**Array**

Array is a linear data structure which store homogenous elements in a continuous memory allocation.

Advantage

Array is Index-based so we can access any element of array in O(1) time complexity that why we should use array when we have more retrieval operation.

Drawback of array----Fixed size, homogenous element.

**Why we need collection?**

- To overcome the disadvantage of array like

-fixed size, collection are dynamic in size.

-array store only homogenous element, collection can store different

type of object.

**What is Collection?**

* In general terms, It is a way to represent group of individual objects as a single unit. (reference of each object is going to store in collection).
* In java, Collection is an root interface which has its sub-interfaces(like List,Set,Queue) and various implementation classes(data structure), introduced in java 1.2 and present in java.util package.
* Collection interface tells what’s its implementation classes must do(standardization), not how(abstract methods).

**Collection Framework**

* In Java Collection Framework we will find ready-made data Structure which are pre-compiled and ready to use, we can store and manipulate the objects inside them.
* Collection framework = Combination of

Collection interface, its sub-interfaces and its

implementation class +

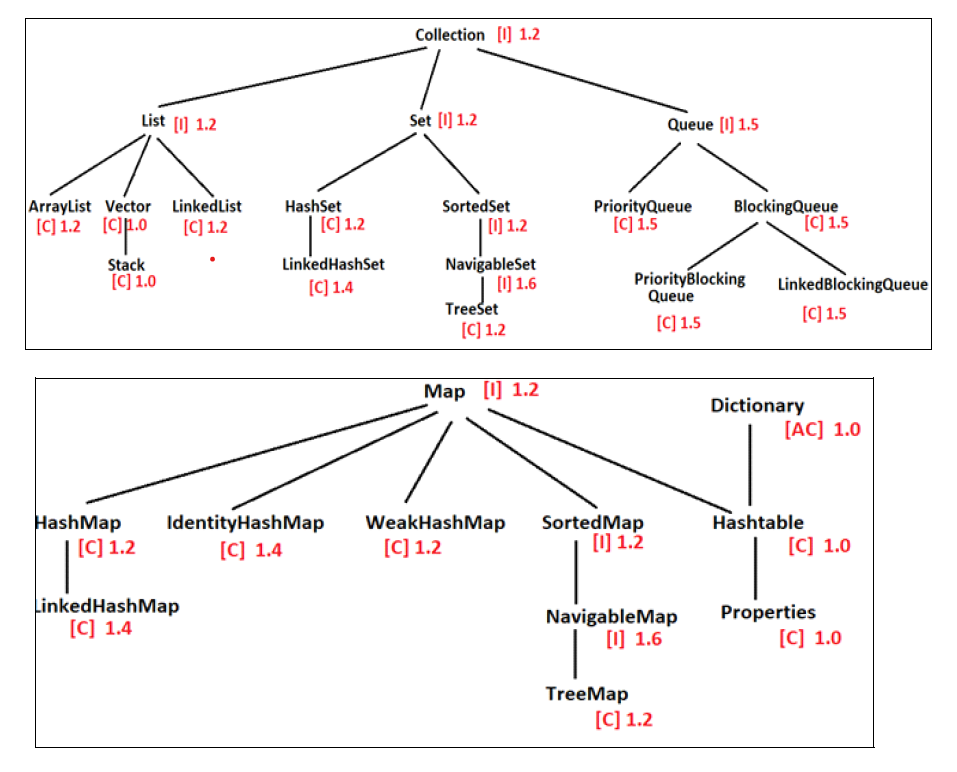
Map interface, its sub-interface and its

