What is Collection ?

Ans – if we want represent individual group of objects in single entity we use collection

In general   
Collection interface considered as root interface of collection framework

9 Key features

1. Collection(v1.2)
2. List(1.2)
3. Set
4. SortedSet
5. NavigableSet
6. Queue
7. Map
8. SortedMap
9. NavigableMap

There is no concrete class which is implement Collection directly .

* Difference between Collection and Collections

1. Collection is a interface
2. Collections is a utility class present java.util package to define several utility method for Collection object Ex Sorting, Searching etc.

List interface (v1.2)

* Implemented class of List interface
* Array List(1.2)
* LinkedList(v1.2)
* Vector (v1.0)
* Stack(v1.0)
* Vector and Stack class are also called legacy class because which came into 1.0 version
* ArrayList and LinkedList and Vector Classes implement List interface
* Stack extends Vector class

Note : In 1.2 version vector and stack class are updated to implement List interface

It is the child interface of collection

If we want represent a group of individual object as single entity where

* Duplicates are allowed
* Insertion order must be preserved then we should go for List.

Set Interface (v1.2) :

* it is the child interface of Collection.
* if we want represent individual group of single entity we use collection
* duplicate are not allowed
* insertion order not preserved

Implement Class

* HashSet(v1.2)
* LinkedHashSet(v1.4)

SortedSet (v1.2)

* It is the child interface of set
* duplicate are not allowed
* all object should be inserted in some sorting order

NavigableSet interface (v1.6)

* it is the child interface of SortedSet interface
* it contains several method for navigation purpose

TreeSet (v1.2) is a class which implement NavigableSet interface

Queue interface (v1.5)

* it is the child of collection
* if we want represent a group of individual objects prior to processing the we should go for queue
* usually queue follow FIFO order but based on our requirement we can implement our own priority order also Ex: before sending a mail all mail id’s we have to store in some data structure I which order we added mail id’s In the same order only mail should be delivered for this requirement it is the best choice.
* Implemented classes
* Priority Queue
* BlockingQueue

Priority BlockingQueue

LinkedBlockingQueue

Note: Priority BlockingQueue, LinkedBlockingQueue are classes extends BlockingQueue.

All the above interfaces (Collection, List, SortedSet, Navigable and Queue) meant for representing a group of individual objects

Map(v1.2): If we want to represents a group of object as key value pairs then we should go for Map

* Map is not a child interface of Collection
* If we if we want represent a group of individual objects key value pairs then we should go for map
* Methods are
  + Set KeySet(); return only keys keys cannot be duplicated that’s why used Set
  + Collection values() : value can be duplicated that’s why used Collection
  + Set entrySet();return entry entry cannot be duplicated that’s why used Set
  + Data store in set is entry type then we have typecast set to Map.Entry to get entry

Ex:

|  |  |
| --- | --- |
| **Key** | **Value** |
| 102 | Amir |
| 103 | Ayan |
| 104 | Shahid |
| 105 | Anil |

* Both key and value objects only
* Duplicate keys are not allowed but values can be duplicated

Implementation classes of Map

* HashMap(v1.2)

LinkedHashMap (v1.4)

* WeakHashMap(v1.2)
* IdentityHashMap(v1.4)

Dictionary (Abstract class) (v1.0) these are also legacy classes

* Dictionary(AC)
* Hashtable
* Properties

**SortedMap**:

* It is the child interface of map
* TreeMap is a impl class
* If we want to represent a group of key value pair according to some sorting order of Key’s
* In SortedMap Sorting should be based on key but not based on value

**NavigableMap**:

* It is the child interface of map
* It defines several methods for navigation purposes
* TreeMap is the only Class which implement NavigableMap
* HashMap:
* Is a class of map
* Insertion order not preserved
* based on hashcode

1. HashMap is non synchronized. It is not-thread safe and can’t be shared between many threads without proper synchronization code whereas Hashtable is synchronized. It is thread-safe and can be shared with many threads.  
   2. HashMap allows one null key and multiple null values whereas Hashtable doesn’t allow any null key or value.

