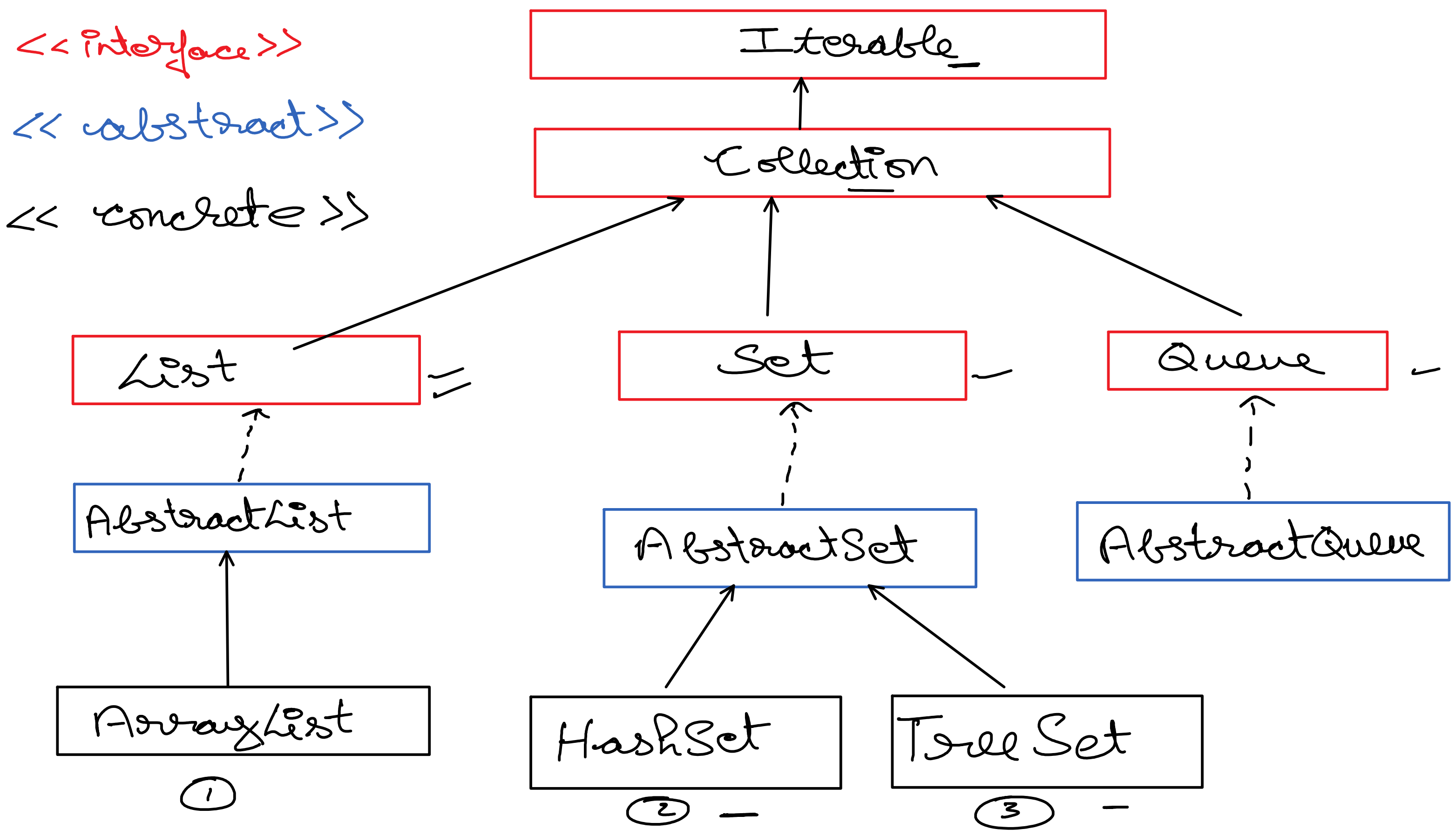
Pg2

Friday, June 5, 2020 7:59 AM

Collection interface :