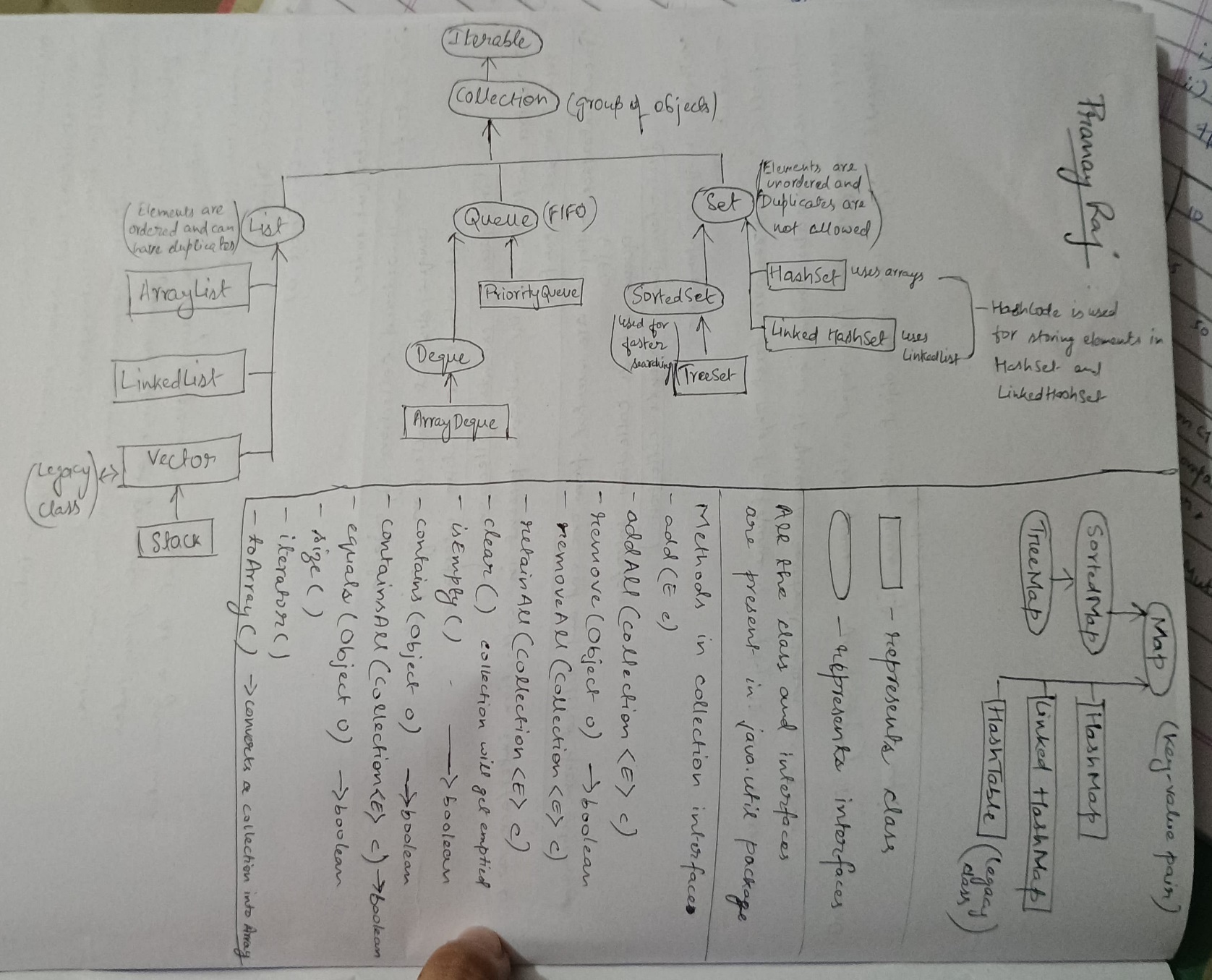
implementation class +

Collections Utility class.

Difference between Collection and Collections

|  |  |
| --- | --- |
| Collection | Collections |
| It is an interface | it is a utility class |
| It contains abstract methods | it contains only static methods |
| Collection method are used to store and perform manipulation on objects.  ex- add(),remove() | -Collections methods are used  for searching and sorting.  ex- sort(),reverse(),binarySearch() |



5

**Adding Elements in collection**

Collection can store only objects.