* Hahtable:
* Best choice for searching
* Thread safe
* Every ,method is synchronized
* Underline data structure is hashcode based on key
* Insertion order is not maintain
* Properties: if the requirements is change details frequently Ex: our requirement is change the password every 3 monthsso we should go for properties it is not recommended to hardcode the logic in our program.
* In normal map(like hashmap hashtable, treemap key and value can be anytype but in the case of properties key and value should be string type

**Cursor:**

If you want get object one by one from collection we use Cursors Concept

* Enumeration

Limitation of Enumeration:

* + - We can apply Enumeration concept only for legacy class
    - and it is not a universal curser
    - by using enumeration we can get only read access
    - we can’t perform remove operation
    - to overcome above limitation we should go for iterator
    - Method’s 1 hasnextelement() 2 nextelement()
* Iterator
  + - We can apply iterator concept for any collection object
    - It is universal cursor
    - By using iterator we can perform both read and remove operation’s
    - Method’s 1. Boolean hasNext() 2 Object next() 3 void remove()
    - Limitation
    - By using Enumeretion and iterator we can towards forward
    - Only we can’t move backward
    - These are single directional cursor
    - By using iterator we can perform only read and remove opreation
    - And we can’t perform replacement of additional new object
    - To overcome above limitation we should go for list iterator
* List Iterator
  + list iterator we can move either forward direction or to the backward direction and hence it is bidirectional cursor
* we can perform replacement and addition of new objects
* it’s a child interface of Iterator and hence all method available in Iterator by default available in listiterator
* list iterator defines the following nine methods

1. Boolean hasNext()
2. Object next()
3. Int nextIndex()
4. Boolean hasPrevious()
5. Object previous()
6. Int previousIndex()
7. Void remove()
8. Void add(Object o)
9. Void set(Object o)

**Utility Class:**

* Collections
* Arrays

**Sorting:**

If you want default natural sorting Sorting:

* Comparable interface

If you want customize sorting

* Comparator interface
* Collection: if you want represent ……
* Collection interface defines most common method which are applicable for any collection object these method are inside in Collection
* Add();
* addAll()
* remove() ->
* in Collection remove method match object value then remove object
* whereas List remove method considered as index

Ex: remove(5);

* + Collection will match value if found then remove
  + List find index if found then remove
* removeAll()
* clear()
* retainAll()
* contain()
* containAll()
* isEmpty()
* size()
* toArray() return object array
* iterator() return iterator object

Note: there is no concrete class which implements collection interface directly

**List interface:**

* If we want represent a group of individual object as single entity where
* Duplicates are allowed
* Insertion order must be preserved then we should go for List.
* Histroginus object are allowed on all collection except (TreeSet , TreeMap)
* Histroginus means different type of object
* Underline data structure is resizable array or growable array

**ArrayList Constructor**

* Default initials capacity of 10(not confirmed there is no method to check capacity)
* Assign manual size new ArrayList( int intialcapacity)
* Create an equalent ArrayList object it is use to convert another object to List ex: treestet to ArrayList . List t = new ArrayList(Collection c)
* New Capacity = (Current Capacity \*3/2 +1)

Methods to be practice:

* Add() :on 4th index of object in 2 object collection
* Get()
* Remove() from unadded index ;
* Listiterator()

Usually we can use collections to hold and transfer objects from one location to another location {container} to provide support for this requirement every collection class by default implements serielizable and clonable interfaces

* ArrayList and vector classes implements random-access interfaces also so that any random element we access with same speed

**Random-access** **interface**: present in java.util package and it does not contain methods it’s a marker interface. Where required ability will be provided by jvm.

**Difference between ArrayList and vector**

|  |  |
| --- | --- |
| it is non synchronized | every method is synchronized |
| Multiple thread can access no thread safety | Thread safe |
| It’s fast | It’s slow |
| 1.2v it’s not legacy | 1.0 it’s legacy |

**Difference between ArrayList and LinkedList**

|  |  |
| --- | --- |
| **Arraylist is best choice for retrieval operation** | **Best choice in insert or delete data in middle** |
| **Worst choice in insert or delete data in middle** | **Worst choice for retrieval operation** |
| **Arraylist stored elements in consecutive memory location and retrieval will become easy** | **Linkedlist is not store elements in consecutive memory location retrieval operation become difficult** |
| **ArrayList underline data structure is follow Array concept** | **LinkedList underline data structure is follow double-link-list concept** |

* How to get synchronized object of ArrayList.
* By default is non synchronized but we can get synchronized version of ArrayList object by using syncronizedList() of collections class

List list = new ArrayList();

List synList = Collections.syncronizedList()

Similar we can get synchronized version of set and map object by using these method

* Collections.syncronizedSet()
* Collections.syncronizedMap()

**LinkedList**:

* The underline data structure is doublelink list
* Insertion order is preserved
* Duplicate objects are allowed
* Hitrogionius object allow
* Null insertion is possible
* Linked list impl serielizable and clonable interface but not randow access interface
* Linked is the best choice if our frequent operation is insertion an deletion in the middile

LinkedList Constructor

* LinkedList l = new LinkedList() ……Empty LinkedList create
* LimkedList l = new LinkedList(Collection)…given collection convert to LinkedList

LinkedList class specific methods

Usually we can link list to develop stack and queues to provide support for this requirement linklist defines the following specific methods