1. Collection is an interface defined in java.util.
2. Collection interface provides the mechanism to store group of objects(elements) together.
3. All the elements in the collection are stored in the form of Objects. (i.e. only Non-primitive is allowed)
4. which helps the programmer to perform the following task.

   a. **add an element into the collection**
   b. **search an element in the collection**
   c. **remove an element from the collection**
   d. **access the elements present in the collection**



Methods Of Collection interface :

Purpose	return type	method signature
add an element		**1.** add(Object) **2.** addAll(Collection)
search an element		1. contains(Object) 2. containsAll(Collection)
remove element		1. remove(Object) 2. removeAll(Collection) 3. retainAll(Collection) 4. clear()
Access elements	Iterator	1. iterator( ) 2. we can access elements with the help of for each loop.
Miscellaneous		1. size() 2. isEmpty() 3. toArray() 4. hashCode() 5. equals()

List interface:

1. List is an interface defined in java.util package.
2. List is a sub interface of Collection interface. Therefore, it has all the methods inherited from Collection interface.

Methods Of List Interface :

Purpose	return type	method signature
add an element		**1.** add(Object) **2.** addAll(Collection) ================ 3. add(int index , Object)

		4. addAll(int index , Collection)
search an element		1. contains(Object) 2. containsAll(Collection) ================ 3. indexOf(Object)
remove element		1. remove(Object) 2. removeAll(Collection) 3. retainAll(Collection) 4. clear() ================= 5. remove(int index)
Access elements	Iterator	1. iterator( ) ================= 2. listIterator() 3. get(int index) ================= Note : we can access with the help of for each loop.
Miscellaneous		1. size() 2. isEmpty() 3. toArray() 4. hashCode() 5. equals()

What is list ? What is it Characteristics ?

1. List is a collection of objects.
2. List is an ordered collection of Objects. (It maintains the insertion order of elements)
3. List has indexing, therefore, we can insert, remove or access elements with the help of index.
4. List can have duplicate objects.

<u>ArrayList :</u>

1. It is a Concrete implementing class of List interface.
2. The characteristics of ArrayList is same as List.
3. ArrayList is defined in java.util package.

Note:

1. All the abstract methods of List and Collection interface is implemented.
2. We can create an instance of ArrayList

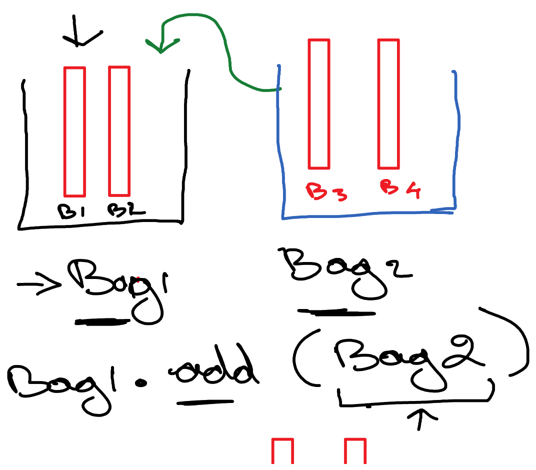
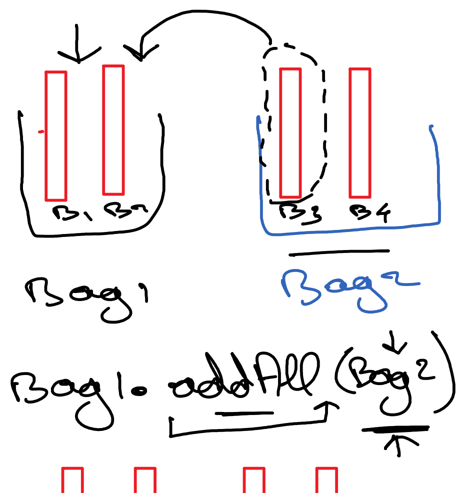
Constructors:

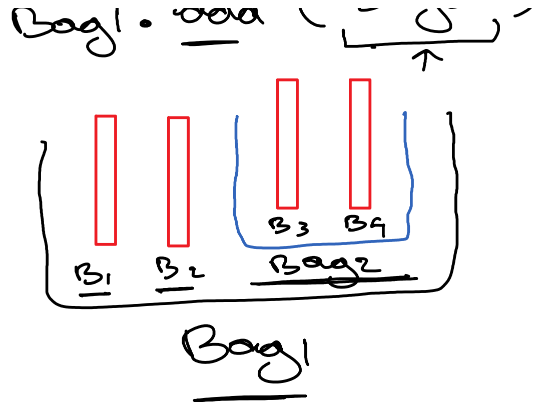
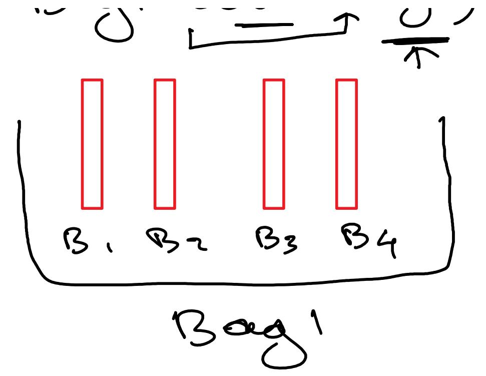
ArrayList( )	creates an empty ArrayList Object
ArrayList(Collection)	creates an ArrayList Object, and copies all the elements present in the given collection into the ArrayList created.

Note :

1. toString method is Overridden such that it returns a view of all the elements present in the array list.

   [element1 , element2, …] ….

Add(Object)	AddAll(Collection)
	

 bag1.size() ---> $\boxed{3}$ --> nested Collection object	 bag1.size() --- > 4 --> not nested

Note :

The elements in the ArrayList can be accessed in the following ways :

1. get method
2. iterator
3. ListIterator
4. for each loop

1. get(index) :

    1. get method is used to access the elements present in the ArrayList.
    2. it accepts index as an argument.
    3. It returns the Object type.

    Example refer : **workspace/collections/src/arraylist/get**

2. To access the elements of ArrayList using iterator.

    1. iterator() is method which belongs to Iterable Interface.
    2. iterator() method creates an Iterator type object and returns the reference.
    3. iterator() method returns the reference in Iterator (interface) type

java.util.Iterator :

Iterator interface has 2 important methods to perform iteration.

1. **hasNext()** : it checks whether there is an element to be accessed from the collection, if present it returns true, else it returns false.

2. **next()** : it is used to access the element, and moves the cursor to the next element. The return type of next() is Object.

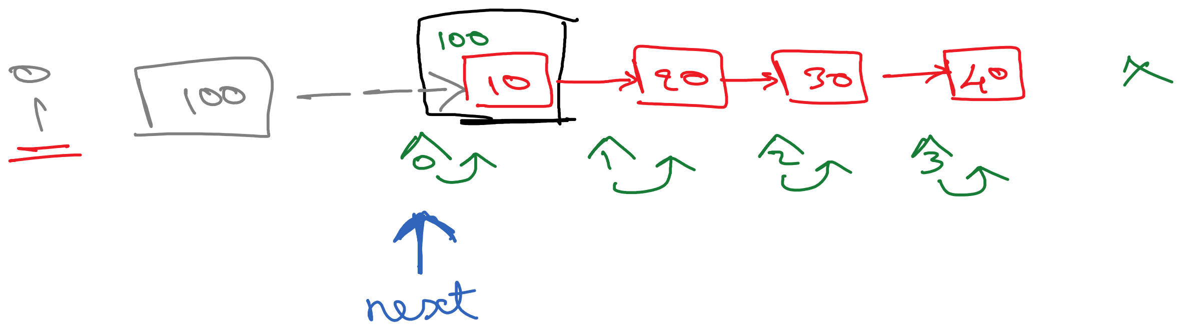
example :

