Bootcamp Training

Java Spring AWS Frontend Training

SummitWorks™
GLOBAL SOLUTION ARCHITECTS

Authorized & published by Summitworks Technologies Inc

Agenda: Day -4

- Arrays and Collection framework
  - Collection Framework Hierarchy
  - List
    - Array List vs Linked List
  - Sorting using Arrays Class
  - Cursors
    - For each and Enumeration
    - Iterator and List Iterator
  - Vector
  - Set
  - Set vs List and importance of equals() and hashcode() methods
  - Hash Set, Linked Hash Set ,Sorted set and Tree Set
  - Comparable and Comparator interface
  - Generics and Sorting Collections

Collection Framework - Introduction

Drawbacks of Arrays

- Fixed size
- Accepting same type of elements [this can overcome with object array]
- No default data structure
- Unable to identify duplicate values
- Don’t have proper order of the elements

Object Array

- Since object is super class for all classes in java programming, so we can make a object array which can accept any object.

Introduction

- Utility means usefulness. Utility classes are those that help in creation of other classes.
- The *java.util* package contains utility classes like
  - Date,
  - Calendar,
  - Stack,
  - Linked List,
  - StringTokenizer, etc.

Drawbacks of object array

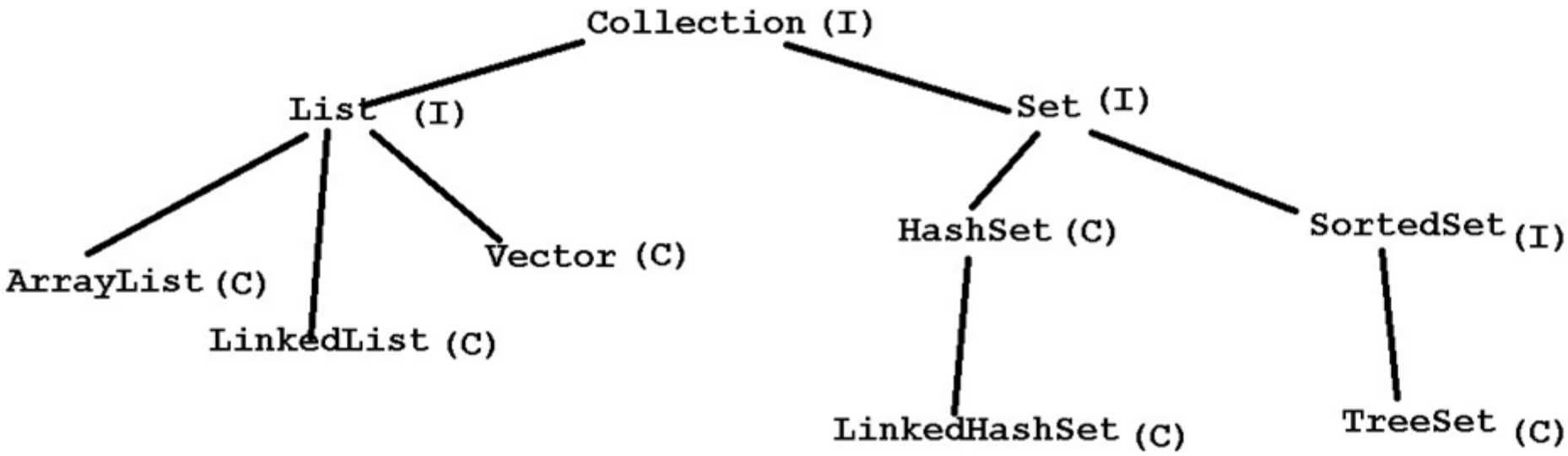
- Fixed size
- Unable to identify duplicate values
- No built in data structure

✓ to overcome these draw backs, sun introduce Collection framework

Collections

- **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.
- Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