When we add any object in collection, without using generics then every object in collection will be upcasted to Object Class type and then it will be stored.

For ex

* primitive type values(10,2.5,true,”Major Harshan”, ‘c’ ) when added to Collection 🡪 first autoboxing happens(primitive type values converted to wrapper class object) and then upcasted to Object class type.
* User-defined objects and String 🡪 directly upcasted to Object class type.
* As object is now upcasted to Object class type, perform downcasting to access sub-class specific properties.
* When generics are used, object’s are not upcasted to Object class type(Reason- because now collection will store only same type of objects, collection become homogeneous). So no need of upcasting,downcasting.

**Note –** All classes implementing collection interface or map interface toString(), equals(), hashCode() are overridden.

Note - Learn Collection methods from notebook

**Generics**

Whatever type you mentioned in generics only that type objects will be stored in collection, collection becomes homogenous. This gives one advantage no need of upcasting and downcasting, as all are of same type.

**List Interface**

-List is implemented by ArrayList, Vector and LinkedList.

-List is index-based.

-List follow order of insertion.

-List allow duplicates value.

-List allow multiple null values.

Note- Learn List methods from notebook

**ArrayList & LinkedList**

|  |  |
| --- | --- |
| ArrayList | LinkedList |
| 1. It implements only List interface | It implements both List as well as Queue interface |
| 1. Random access is faster here.   O(1) time complexity | Random access is slower here.O(n) time complexity(because it always start from the first node). |
| 1. ArrayList is preferable when we have more retrievable operations. | LinkedList is preferable when we have more insertion and deletion operations. |
| 1. Default capacity of ArrayList is 10. | Default capacity of LinkedList is 0.(empty LinkedList is created) |
| 1. ArrayList grows by 60% of its capacity. | LinkedList grows by 1,when we add new element only then a new node will be created. |
| 1. ArrayList underlying data structure is array. | LinkedList is built using doubly LinkedList data structure. |

-Both follow order of insertion.

-Both store duplicates values.

-multiple null values can be stored in both.

Note – Study all programs from notebook & eclipse.

**Vector**

* Vector implements List interface and superclass of Stack.
* Vector is a legacy class.
* Vector is synchronised.
* Vector object can be accessed by only one thread at a time.
* Vector object is thread-safe.
* Vector default capacity is 10.
* Vector grows by twice of its size.
* Vector underlying data-structure is Array.

|  |  |
| --- | --- |
| Vector | ArrayList |
| 1. Vector is a legacy class(introduced in JAVA 1.0) | ArrayList is not a legacy class(introduced in JAVA 1.2) |
| 1. Vector grows by twice of its capacity. | ArrayList grows by 60% of its capacity. |
| 1. Vector class is synchronised | ArrayList class is not synchronised |
| 1. Vector class is synchronised that why it is Slower. | ArrayList class is not synchronised that why it is Faster. |
| 1. Its object is thread-safe | Its object is not thread-safe |

**LinkedList**

**Stack**

1. Stack is a data structure which store and process objects in Last In First Out format (means data which has entered last will be the first to come out).
2. Stack has two ends – top, bottom.
3. Both insertion and deletion happens from top only.
4. Insertion operation known as push.
5. Removal operation known as pop.

Real-time example of Stack

1. Browser back button
2. Function call in programming.
3. Undo/redo
4. Bangles in hand of lady.

Note-Learn Methods of Stack

push(object) , pop() , peek() , size()

**Q-** Difference between size and capacity ?

Ans - Size means the number of objects currently stored in the

collection.

Capacity means the maximum numberof objects the

collection can hold before resizing.

For ex -When ArrayList is declared its size is 0 as it is storing no objects but its capacity is 10.

**Comparable Interface**

What?