Let us assume ArrayList :

ls = [10, 20, 30, 40]

Step 1 :

Iterator i = ls.iterator() ;



Step 2 : Access the items / elements in the collection

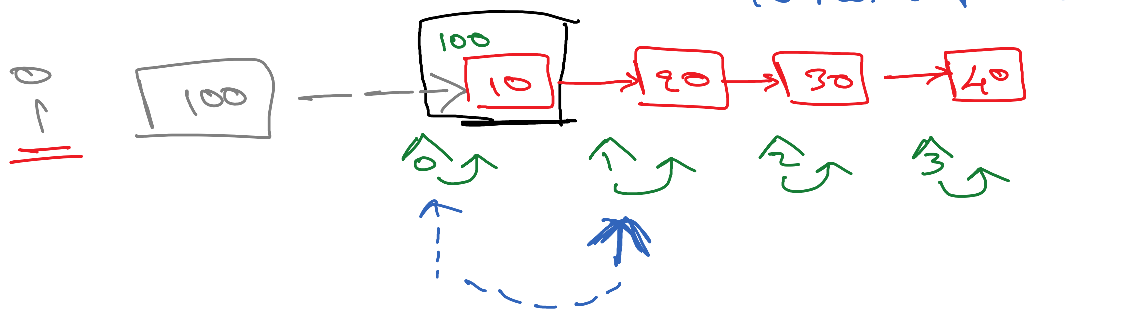
1.) use the method next ()

i. next()

1. gives element
2. move cursor

i. next()

1. gives element
2. move cursor to next position



→ 10 is given in the form of Object type

NoSuchElementException :

In the Iterator, when we try to access an element using next() method and if there is no element present, we get NoSuchElementException.

example :

refer,

Disadvantage Of Iterator :

1. We can iterate only once.
2. We cannot access the elements in reverse order.

3. **listIterator() :**

listIterator() is a method that belongs to List interface, listIterator method creates an object of ListIterator type.

1. return type of listIterator() is ListIterator interface.

ListIterator :

ListIterator is a sub interface (child) of Iterator interface, it is defined in java.util package.

Methods of ListIterator :

hasNext()	to check whether next element to be accessed is present or not.
next()	it gives the element, and moves the cursor forward.
hasPrevious()	to check whether previous element to be accessed is present or not.
previous()	it gives the previous element pointed by the cursor, and moves the cursor backward.
remove(Object)	it removes the current object pointed, from the collection, which is under iteration.
add(Object)	it is used to add a new object into the collection.

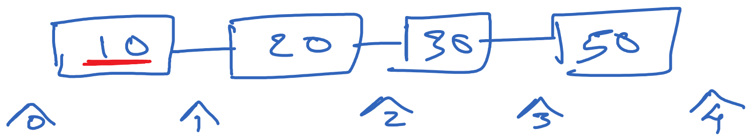
Example:-

(a) Let us consider ArrayList:

ls = [10, 20, 30, 40]

Step1: create ListIterator

ListIterator i = ls.listIterator()



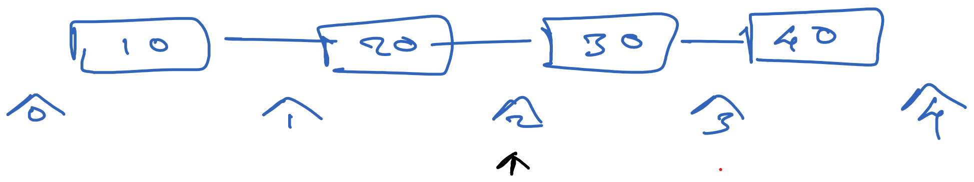
Note :

listIterator() method is overloaded :

1. listIterator( ) :- creates an ListIterator object type, and cursor will be pointing to first element.
2. listIterator(int) :- creates an ListIterator object type, and cursor will be pointing to the position provided.

example :

ListIterator i = ls.listIterator(2) ;



i.next() : 30

Example refer :
===

4. **for each loop** :

