Collection frameWork

* Represent group of objects as a single entity.
* Introduced in 1.2 version.
* Provides architecture to store and manipulate the data.
* Contains group of classes and Interfaces to work with the different objects.
* Most collection frameworks are belongs to Util package.

  `java.util.*;`

Collection frameWork

- Collection
  - List
  - set
  - Queue
- Map

1. Collection Data
2. Map Data
3. Cursors
4. Sorting.
5. Stream API

Collection frameWork:-

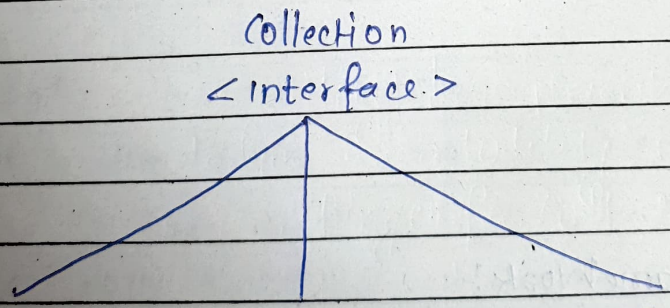
It defines several classes and interfaces which can be used to represent a group of object as a single entity

Java	C++
(I) Collection.	(I) container
(II) Collection frameWork	(II) STL

Collections

Arrays	Collection framework
(i) can store Homogenous data	(i) can store Heterogeneous data.
(ii) primitive as well as object data can be stored.	(ii) only store object data.
(iii) Size is fixed	(iii) size is dynamic.
(iv) Memory Wastage	(iv) No Memory Wastage.
(v) Operations are complex.	(v) Easy as it contains More methods for operations.
(vi) performance is good	(vi) lower as compared to Array.

* Collection	vs Collections
* Root Interface	class contains methods to perform operations.



* Root Interface of Collections : Collection
* " " " Map : Map
* Java.util.package.
* Parent interface of collection : Iterable.

*

Key Interfaces

Collection
List
Set
Sorted Soot
NavigableSet
Queue

Map
SortedMap
NavigableMap
Map Entry
Cursors
Enumeration
Iterator
ListIterator

Sorting
Comparable
Comparator.

Utility classes

(i) Collections
(ii) Arrays

// List Interface

Iterable
collection
List

List Implementation classes.

(i) Array List (1.2)
(ii) LinkedList (1.2).
(iii) Vector (1.0) } legacy
(i) Stack. (1.0) } classes

characteristics of collection framework.

(i) Version
(ii) Hetrogenous data
except: TreeSet TreeMap (Homogeneous)
(iii) Representing group of objects.
(iv) Cursors used to iterate over collections.
(v) NULL insertion: all classes.
(vi) Duplicate objects: Allowed
except: Set Not allowed.
(vii) Methods: Synchronized or non-syn
1.0 ver: sync latter → non-sync
(viii) Insertion order: All list classes preserved but all set not preserved.
(ix) Resizable Array, Double LinkedList, Hash Table are data storage format used in collection framework.

Page No.
Date 08/01/2022

List Implementation Classes

ArrayList

```
ArrayList al = new ArrayList();
```

9 key Interfaces of collection framework.

(1) Collection (I) : 1.2V

* If we want to represent a group of individual objects as a single entity then we should go for collection.
* Collection Interface defines most common methods which are applicable for any collection object.
* In general, collection interface is considered as root interface of collection framework.

Note :- There is no concrete class which implements collection Interface directly.

Collection :- an interface which can be used to represent a group of individual objects as a single entity.

while,

Collections is a utility class present in java.util package to define several utility methods (like sorting searching) for collection objects.

Page No.
Date

(II) List (I) : Came in 1.2 Version.
(i) Duplicates are allowed.
(ii) Insertion order is preserved.

List :- Implementation classes :-
* Array list - 1.2
* LinkedList - 1.2
* Vector - 1.0 } legacy classes.
* Stack - 1.0

(III) Set (I)
- Duplicates are not allowed.
- Insertion order Not preserved.

