Java Spring & AWS
Sep 7, 2021 - Oct 8, 2021
Monday to Friday
9:30 AM ET - 4:30 PM ET

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 - Introduct

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 jav programming, so we can make a object array 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

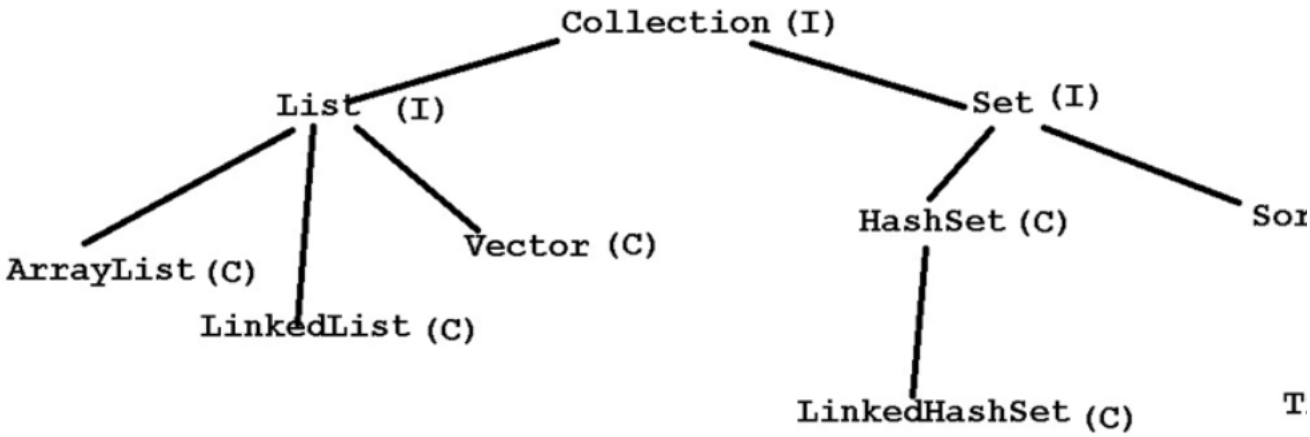
- Fixed size
- Unable to identify duplicate values
- No built in data structure

✓ to overcome these draw backs, sun introduce Cc framework

Collections

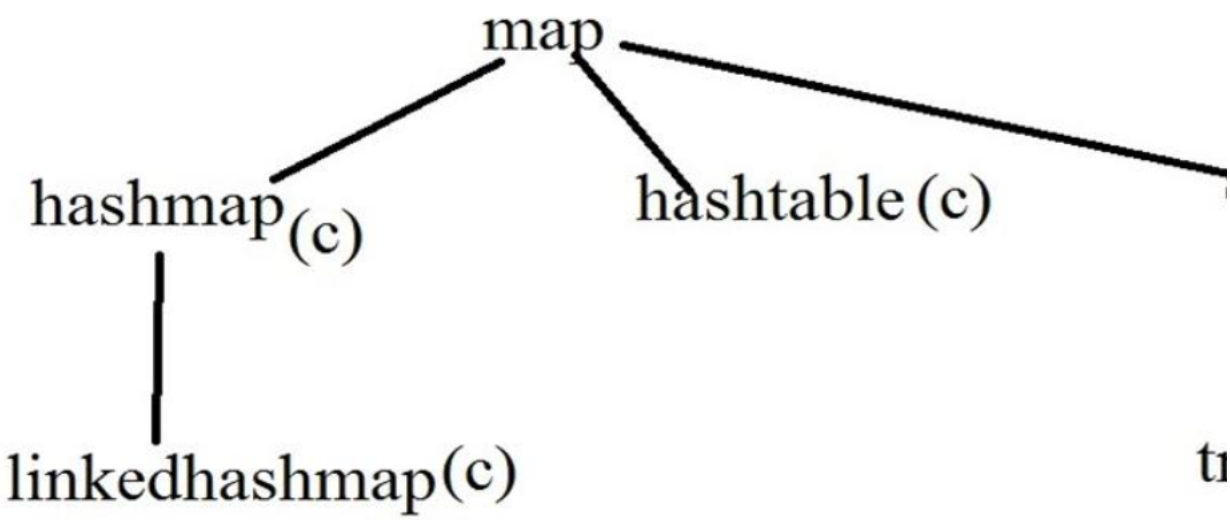
- **Collections in java** is a framework that provides an architect manipulate the group of objects.
- All the operations that you perform on a data such as search insertion, manipulation, deletion etc. can be performed by J
- Java Collection simply means a single unit of objects. Java Cc framework provides many interfaces (Set, List, Queue, Dequ classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet LinkedHashSet, TreeSet etc).