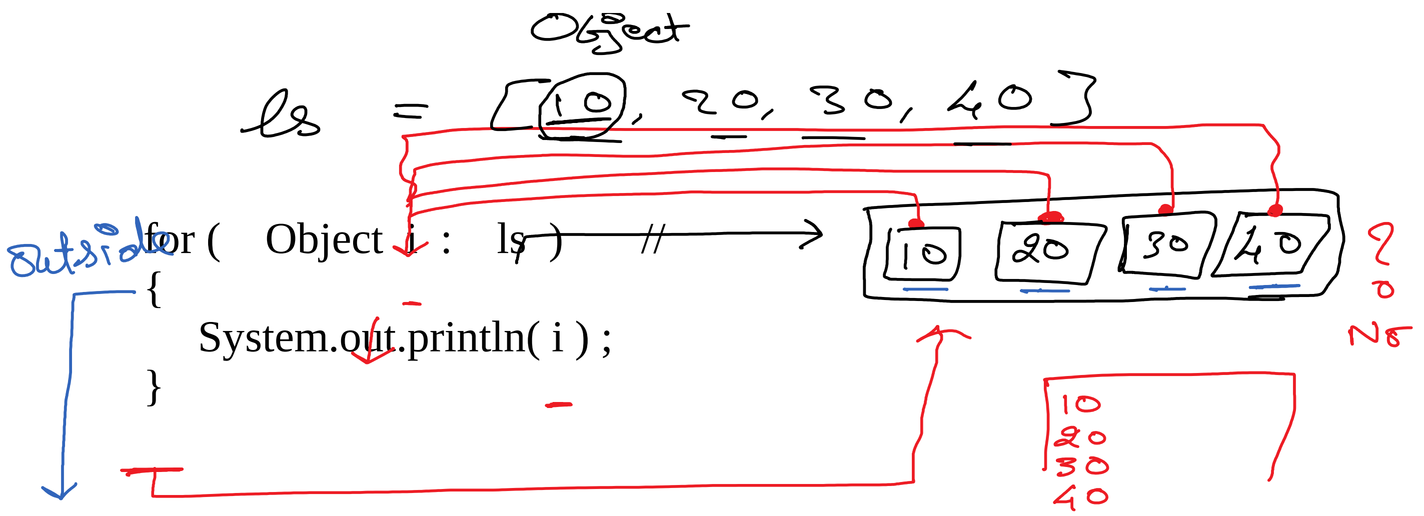
syntax :

```
    for (  type  variable :   reference of  collection/Array )
    {
        // statements
    }
```

1 value

Object

```
for (    Object i  :    ls )
{
    System.out.println( i ) ;
}
```



Pg3

Wednesday, June 10, 2020 8:54 AM

Note :

We can classify collections into two categories :

1. non-generic collection.
2. generic collection.

1. Non-Generic Collection :

   it is a heterogeneous (different types) collection of elements. Every element is converted and Stored as java.lang.Object class type.

   Example refer :

2. **<u>Generic Collection :</u>**

   It is a homogeneous collection of elements, (collection of same type of elements).

   <u>Syntax to create Generic Collection :</u>

   1. Syntax to create reference variable for Generic Collection:

   Collection_type< Non-primitive Datatype > variable

```
ArrayList< Integer >   ls  ;
```

2. Syntax to create Generic Collection Object :

```
new  Collection_name< Datatype > ()  ;

new ArrayList< Integer >()  ;
```

```
ArrayList<Integer> ls  =  new ArrayList<Integer>()  ;
```

arraylist where we can store only integers

From JDK 7 onwards,.

```
ArrayList<Integer> ls  = new ArrayList< >()  ;
```

Note :

1. the elements are not converted to java.lang.Object class type, instead they are tried to be converted for the given Generic Type.
2. The return type of the elements in Generic collection will be same as the given Generic Type

To Sort an ArrayList :

We can sort the arraylist with the help of sort() method present in **Collections** class

1. sort method is static in Collections class.
2. sort method is Overloaded.