Set :- Implementation classes :-
* HashSet - 1.2 V
* LinkedHashSet - 1.4 V

(IV) SortedSet (I) - 1.2 V
* child interface of set
* Duplicates Not allowed.
* Insertion according to some order.

(V) NavigableSet (I) : 1.6 V
* child interface of SortedSet.
NavigableSet
* TreeSet
- Defines several methods for navigation purposes.

(VI) Queue (I) :- 1.8V

implementation classes :-
* Priority Queue - 1.5V
* Blocking Queue - 1.5V
  * Linked Blocking Queue - 1.5V
  * Priority Blocking Queue - 1.5V.

* If we want to represent a group of individual objects prior to processing then we should go for Queue.

Note :- All the above interfaces meant for representing a group of individual objects.

* if we want to represent a group of objects as key-value pairs then we should go for Map interface.

(VII) Map (I) : Not child of Collection Interface.
* Represent as group of key-value pair.
* Duplicate keys are not allowed.
* Duplicate values are allowed.

Map - 1.2V Implementation classes :-
* HashMap - 1.2V
  * LinkedHashMap - 1.4V
* WeakHashMap - 1.2V
* IdentityHashMap - 1.4V
* HashTable (child of Dictionary) - 1.0V
  * Properties - 1.0V.

Page No.
Date

(VIII) Sorted Map (I) – 1.2V
- Child Interface of Map
- Represent a group of objects as key-value pair according to some ~~order~~ sorting order of keys.

Map
* SortedMap
* NavigableMap
* TreeMap.

(IX) NavigableMap – 1.6V
- child Interface of SortedMap
- Defines several Utility Methods for Navigation Purpose
- Implementation class: TreeMap (1.2V)

* OverView

Collection

(I) List – 1.2V
ArrayList – 1.2
LinkedList – 1.2
Vector – 1.0
Stack – 1.0

(2) Set – 1.2V
HashSet – 1.2V
LHS – 1.4V
SortedSet – 1.2V
NavigableSet – 1.6V
TreeSet – 1.2V

(III) Queue
PQ
*BQ
PBQ
LBQ
1.5

IC

Map – 1.2V

Dictionary

(I) HashMap – 1.2
→ LinkedHashMap – 1.4V
(2) WeakHashMap – 1.2V
(3) IdentityHashMap – 1.4V

(II) SortedMap – 1.2
→ NavigableMap – 1.6
→ TreeMap – 1.2

(III) HashTable
→ properties
1.0

Sorting
(I) Comparable :- Default Natural Sorting Order.
(II) Comparator: customized Sorting order.

Date / /

* Collection Interface

* Root Interface of collection Framework.
* Methods

boolean (i) add (object o)
boolean (ii) addAll (Collection c)
boolean (iii) remove (Object o)
boolean (iv) removeAll(Collection c)
Void ~~boolean~~ (v) clear()
boolean (vi) retainAll (Collection c)
boolean (vii) isEmpty()
int (viii) size()
boolean (ix) contains(Object o)
boolean (x) containsAll(Collection c)
object[] (xi) toArray() - return type object array
Iterator (xii) iterator() - return type Iterator object.

Note:- collection interface doesn't contain any method to retrieve object.

List Interface

* Index play Important role.
* By default all collections methods are available to List.
* Specific Method

add(index, obj)
addAll(index, c)
remove (int index)
indexof (object o)
lastIndexOf (object o)
get (int index)
listIterator ( )
set(int index, object o)

* ## ArrayList

(I) Underlying data structure :- Resizable Array / Growable Array
(II) Duplicate Allowed.
(III) Insertion order preserved.
(IV) Heterogenous Allowed (except TreeSet TreeMap)
(V) Null insertion is possible.