* Comparable is an interface which is present in java.lang package and have only one abstract method

public int compareTo(T arg){  
 }

T is a placeholder: It stands for a specific type, defined when a class implements Comparable<T>. For example, if a class is declared as class Person implements Comparable<Person>, then T becomes Person, and the method signature int compareTo(T o); effectively becomes int compareTo(Person o);

* This method return three int values.

Negative integer-if current object is less than the passed object.

Positive Integer -if current object is greater than the passed object

Zero -if both this object is equal to other object.

* The class which implements this comparable interface that class’s objects are called mutually comparable.

Why we need this ?

For sorting(or comparing) the objects of same class.

How sorting happens?

* The objects which we want to sort, that object’ s class must implements Comparable interface and override compareTo() method.

(See the program in eclipse for better understanding)

(just for understanding)

All wrapper classes and String class implements comparable interface that why all wrapper class objects are mutually comparable.

User defined class must implement comparable interface or declare a class which implements comparator interface, if we want to sort the objects of user-defined class.

TreeSet, TreeMap store elements in sorted order and PriorityQueue store smallest element at head, so for all this, sorting is required (for sorting comparable or comparator is needed) and sorting always happen between same type of objects.So thats why TreeSet, TreeMap and PriorityQueue are homogenous collection.

Note- implement comparable interface always with generics

Reason

Reduce Errors – As without generics comparable interface can hold only Object class type elements so when you call compareTo() method and pass your object in parameter, object will be upcasted to Object type and then you have to receive that object, in compareTo() method’s parameter in Object type reference variable and after receiving you have to downcast it, to access sub-class specific properties and then compare with the current object. So if programmer forgot to downcast, it will be error and this upcasting and downcasting will take time also.

And if you pass some other class object in parameter(instead of employee, you passed Student), so at the time of downcasting it to Employee type, ClassCastException will come.

**Comparator Interface**

What ?

Comparator is an interface which is present in java.util package and have abstract method called

public int compare(T o1 , T o2){

}

This method return Three int value

Positive value - current object is greater than passed object

Negative value - current object is smaller than passed object

Zero value - current object and passed object both are equal.

Why we need comparator?

Allows us to define multiple custom sorting outside of the class, without modifying the class itself. For example, you may want to sort Person objects by name, age, or height in different parts of your application.

1. Multiple Sorting Orders for the Same Class:

If a class has multiple meaningful ways to be sorted, use Comparator to define each of them. For instance, if Person can be sorted by age, name, and height, each can be defined as a separate Comparator, allowing you to use whichever sorting order you need at runtime.

1. Sorting Third-Party Classes or Classes You Can’t Modify:

If the class you’re working with doesn’t implement Comparable (like third-party classes or library classes), you can’t change its natural ordering. Instead, you can define custom Comparators to sort instances of that class without modifying it.

How ?

* Make a separate class which implement comparator interface and override compare() method.
* Then make object of that comparator class and then create object of collection and pass the objectReference of the comparator class in parameter. Then that collection, sort the objects of that class, on basis of that comparator class.

(See the program in eclipse for better understanding)

Comparable and Comparator

|  |  |
| --- | --- |
| Comparable Interface | Comparator Interface |
| 1. Present in java.lang package | Present in java.util package |
| 1. Comparable interface has only one abstract method called compareTo(T o) | Comparator interface has one abstract method called compare(T o1 , T o2) |
| 1. Comparable provides a single sorting sequence.   Reason- Because we can override compareTo() method only once in a class. | The Comparator provides multiple sorting sequences.  Reason – because we can make n  number of comparator classes. |
| 1. Comparable affects the original class, i.e., the actual class is modified. | Comparator doesn't affect the original class, i.e., the actual class is not modified. |

**SET**

1. Set is an sub-interface of collection but implemented by HashSet, LinkedHashSet, TreeSet.
2. Set is not index based. So we cannot access elements index-wise.
3. Set do not have its own specific Method.
4. Order of insertion is not followed by Set. (only LinkedHashSet follow order of insertion).
5. Set is a collection of unique objects, does not allow duplicates.
6. Set allow only one null value(only TreeSet do not allow null Value).