    1. sort(arraylist ls)
    2. sort(arraylist ls , comparator ref)

1. sort(arraylist ls) :

    > it sorts the arraylist by default in ascending order.
    > it can sort only the arraylist, if the arraylist is homogeneous (all elements should be of same type), if not we get ClassCastException.
    > The elements in the arraylist must be **comparable type**. If it is not Comparable type we get ClassCastException.

2. sort(arraylist , Comparator) :

    > if the elements of arraylist is not comparable type and we have to sort the arraylist we can achieve it with the help of Comparator.
    > It also sorts the arraylist in ascending order by default.

Example Programs : refer,

workspace/collections/src/arraylist/sort/ StudentDriver1.java

To Sort in Descending order :

Step1 : sort the array list in ascending order first.

Step2 : use Collections.reverse() method to reverse the order of Collection

Example Programs :   refer,

workspace/collections/src/arraylist/sort/S4.java

==

Pg5

Thursday, June 11, 2020 8:55 AM

<u>Set :</u>

1. Set is an interface, it is a sub interface of Collection, therefore all the methods of Collection is inherited.
2. Set is defined in java.util package.

<u>Characteristics Of Set :</u>

1. it is an unordered collection of elements. (the order of insertion is lost)
2. Set does not allow duplicate elements.
3. Set does not have indexing. Therefore, we cannot access, insert or remove based on index.
4. We can access the elements of set only by using:
    1. iterator()
    2. for each loop

<u>Concrete Implementing Classes Of Set :</u>

1. **<u>HashSet :</u>**

    - it is a Concrete implementing class of Set interface.
      It has all the methods inherited from Collection interface.

Characteristics :

1. It is unordered.
2. No Duplicates
3. No index

Constructors :

HashSet() :	creates an Empty HashSet object
HashSet(Collection c)	creates an HashSet and will copy all the elements of the Collection into the HashSet.

Methods:

To add an element	1. add(Object) 2. addAll(Collection)
TO search an Element	1. contains(Object) 2. containsAll(Collection)
To remove an element	1. remove(Object o) 2. removeAll(Collection c) 3. retainAll(Collection c) 4. clear ()
To access the elements	1. iterator() 2. using for each loop stmt
Misc.	1. isEmpty() 2. size() 3. toArray() 4. equals() 5. hashCode()

2. <u>**TreeSet :**</u>

1. It is a concrete implementing class of Set interface.
2. TreeSet will also have all the methods of Collection interface.

Characteristics :

1. The elements will be sorted by default.
2. The elements to be added in treeset must be Comparable type, else we

get ClassCastException.

3. All the elements in TreeSet should be of same type(Homogeneous), if not we get ClassCastException.
4. Duplicate elements not allowed.
5. No indexing, therefore we cannot add, remove elements using index.

Constructors :

TreeSet()	it creates empty TreeSet Object
TreeSet(Collection c)	it creates a TreeSet, and copies all the elements from the Collection given into the TreeSet.

Methods :

To add an element	1. add(Object) 2. addAll(Collection)
TO search an Element	1. contains(Object) 2. containsAll(Collection)
To remove an element	1. remove(Object o) 2. removeAll(Collection c) 3. retainAll(Collection c) 4. clear ()
To access the elements	1. iterator() 2. using for each loop stmt
Misc.	1. isEmpty() 2. size() 3. toArray() 4. equals() 5. hashCode()

Pg6

Saturday, June 13, 2020 7:53 AM

Maps :