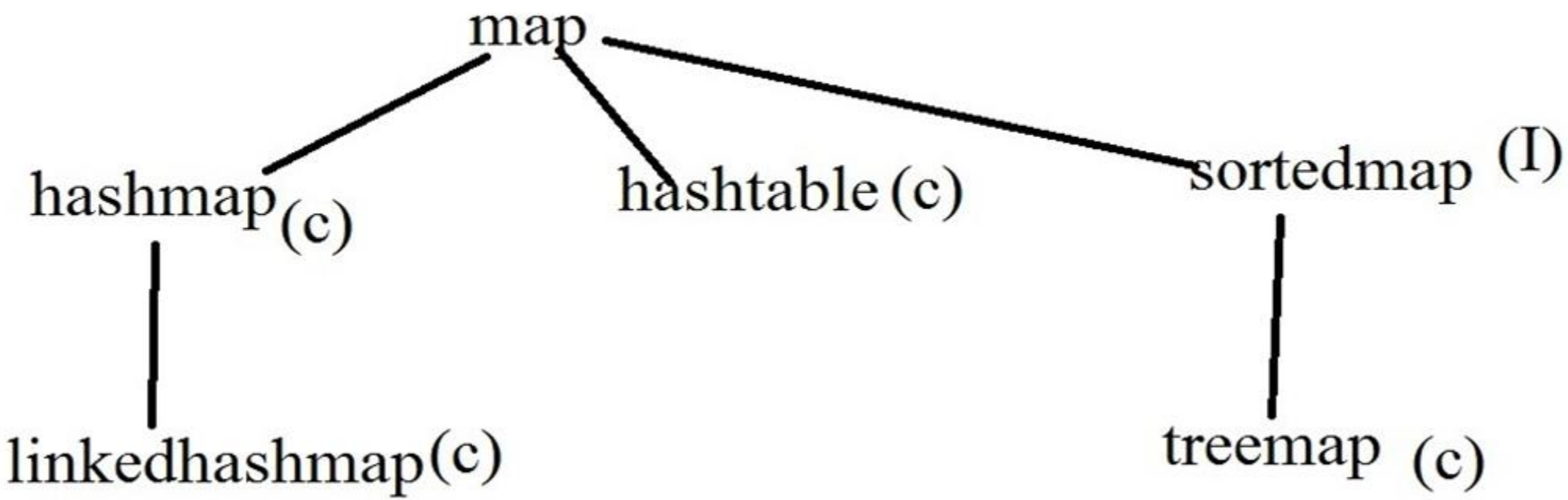
Collection Inheritance hierarchy



from java.util package

Map interface

Map Interface



Methods of Collection Interface

Method	Description
boolean add(E e)	Adds the specified element to the collection and returns true. It returns false only in case the collection does not accept duplicates (like Set).
boolean addAll (Collection<? extends E> c)	Adds all the elements in the specified collection to this collection and returns true if collection changed.
void clear()	Clears the collection by Removing all the elements from this collection.
boolean contains(Object o)	Returns true if this collection contains the specified object.
boolean containsAll (Collection<?> c)	Returns true if this collection contains all the elements in the specified collection.
boolean equals(Object o)	Compares the specified object with this collection for equality and returns true if equal.
int hashCode()	Returns the hash code value for the collection.
boolean isEmpty()	Returns true if this collection is empty.
Iterator<E> iterator()	Returns an iterator to iterate the elements in this collection.
boolean remove(Object o)	Removes a single instance of the specified element from this collection, if it is present.
boolean removeAll (Collection<?> c)	Removes all the elements of this collection that are also contained in the specified collection.
boolean retainAll (Collection<?> c)	Retains only the elements in this collection that are contained in the specified collection.
int size()	Returns the number of elements in this collection.
Object[]toArray()	Returns an array containing all the elements in this collection. The return type of array of object will be of type Object class.
<T> T[] toArray(T[] a)	Creates an array containing all the elements in this collection. The return type of the array of objects will be the according to the type of objects in the collection.

Methods of List interface

Method	Description
boolean addE(E e)	Appends the specified element to the list.
void add (int index, E element)	Inserts the specified element at the specified position in this list.
boolean addAll(Collection<? extends E> c)	Appends all the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.

List Interface

- Accepts heterogeneous elements
- Auto grow able size
- Order of display is insertion order
- Accepts both primitives and objects[because of auto boxing]
- Duplicate insertion is also possible