Q- How a Set do not allow duplicate element?

1. Whenever we add object into Set, add() will internally call equals() and compare current object with every other object present in Set. And if equals() returns false while comparing with every other object only then the current object is added into Set. And if equals() returns true this means object is already present in Set then the current object will not be added into Set.

equals() method is overridden in set it compares value instead of address.

Note – If we try to store duplicate elements in Set it will not throw any error or exception, it will simply not store duplicate element.

**HashSet**

1. HashSet implements Set interface so all the properties of Set comes in HashSet.
2. HashSet Underlying data structure is HashMap. So its default capacity, load factor, by how much it will grows is same as HashMap.

(just for understanding)

When we add Wrapper class object and String class object in HashSet, add() can easily calls equals() method and check.(because in all wrapper-class and String class- hashCode(), equals() are overridden)

To add user-defined object in set, it is compulsory to override hashCode() method and equals() method in user-defined class. So that Set can check the object, based on one attribute.

The hashCode() must return the attribute in an integer format which helps the Set to maintain unique records.

And as toString() is also overridden in all wrapper classes and String class, so as good practice override toString() method whenever you create user-defined class).

See program in notebook & in eclipse for better understanding.

**LinkedHashSet**

1. extends HashSet and implemets Set interface so all properties of set interface comes.
2. Order of insertion- follow (this is the only difference)
3. Underliying data-structure is HashMap and Doubly Linked List.
4. So LinkedHashSet default capacity, load factor and by how much it grows is same as HashMap.

**TreeSet**

1. TreeSet is an implementation class of NavigableSet which is child interface of SortedSet which is child interface of Set.
2. TreeSet does not follow order of insertion(as it stores element in sorted order(ascending) ).

(Sorting is only possible when all objects are of same class type that

makes TreeSet homogenous collection and for sorting either

comparable interface or comparater interface is required)

1. Does not allow duplicate elements.
2. Do not allow null value.

(to maintain sorted order, TreeSet internally calls compareTo() method in case of comparable interface or compare() method in case of comparator using object, so if you insert null value in TreeSet and you try to call any method using null then it will give you null pointer exception so that why you can’t store null value in TreeSet)

1. TreeSet initial capacity is 0.
2. Underlying data structure of TreeSet is Red-Black Tree(binary search tree).
3. TreeSet has two overloaded constructor

- Constructor without parameter- Empty TreeSet is created to

arrange the elements based on comparable implementation.

- Constructor with comparator type parameter – Empty TreeSet is

created to sort the elements based on the comparator

implementation.

**Queue**

1. Queue is a data structure which store and process objects in First In First Out order.
2. Queue has 2 ends head & tail.
3. Insertion of object will be done from tail-end.
4. Deletion of object will be done from head-end.
5. Insertion operation is known as enqueue or offer.
6. Removal operation is known as poll.

Real Life Example of Queue

1. Request handling by Web Server – incoming request are queued and handled on first come first serve basis.
2. Play next feature in song playlist – songs are line in queued and first song in the playlist will be played first.(but underlying data-structure can be Linkedlist as insertion and removal of song in palylist can be efficiently done by LinkedList)

|  |
| --- |
| QUEUE |
| 1. Implemented by PriorityQueue, LinkedList, ArrayDeque |
| 1. Queue follow order of insertion i.e FIFO(except priority queue). |
| 1. Queue allow duplicates. |
| 1. Null elements are not allowed. |
| 1. Queue has its own specific methods |

Queue Methods learn from notebook

**PriorityQueue**

1. PriorityQueue is an implementation class of Queue Interface.
2. Priority Queue does not follow FIFO(first-in, first-out).
3. PriorityQueue stores smallest element at head, rest of the element will not be in sorted order(That why when we use poll operation we will get the smallest element, and when we use poll operation again we will get next smallest element because smallest element will be stored in the head always)

So for storing smallest element at head we have to use sorting this means all the objects in PriorityQueue should be of same class type which makes PriorityQueue homogenous collection.