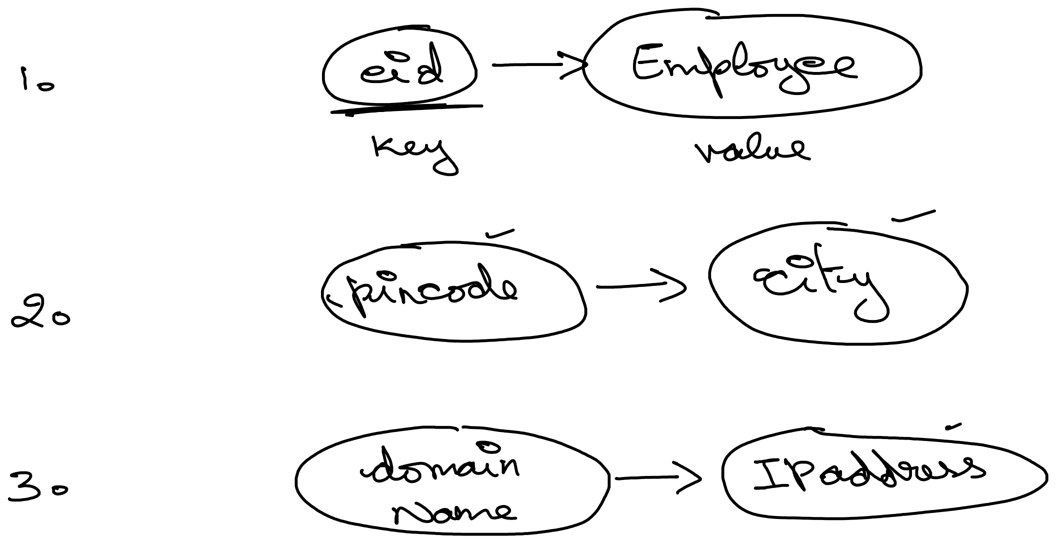
Map is a data structure, which helps the programmers to store the data in the form of key value pairs.

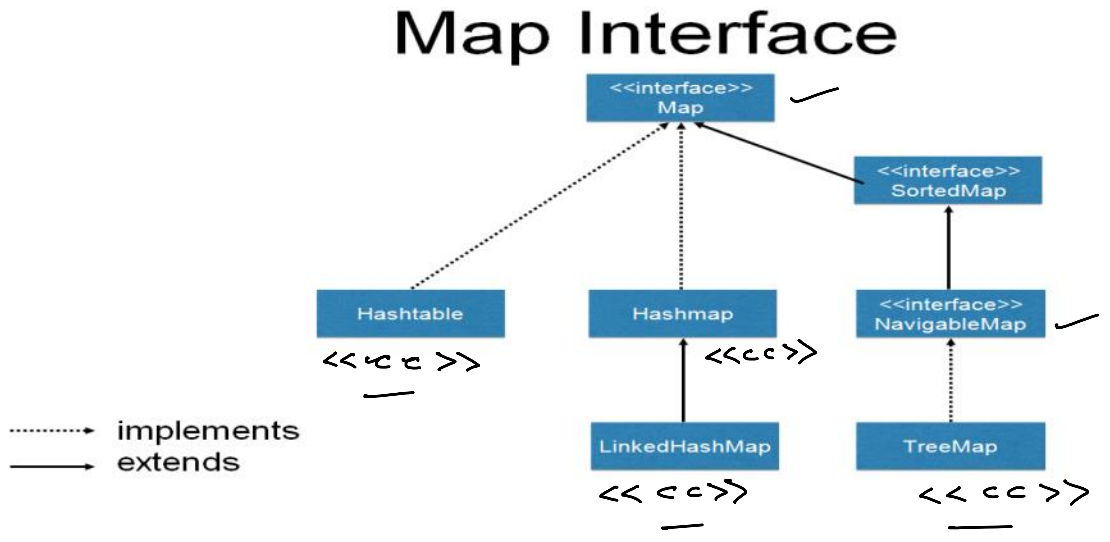
Where every value is associated with a unique key.