Methods of List interface

boolean addAll(int index, Collection<? extends E> c)	Inserts all of the elements in the specified collection into this list at the specified position
void clear()	Clear by Removing all the elements from this list.
boolean contains(Object o)	Returns true if this list contains the specified element.
boolean containsAll(Collection<?> c)	Returns true if this list contains all the elements of the specified collection.
boolean equals(Object o)	Compares the specified object with this list for equality.
E get(int index)	Returns the element at the specified position in this list.
int hashCode()	Returns the hash code value for this list.
int indexOf(Object o)	Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean isEmpty()	Returns true if this list contains no elements.
Iterator<E> iterator()	Returns an iterator over the elements in this list in proper sequence.
int lastIndexOf (Object o)	Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
ListIterator<E> listIterator()	Returns a list iterator over the elements in this list.
ListIterator<E> list Iterator(int index)	Returns a list iterator of the elements in this list, starting at the specified position in this list.
E removeint(int index)	Removes the element at the specified position in this list.
boolean removejava.lang. Object(Object o)	Removes the first occurrence of the specified element from this list, if it is present.
boolean removeAll(Collection<?> c)	Removes from this list all of its elements that are contained in the specified collection.
boolean retainAll(Collection<?> c)	Retains only the elements in this list that are contained in the specified collection.
E set(int index, E element)	Replaces the element at the specified position in this list with the specified element and returns the elements previously stored at the specified position.
int size()	Returns the number of elements in this list.
List<E>subList(intfrom Index, int toIndex)	Returns a sub list containing elements between the specified fromIndex, inclusive, and toIndex.
Object[]toArray()	Returns an array containing all of the elements in this collection. The return type of array of object will be of type Object class.
<T> T[] toArray(T[] a)	Creates an array containing all of the elements in this collection. The return type of the array of objects will be the according to the type of objects in the collection.

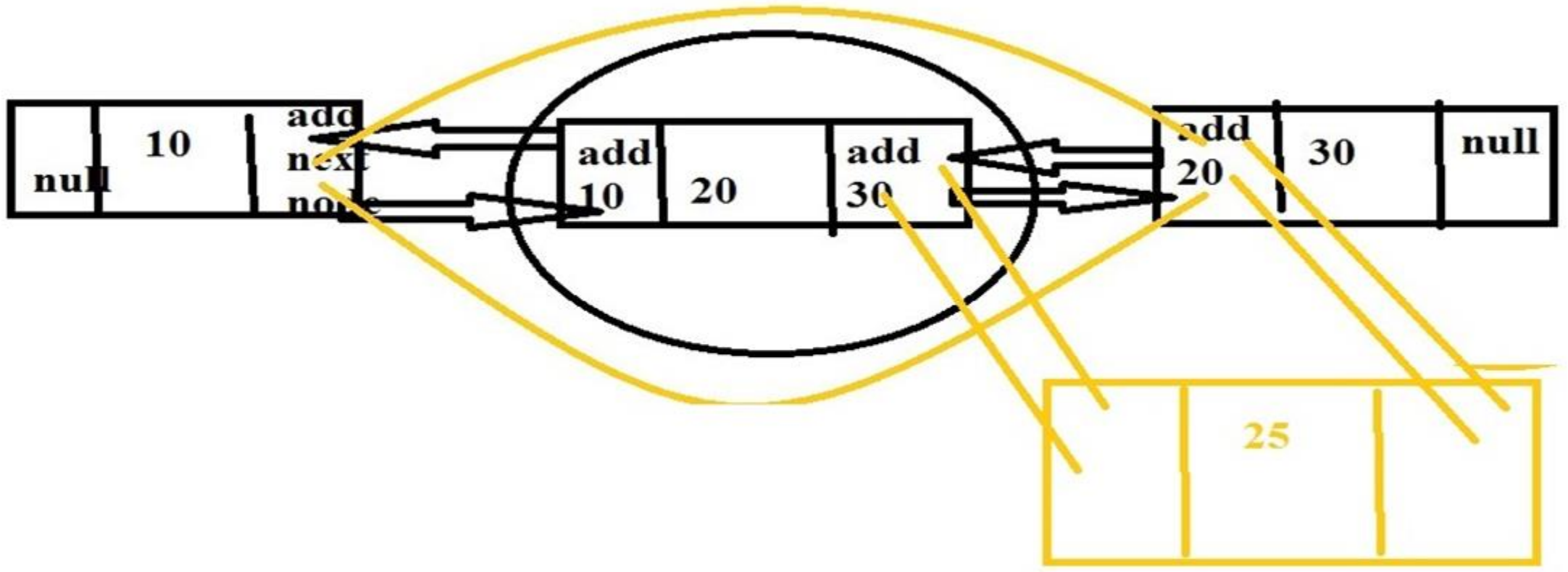
ArrayList

- Arraylist is suitable if our frequent operation is retrieval.
- If you do operations in the middle of the list, it need lot of shifts, so array list is not suitable if your frequent operation is insert/delete in the middle of the list

LinkedList

- Method of linkedlist
  - Addfirst(Object o)
  - addLast(object o)
  - getFirst()
  - getLast()
  - removeFirst()
  - removeLast()

LinkedList Implementation



Difference between ArrayList and LinkedList

ArrayList	LinkedList
1) ArrayList internally uses **dynamic array** to store the elements.	LinkedList internally uses **doubly linked list** 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.	Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory.
3) ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.