Collection Inheritance hierarchy



from java.util package

Map interface



Methods of Collection Interface

Method	Description
boolean add(E e)	Adds the specified element to and returns true. It returns fals collection does not accept dup
boolean addAll (Collection<? extends E> c)	Adds all the elements in the sp this collection and returns true
void clear()	Clears the collection by Remo from this collection.
boolean contains(Object o)	Returns true if this collection object.
boolean containsAll (Collection<?> c)	Returns true if this collection elements in the specified collc
boolean equals(Object o)	Compares the specified object equality and returns true if equ
int hashCode()	Returns the hash code value f
boolean isEmpty()	Returns true if this collection i
Iterator<E> iterator()	Returns an iterator to iterate th collection.
boolean remove(Object o)	Removes a single instance of from this collection, if it is pres
boolean removeAll (Collection<?> c)	Removes all the elements of th also contained in the specified
boolean retainAll (Collection<?> c)	Retains only the elements in th contained in the specified colle
int size()	Returns the number of elemen
Object[]toArray()	Returns an array containing al collection. The return type of a type Object class.
<T> T[] toArray(T[] a)	Creates an array containing al collection. The return type of th will be the according to the typ collection.

Methods of List interface

Method	Description
boolean addE(E e)	Appends the specified element to th
void add (int index, E element)	Inserts the specified element at the this list.
boolean addAll(Collection<? extends E> c)	Appends all the elements in the spe the end of this list, in the order that 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

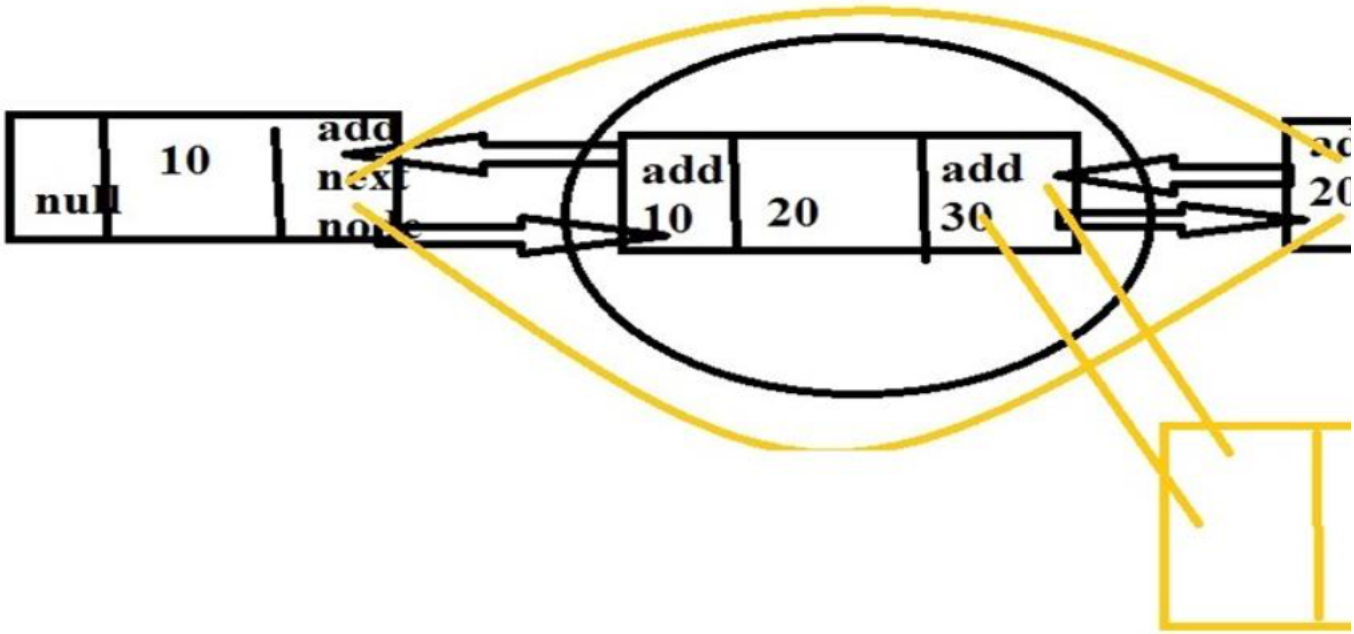
boolean addAll(int index, Collection<? extends E> c)	Inserts all of the elements in the spe this list at the specified position
void clear()	Clear by Removing all the elements
boolean contains(Object o)	Returns true if this list contains the s
boolean containsAll(Collection<?> c)	Returns true if this list contains all th specified collection.
boolean equals(Object o)	Compares the specified object with t
E get(int index)	Returns the element at the specified
int hashCode()	Returns the hash code value for this
int indexOf(Object o)	Returns the index of the first occurre element in this list, or -1 if this list do element.
boolean isEmpty()	Returns true if this list contains no el
Iterator<E> iterator()	Returns an iterator over the element proper sequence.
int lastIndexOf (Object o)	Returns the index of the last occurre element in this list, or -1 if this list do element.
ListIterator<E> listIterator()	Returns a list iterator over the eleme
ListIterator<E> listIterator(int index)	Returns a list iterator of the element at the specified position in this list.
E removeint(int index)	Removes the element at the specifie list.
boolean removejava.lang.Object(Object o)	Removes the first occurrence of the from this list, if it is present.
boolean removeAll(Collection<?> c)	Removes from this list all of its elem contained in the specified collection.
boolean retainAll(Collection<?> c)	Retains only the elements in this list in the specified collection.
E set(int index, E element)	Replaces the element at the specifie list with the specified element and re previously stored at the specified po
int size()	Returns the number of elements in t
List<E>subList(intfrom Index, int toIndex)	Returns a sub list containing elemen specified fromIndex, inclusive, and t
Object[]toArray()	Returns an array containing all of th collection. The return type of array o type Object class.
<T> T[] toArray(T[] a)	Creates an array containing all of th collection. The return type of the arr the according to the type of objects i