Note :

1. key cannot duplicate.
2. one key can be associated with only one value.
3. Maps helps us to access the values easily with the help of its' associated key.

example:





1. Map is an interface in java defined in java.util package.
2. We can create generic map by providing the type for both key as well as value. < key_type , value_type >, < K , V >
3. we can obtain 3 different views of a map.

   a. we can obtain a list of values from a map. - collection
   b. We can obtain a set of keys from a map. - set
   c. We can obtain a set of key-value pairs. — set

Key		value
101	=	smith
102	=	martin
103	=	Miller

a. list of values

[smith , martin , Miller]

b. set of keys

[101 , 102 , 103]

[101 , 102 , 103]

c. set of Key= value pairs

$$\rightarrow \left[\overbrace{\left(\underbrace{\underline{101} = \underline{smith}}_{\text{Key value pair}} \right)}^{\text{element}} , \left(102 = martin \right) , \left(103 = miller \right) \right]$$

Note :	One key value pair is called as an entry or a Mapping

Methods Of Map interface :

Method Signature	purpose
put(key , value)	1. add an entry to the map (key-value pair) 2. replace the old value with a new value of an existing entry in the map.
putAll(Map)	it will copy all the entries from the given map into the current map.
containsKey(key)	if the key is present returns true else returns false
containsValue(value)	if the value is present it returns true, else it returns false.
remove(key)	if the key is present the entry is removed from the map and the value is returned. if the key is not present nothing is removed and null is returned.
clear()	it removes all the entries in the map.

get(key)	it is used to access the value associated with a particular key. if the key is present it returns the value. if the key is not present it returns null.
values()	it returns a collection of values present in the map. return type Collection< v >
keySet()	it returns a set of keys present in the map return type Set<k>
entrySet()	it returns a set of all the entries present in the map. return type Set< < k, v> >
size()	
isEmpty()	
equals()	
hashCode()	

HashMap :

It is a concrete implementing class of Map interface.

1. data is stored in the form of key=value pair.
2. Order of insertion is not maintained.
3. key cannot be duplicate, values can be duplicate.
4. key can be null.
5. value can be null.

TreeMap :

It is a concrete implementing class of Map interface.

1. it also stores data in key=value pair.
2. it will sort the entries in the map with respect to keys in ascending order.
3. The key in TreeMap must be comparable type. If it is not comparable type we get ClassCastException.

4. In TreeMap key cannot be null, If it is null we get NullPointerException.
5. A value in TreeMap can be null.

HashTable :

It is a concrete implementing class of Map interface.

1. It is used to store data in key=value format.
2. in hashtable the insertion order is not maintained.
3. In hashtable both key and value cannot be null. If it is null we get NullPointerException.

==

Task :

1. refer LinkedList, LinkedHashSet, queue

==

Conversions :

1. To Convert ArrayList to HashSet

```
ArrayList ls = new ArrayList() ;
ls.add(10) ;
ls.add(10);
ls.add(20);
ls.add(10);
ls.add(30);
Set s = new HashSet(ls);
S.o.pln(s);
```

2. **HashSet to ArrayList**

3. **ArrayList to TreeSet**

4. **TreeSet to ArrayList**

5. **Map to Set :**

with the help of entrySet() method of Map interface

```
Map m1 = new HashTable();
  m1.put(1, "hello");
  m1.put(2, "bye");
Sopln(m1); // { 1=hello, 2=bye }
// Map to Set.
  Set s1 = m1.entrySet();
Sopln(s1); // [ (1=hello), (2=bye) ]
```

To Convert A collection into an Array :

We can convert a Collection into an Array with the help of toArray() method present in Collection interface.

The return type of toArray() method is an Object[]

Task2 :

1. Design a method to convert an Array into an ArrayList.

    input : array
    task: convert array to Arraylist
    return : ArrayList

```
static ArrayList toList( int[] arr )
{
    ArrayList ls = new ArrayList()  ;
    for( int  j = 0 ; j < arr.length ; j++ )
    {
        ls.add( arr[j] ) ;
    }
    return ls  ;
}
```

2. Design a method to convert an Array into an HashSet.
3. Design a method to convert an Array into an TreeSet.

=======================================

Note :

1. We cannot access or iterate a map using Iterator or For each loop

Note :

1. **We cannot access or iterate a map using Iterator or For each loop.**