Use

- Linkedlist is best suitable if our frequent operation is doing operations in the middle of the list.

Vector Class

Methods

- void addElement(Object obj)
- Adds the specified component to the end of this vector, increasing its size by one.

- int capacity()
- Returns the current capacity of this vector.

- Object elementAt(int index)
- Returns the component at the specified index.

- void insertElementAt(Object obj, int index)
- Inserts the specified object as a component in this vector at the specified index.

Difference between vector and arraylist

- All methods in vector are synchronized
- All methods in arraylist are not synchronized

Stack

It’s a subclass of vector class

Methods

- Push()
- Peek()
- Pop()
- Search()

Cursors

There are three types of cursors in java programming

- 1. Enumeration
- 2. Iterator
- 3. Listiterator

Enumeration

- Methods:
  - hasMoreElement()
  - nextElement()

  It will support only legacy classes

  We can get read only acces, we cant modify and remove the objects.

Iterator

- Methods:
  - Boolean hasNext()
  - Object next()
  - remove()
- Object creation:
  Collectionobject.iterator()
- Drawbacks:

We can remove only and cant modify

List Iterator

- Methods:
  - ✓hasNext()
  - ✓hasPrevious()
  - ✓Next()
  - ✓Previous()
  - ✓Set(object)

- Object creation:

- Drawbacks: support only list implemented classes

Iterator vs Foreach In Java

- In for-each loop, we can’t modify collection, it will throw a ConcurrentModificationException on the other hand with iterator we can modify collection.
- Modifying a collection simply means removing an element or changing content of an item stored in the collection. This occurs because for-each loop implicitly creates an iterator but it is not exposed to the user thus we can’t modify the items in the collections

Generics

Before generics, we can store any type of objects in collection i.e. non-generic. Now generics, forces the java programmer to store specific type of objects.

There are mainly 3 advantages of generics. They are as follows:

1) Type-safety : We can hold only a single type of objects in generics. It doesn't allow to store other objects.

2) Type casting is not required: There is no need to typecast the object.

Cont..

Before Generics, we need to type cast.

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);//typecasting
```

After Generics, we don't need to typecast the object.

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);
```

Cont..

- 3. **Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.
- List<String> list = **new** ArrayList<String>();
- list.add("hello");
- list.add(32);//Compile Time Error

syntax

- Syntax to use generic collection

- ClassOrInterface<Type>
- Example to use Generics in java

- ArrayList<String>

Example

- Here, we are using the ArrayList class, but you can use any collection class such as ArrayList, LinkedList, HashSet, TreeSet, HashMap, Comparator etc.
- ArrayList<String> list=**new** ArrayList<String>();
- list.add("rahul");
- list.add("jai");
- //list.add(32);//compile time error
-
- String s=list.get(1);//type casting is not required
- System.out.println("element is: "+s);
-
- Iterator<String> itr=list.iterator();
- **while**(itr.hasNext()){
- System.out.println(itr.next());
- }

Using Map interface

```
import java.util.*;
class TestGenerics2{
public static void main(String args[]){
Map<Integer,String> map=new HashMap<Integer,String>();
map.put(1,"vijay");
map.put(4,"umesh");
map.put(2,"ankit");
//Now use Map.Entry for Set and Iterator
Set<Map.Entry<Integer,String>> set=map.entrySet();
Iterator<Map.Entry<Integer,String>> itr=set.iterator();
while(itr.hasNext()){
Map.Entry e=itr.next();//no need to typecast
System.out.println(e.getKey()+" "+e.getValue());
}
}}
```

Generic class

- A class that can refer to any type is known as generic class. Here, we are using **T** type parameter to create the generic class of specific type.

Syntax of generic class

Creating generic class:

- **class** MyGen<T>{
- T obj;
- **void** add(T obj){**this**.obj=obj;}
- T get(){**return** obj;}
- }

The T type indicates that it can refer to any type (like String, Integer, Employee etc.). The type you specify for the class, will be used to store and retrieve the data.

Example

```
class TestGenerics3{
public static void main(String args[]){
MyGen<Integer> m=new MyGen<Integer>();
m.add(2);
//m.add("vivek");//Compile time error
System.out.println(m.get());
}}
```

Type Parameters

- The type parameters naming conventions are important to learn generics thoroughly. The commonly type parameters are as follows:

- T – Type for GenericClasses
- E – Element for generic Methods
- K – Key keys of map interface
- V – Value value of map interface

Generic Method

- Like generic class, we can create generic method that can accept any type of argument.