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

linkedList (Continue)

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

LinkedList Implementation



Difference between ArrayList and LinkedI

ArrayList	LinkedList
1) ArrayList internally uses **dynamic array** to store the elements.	LinkedList internally u 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 Li ArrayList because it us bit shifting is required i
3) ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act** because it implements
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for**

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

Methods

- void addElement(Object obj)
- Adds the specified component to the end of this vector, increasing

- 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 sp

Difference between vector and arraylist

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

It's a subclass of vector class

Methods

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

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 th

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 ConcurrentModificationException on the other hand with modify collection.
- Modifying a collection simply means removing an elem content of an item stored in the collection. This occurs b loop implicitly creates an iterator but it is not exposed to t can't modify the items in the collections

Before generics, we can store any type of objects in collection
Now generics, forces the java programmer to store specific typ
There are mainly 3 advantages of generics. They are as follows:

1) Type-safety : We can hold only a single type of objects in ge
allow to store other objects.

2) Type casting is not required: There is no need to typecast th

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);
```

- 3. **Compile-Time Checking:** It is checked at compile time so p occur at runtime. The good programming strategy says it is f handle the problem at compile time than runtime.
- List<String> list = **new** ArrayList<String>();
- list.add("hello");
- list.add(32);//Compile Time Error

Syntax (0:17/0:19)

- 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 collectic 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. H
  using **T** type parameter to create the generic class of specific

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, In
etc.). The type you specify for the class, will be used to store
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 t
  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 ac argument.

Let's see a simple example of java generic method to print arra are using here **E** to denote the element.

Bounded Type Parameters

- There may be times when you'll want to restrict the kinds c allowed to be passed to a type parameter. For example, operates on numbers might only want to accept instances c subclasses. This is what bounded type parameters are for.
- To declare a bounded type parameter, list the type par followed by the extends keyword, followed by its upper bou

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

Java HashSet class

- Java HashSet class is used to create a collection that uses a h storage.

The important points about Java HashSet class are:

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

What is Hashing

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

Hash function should return the same hash code each a
when function is applied on same or equal objects. In other wo
objects must produce same hash code consistently.

When we use hashing

- Hashing is designed to solve the problem of needing to **ef** store an item in a collection. For example, if we have a list of English and we want to check if a given word is in the inefficient to successively compare the word with all 10,00 find a match. Hashing is a technique to make things m effectively narrowing down the search at the outset.
- Hashing means using some function or algorithm to map some representative integer value. This so-called **h** simply **hash**) can then be used as a way to **narrow down o** looking for the item in the map.

- 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 t already present and returns
boolean addAll(Collection<? extends E> c)	Adds all the elements in the to this set if they are not alre returns true if set is changed
void clear()	Clears the Set by removing this set.
boolean contains(Object o)	Returns true if this set conta object.
boolean containsAll(Collection<?> c)	Returns true if this set conta of the specified collection.
boolean equals(Object o)	Compares the specified obje equality.
int hashCode()	Returns the hash code value
boolean isEmpty()	Returns true if this set is em
Iterator<E> iterator()	Returns an iterator over the
boolean remove(Object o)	Removes the specified elem is present.
boolean removeAll(Collection<?> c)	Removes from this set all its contained in the specified co
boolean retainAll(Collection<?> c)	Retains only the elements in contained in the specified co
int size()	Returns the number of elem
Object[]toArray()	Returns an array containing this Set. The return type of a be of type Object class.
<T> T[]toArray(T[] a)	Creates an array containing this Set. The return type of t will be according to the type collection.

- Set is a collection that does not contain duplicates.
  - Set collection in java.util models the mathematical concep
  - The concepts of Union, Intersection and difference of a s in the Set interface and supported by its subclasses.
- Two subclasses exists
  - HashSet and TreeSet.
  - TreeSet is a sorted collection as it inherits the SortedSe 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 t the order is not guaranteed by this class.
- In case you need to retrieve the elements in a sorted or 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 pre
of the elements

TreeSet

- offers a strict control over the order of elements in the colle
- The collection is a sorted collection.
- may not offer you the best performance in terms of retr 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 & Compara

Comparable	Com
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 **multip** words, we can sort the coll elements such as id, name a
2) Comparable **affects the original class** i.e. actual class is modified.	Comparator **doesn't affect** class is not modified.
3) Comparable provides **compareTo() method** to sort elements.	Comparator provides **compa**
4) Comparable is found in **java.lang** package.	Comparator is found in **java**
5) We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list e by **Collections.sort(List,Co**

Queries?