And comparable interface or comparator interface is needed either one is needed to sort the objects.

4) It allows duplicates.

1. It does not allow null value(null pointer exception will come while calling compareTo() method or compare() method).
2. PriorityQueue underlying data-structure is binary heap data structure(specially min-heap)
3. Default capacity is 11. Grows by 50% of its current capacity.

Note- Study all programs from notebook and Eclipse

**ArrayDeque**

**MAP**

1. Map is a root interface(not sub-interface of Collection) which is implemented by HashMap, LinkedHashMap, TreeMap, HashTable classes.
2. Map is used to store key-value pair.
3. Key must be unique, values can be duplicated.
4. Using key we can access value.

Note – Learn Map methods from notebook

Study all programs from Notebook and eclipse

**HashMap**

1. HashMap implements Map Interface.
2. HashMap store elements in key-value pair.(key must be unique, values can be duplicated.)
3. HashMap does not follow order of insertion.
4. HashMap allow one null key, multiple null values are allowed.
5. HashMap underlying data-structure is array + linked-list.
6. HashMap by default capacity is 16, and it grows by twice of its current capacity and load factor is 75%(means when 75% of its current capacity is filled with object, HashMap will resize itself).
7. We can change the load factor of HashMap, with the help of parameterized constructor.
8. HashMap has 3 overloaded constructor

- HashMap m1 = new HashMap() ;

capacity is 16 and load factor is 0.75

- HashMap m1 = new HashMap(n) ;

capacity is n and load factor is 0.75

- HashMap m1 = new HashMap(n , lf) ;

capacity is n and load factor is lf

**Load Factor(or Filled Ratio)**

Load factor tells when a Collection should resize-itself.

For ex- By default the load factor is 0.75 in HashMap means when 75% of HashMap is filled with objects, then HashMap has to grow by twice of its current capacity.

**LinkedHashMap**

1. It is child class of HashMap.
2. It follow order of insertion means keys will be iterated in the same order they were added.
3. Its underlying data-structure is HashTable+LinkedList.

**TreeMap**

**HashTable**

1. HashTable implements map interface.
2. HashTable is a legacy class.
3. HashTable underlying data-structure Array+LinkedList.
4. HashMap and HashTable uses same data-structure so its default capacity and how much its grows all are same as HashMap but difference is

- HashTable does not allow null key and null value.

- HashTable methods are synchronised and its object is thread-

safe.

HashSet & HashMap

|  |  |
| --- | --- |
| HashMap | HashSet |
| 1. Implements Map interface | Implements Set interface |
| 1. Store key-value pair | Store only value |
| 1. Map can store duplicates   values, key must be unique. | Store unique values, duplicates are not allowed |
| 1. Multiple null values can be   Stored. | Only one null value can be stored |
| 1. HashMap underlying data structure is array,LinkedList,Red-Black tree. | HashSet underlying data-structure is HashMap. |

HashMap & Hashtable

|  |  |
| --- | --- |
| HashTable | HashMap |
| 1. HashTable is a legacy class | HashMap is not legacy class |
| 1. HashTable is synchronised | HashMap is not synchronised |
| 1. HashTable is synchronised, so its object is accessed by one-thread at a time. | HashMap is asynchronous, so its object is accessed by multiple threads at a time. |
| 1. HashTable object is thread-safe | HashMap object is not thread-safe |
| 1. HashTable do not allow null-value and null key | HashMap allow one null-value and multiple null keys |

Singleton class

Explain System.out.println()

Reverse arrayList --- two pointer approach

asList(T…a) method