* Void addFirst(Object o)
* Void addLast(Object o)
* Object getFirst()
* Object getLast()
* Object removeFirst()
* ObjectremoveLast()

Vector class:

* Underline data structure is resizable array and growable array
* Insertion order is preserved
* Duplicates are allowed
* Hystroginus object are allowed
* Null insertion is allowed
* Serielizable and clonable and random-access interface
* Every method is present in vector is synchronized and hence victor object is thread safe

**Constructor**:

1. Vector v = new Vector()

create empty object with initials capacity 10 once vector reaches max capacity then a new vector object will be created with new capacity = current capacity\*2

1. Vector v = new Vector(int initial-capacity)

Creates an empty vector object with specified initials capacity

1. Vector v = new Vector(int initials-capacity, int incremental-capacity)
2. Vector v = new Vector(Collection)

Vector Method is big

* addelement()
* removeelement()

**Stack**:

It is the child class of vector

It is the specially design the class for last in first-out(LIFO)

How many Method are available:

Push() add object

Pop() return top of the object last inserted

Peek() return top of the stack not rewmove

Empty() to check epty stack

Search(Object v) it will return (offset) if found else -1

Offset – index of element from the top of the stack index and offset are difference

Offset index

[1] C [ 2]

[2] B [1]

[3] A [ 0]

Set: child interface of collection set does not have any new method only available collection method set is use

Implements class

* Hashset(1.2)
  + LinkedHashSet(1.4)
* SortedSet( I v1.2)
  + NavigableSet(I v1.6)
  + TreeSet(1.2)
* **HashSet**
* Underline data structure hash table
* Duplicate objects are not allowed
* Insertion order is not preserved
* It is based on hash code of object
* Null insertion allowed(only once)
* Hystrogionious object allowed
* Impments serielizable and clonable but not random access
* Hashset is the best choice if our frequent opretion is search opreatio

**Note**: in HashSet duplicate are not allowed if we are trying to insert duplicate the we won’t get and compile time or runtime any error add simply returns false

**Constructor**

* HashSet l = new HashSet()

Creates an emty hashset object with default initials capacity 16

Default fill ratio 0.75

* HashSet h = new HashSet(int initials capacity)
* HashSet h = new HashSet(int initials capacity,double fillRatio)
* HashSet h = new HashSet(Collection c)

Fill Ratio – after filling .75 ratio a new hashset object will be created

Load-factor – after filling how much a new hashset object will bew created

LinkedHashSet:

* It is the class of hashset
* It is exactly same as hashset including constructor and method

Difference

|  |  |
| --- | --- |
| HashSet(v1.2) | LinkedHashSet(v1.4) |
| Underline DataStructure is hashTable | Underline data structure is combination of Linkedlist and HashTable |
| Insertion order is not preserved | Insertion order is preserved |
| Duplicate not allowed | Duplicate not allowed also |
|  | It is use to develop cache based application where duplicate are not allowed and insertion order should be preserved |

**SortedSet(i):**

* Sorted is the child interface of set
* If we want represent a group of individual objects according to some sorting order without duplicate then we should go for SortedSet.

Methods

* First()
* Last()
* headset(Object o) : less then object
* tailSet(Object o) : >= object
* subSet(Object obj1,Object obj2)>=obj1 and <obj2
* Comparator();

**TreeSet(c):**

* Underline data structure is balance tree
* Duplicate is not allow
* Insertion order not preserved
* Hydrogenous objects are not allowed otherwise runtime exception classcastexception
* Null insertion is allowed only once
* All objects will be inserted based on some sorting order it may default natural sorting order or customized sorting order
* Null pointer check if there is only one null object it will allow else not working till 1.6 after 1.6 it will no allowing

**Constructor**:

* TreeSet t = new TreeSet(); - insertion data in default natural sorting orde
* TreeSet t = new TreeSet(Comparator c) creates empty comparator object where will be inserted in customize sorting specified comparator object
* TreeSet t = new TreeSet(Collection c)
* TreeSet t = new TreeSet(SortedSet s)

**Important Note :**

* if we are depending on default natural order compulsory object should be homogeneous and comparable otherwise throw CCE(Class Cass Exception)
* and object is set to be comparable if only if corresponding class implement comparable
* String and all wrapper classes are implements comparable
* Stringbuffer class does not impl comparable
* If you compare with null value nullpointer exception will be throw

**Comparable(i)**

It is available in java.lang package and it contains only one method compareTo()

* Return –value if obj1 has to come before obj2
* Return +veobj1 has to come after obj2
* Return 0 if obj1 and obj2 are equal
* JTA
* Transaction is a group of related operations as a single unit where the operation will be executed on the principal of either all or none