Constructors of ArrayList :-

1. ArrayList e = new ArrayList();
// Default capacity = 10.
for increasing capacity whenever required.
New capacity = $(\frac{3}{2} * \text{old Capacity} + 1)$

for:- old capacity = 10
$NC = \frac{3}{2} * 10 + 1 = \underline{16}$

II ArrayList L = new ArrayList (int initialCapacity);

(III) ArrayList L = new ArrayList (Collection c);
// Meant for inter-conversion of Collections.

Note:- if we are calling ArrayList Reference Variable, it will print all elements.
Basically in collection, if we are ~~call~~ printing reference Variable, Internally toString() is called and elements are Printed inside square braces.

* Usually we can use Collections to hold and transfer Objects from one place to another place, to provide support for this requirement every collection already Implements Serializable and cloneable interfaces.

Page No.
Date

* RandomAccess Interface
- Present in util package
- It's a marker interface so "it doesn't contain any method".

* Arraylist and Vector Implements RandomAccess Interface so that we can access any Random element with the same speed.
Hence if our frequent operation is retrieval operation, than Arraylist is best choice.

* Arraylist is worst choice if our frequent operation is insertion or deletion in the middle. Because several shift operation are required.

Diff. b/w AL and Vector.

AL	Vector.
(i) Method present Non-Synchronized	Synchronized.
(ii) Not-Thread Safe	Thread-Safe.
(iii) Performance Higher than Vector	lower.
(iv) 1.2 Version	1.0 Version
Non-Legacy classes.	Legacy Classes
(v)	

Page No.
Date

* How to get Synchronized ArrayList Object?
By default ArrayList Object is non-synchronized but we can get synchronized version of Arraylist by using Collection class synchronizedList() method.

Syntax :-

Public static List synchronizedList (List l);

ex:- ArrayList aL = new ArrayList(); → Non-synchronized.

List L = Collections.synchronizedList (aL); → Synchronized.

→ Similarly we can get Synchronized version of Set, Map Objects by using the following methods of Collection class:

P S Set synchronizedSet (Set s);
P S Set synchronizedMap (Map m);

LinkedList

* Underlying data structure - doubly linked list
* Insertion order preserved.
* Duplicate Allowed.
* Heterogenous elements allowed.
* null insertion possible.

→ Linkedlist Implements Serializable and clonable interfaces but not RandomAccess Interface.
→ Linked List is the best choice if our frequent operation is insertion or deletion in the middle.
→ Worst choice if retrieval is required.

Page No.
Date

* Methods → To support stack and queues.

Void addFirst(object o)
Void addLast(object o)
Object getfirst()
Object getLast()
Object removefirst()
Object removeLast()

Constructors :-

(1) LinkedList L = new LinkedList();
creates an empty LL object

(2) LinkedList l = new LinkedList(Collection c).
creates an equivalent LL object of given collection.

demo

```
import java.util.*;
class demo {
    ps v m(String[] args)
    {
        linkedlist l = new linkedlist();
        l.add(1);
        l.add("Rahul");
        l.add(null);
        l.set(0, "One");
        l.add(1, "Rahul Jha");
        l.removeLast();
        l.addfirst("Durga");
```

Vector

(1) Underlying ds → Resizable/Growable Array
(2) Duplicates Allowed.
(3) Insertion Order Preserved.
(4) null insertion possible.
(5) Heterogenous element Allowed.
(6) Implement Serializable & clonable
(7) Also Implements RandomAccess
(8) Most of Methods are synchronized.
(9) Vector object is Thread-safe.
(10) Best choice if frequent operation is retrieval.
(11) Also known as legacy class.

Methods

addElement

for adding objects.

add (object o) — from Collection.
add (int index, object o) — from List
addElement(Object o). — from Vector

for Removing

Remove(object o) — C
removeElement(Oo) — ✓
remove (int index) — L
RemoveElementAt (int index) — ✓
clear() — C
removeAllElements() — ✓.

To Retrieve/Access

Object get (int idx) — C
object ElementAt(int idx) — ✓
O firstElementAt()
O lastElementAt().