Let’s see a simple example of java generic method to print array elements. We are using here **E** to denote the element.

Bounded Type Parameters

- There may be times when you'll want to restrict the kinds of types that are allowed to be passed to a type parameter. For example, a method that operates on numbers might only want to accept instances of Number or its subclasses. This is what bounded type parameters are for.
- To declare a bounded type parameter, list the type parameter's name, followed by the extends keyword, followed by its upper bound.

Set Interface

- Set is a collection that does not contain duplicate objects.
- This interface is implemented by an abstract class AbstractSet which implements some of the methods of Set interface.
  - The concrete classes, HashSet, EnumSet, etc., are subclasses of AbstractSet.
  - This interface is inherited by SortedSet interface to access the element in a sorted order (ascending).
  - The class TreeSet inherits the SortedSet interface and the AbstractSet class.

Java HashSet class

- Java HashSet class is used to create a collection that uses a hash table for storage.

The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing.**
- HashSet contains unique elements only.

What is Hashing

- Hashing in its simplest form, is a way to assigning a unique code for any variable/object after applying any formula/algorithm on its properties. A true Hashing function must follow this rule:

Hash function should return the same hash code each and every time, when function is applied on same or equal objects. In other words, two equal objects must produce same hash code consistently.

When we use hashing

- Hashing is designed to solve the problem of needing to **efficiently** find or store an item in a collection. For example, if we have a list of 10,000 words of English and we want to check if a given word is in the list, it would be inefficient to successively compare the word with all 10,000 items until we find a match. Hashing is a technique to make things more efficient by effectively narrowing down the search at the outset.
- Hashing means using some function or algorithm to map object data to some representative integer value. This so-called **hash code** (or simply **hash**) can then be used as a way to **narrow down our search when** looking for the item in the map.

Cont..

- Accepts heterogeneous objects
- Duplicate objects are now allowed
- Null insertion is possible, but only once
- Order is not preserved, order is based on hash codes

Methods of Set interface

Methods	Description
boolean add(E e)	Adds the specified element to this set if it is not already present and returns true, else false.
boolean addAll(Collection<? extends E> c)	Adds all the elements in the specified collection to this set if they are not already present, and returns true if set is changed, else false.
void clear()	Clears the Set by removing all the elements from this set.
boolean contains(Object o)	Returns true if this set contains the specified object.
boolean containsAll(Collection<?> c)	Returns true if this set contains all the elements of the specified collection.
boolean equals(Object o)	Compares the specified object with this set for equality.
int hashCode()	Returns the hash code value for this set.
boolean isEmpty()	Returns true if this set is empty.
Iterator<E> iterator()	Returns an iterator over the elements in this set.
boolean remove(Object o)	Removes the specified element from this set if it is present.
boolean removeAll(Collection<?> c)	Removes from this set all its elements that are contained in the specified collection.
boolean retainAll(Collection<?> c)	Retains only the elements in this set that are contained in the specified collection.
int size()	Returns the number of elements in the set.
Object[]toArray()	Returns an array containing all the elements in this Set. The return type of array of objects will be of type Object class.
<T> T[]toArray(T[] a)	Creates an array containing all the elements in this Set. The return type of the array of objects will be according to the type of objects in the collection.

Set

- Set is a collection that does not contain duplicates.
  - Set collection in java.util models the mathematical concept set.
  - The concepts of Union, Intersection and difference of a set are available in the Set interface and supported by its subclasses.
- Two subclasses exists
  - HashSet and TreeSet.
  - TreeSet is a sorted collection as it inherits the SortedSet interface (sub-interface of Set) whereas
  - HashSet is not a sorted collection.
- HashSet uses the concept of Hashing.

HashSet

- This class can be used for effectively storing and retrieving the elements but the order is not guaranteed by this class.
- In case you need to retrieve the elements in a sorted order use TreeSet class.
- HashSet permits null to be added to the collection.
- Let us take an example to demonstrate HashSet class.

Linked hash set

- Linked hashset preserves the order, but hashset doesn't preserve the order of the elements

TreeSet

- offers a strict control over the order of elements in the collection.
- The collection is a sorted collection.
- may not offer you the best performance in terms of retrieving elements speedily (use HashSet instead of TreeSet).
- does not permit null in the collection.

Comparator interface

- Methods:

  public int compare(Object o1,Object o2)

  public boolean equals(Object o1)

Difference between comparator and comparable interfaces

Difference between Comparable & 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.

Queries?