Other

int size()
int capacity()
Enumeration Elements().

Constructors

(1) Vector v = new Vector();
// Default Initial Capacity = 10 with empty object
// New Capacity = 2* old Capacity

(2) Vector v = new Vector(int initialCapacity);
// creates a vector with Capacity = initialCapacity

(3) Vector v = new Vector(int IC, int IncrementalCapacity);
initialCapacity

(4) Vector v = new Vector(Collection c).

Stack

* child class of Vector.
* Last In first Out
*

Constructor

Stack s = new Stack();

Methods

(i) push (object o)
(ii) pop ()
(iii) peek()
(iv) ~~isEmpty~~ empty()
(v) search(object o) - return offset (Position from top)
(vi)

Page No.
Date

demo program

```
class StackDemo
{
    P s v m( String[] args)
    {
        Stack s = new Stack();
        s.push("A");
        s.push("B");
        s.push("c");
        SOP(s);   // [A, B, C]  Insertion Order Preserved.
        SOP(s.search("A"));  // 3
        SOP(s.search("z"));  // -1
```

C
B
A

Cursors in Java

* Cursor is used to retrieve element from collection one by one.

* There are three types of Cursors:-
  (i) Enumeration.
  (ii) Iterator.
  (iii) ListIterator.

* Enumeration

* Introduced in 1.0 version
* We can create Enumeration Objects by Using elements() method of Vector class.

```
Public Enumeration elements();
```

ex:-

Enumeration e = v.elements();
where v → vector object reference.

* Methods
(i) public boolean hasMoreElements();
(ii) public Object nextElement();

```
Vector v = new Vector();
for( int i = 0 to 10)
        v.addElement(i);
sop(v); // [0, -- 10]
Enumeration e = v.elements;
while (e.hasMoreElements)
  {
      Integer I = (Integer) e.nextElement();
      sop(I);
  }
```

2. Iterator

* Universal Cursor.
* Both read and ~~write~~ Remove operation.
*

Method

public Iterator iterator();

Iterator Itr = c.iterator();
↓
Any Collection Object.

Page No.
Date

Methods

(i) public boolean hasNext();
(ii) public Object next();
(iii) public void remove();

```
AL l = new AL;
for( i=0 to 10)
    l.add(i);
sop(l);
Iterator Itr= l.iterator();
while(itr.hasNext())
  {
    Integer i = (Integer) Itr.next()
    sop(I);
    itr.remove(i);
  }.
```

Limitation of Iterator

(i) single direction Cursor. (only forward)
(ii) Either read or remove only. ~~But Replace Not there.~~
(iii) Replace ✗
addition of new object ✗

ListIterator

Iterator
↓
ListIterator.

(I) Bidirectional. cursor.
(II) Read ✓
Remove ✓
Replace ✓
Addition of new object ✓

Syntax:

public ListIterator listIterator()

ex:-

ListIterator itr = l.listIterator
↳ Any List object.

Methods

p b hasNext()
p O next()
~~p v remove()~~
p int nextIndex()
} forward.

p b hasPrevious()
p O previous()
p I previousIndex()
} Backward.

p v remove
p v ~~replace~~ set(Object new)
p v add(Object new)

Property	Enu	Itr	ltr
(i) Applicable for	Vector	Any Collection	List
(ii) Movement	forward	forward	Bidirectional
(iii) Accessability	Read	Read Remove	Read Remove Replace, add
(iv) How to Get it?	elements()	iterator()	listIterator()
(v) Methods	hasMoreElement ~~hasNextElement~~ nextElement	hasNext() next() remove	replace() add() Total (9)
(vi) is it legacy	yes 1.0V	No 1.2V	No. 1.2V

Implementation classes of cursors.

We are not creating Enumeration object

```
Vector()
{
    elements()
    {
        class _ implements Enumeration
        {   }
        return this.object;
    }
}
```

for Enumeration
Vector $ 1
Vector $ Itr
Vector $ ListItr.