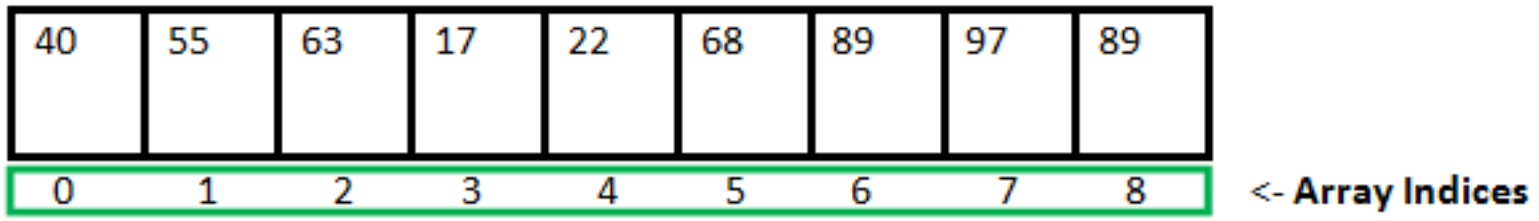
Unit 10:

Holding Collection of Data

Arrays:

- An array is a collection of similar types of data.
- Arrays are used to store multiple values in a single variable.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered, and each has an index beginning from 0.
- Java array can be also be used as a static field, a local variable, or a method parameter.
- The **size** of an array must be specified by int or short value and not long.
- An array can contain primitives (int, char, etc.) and object (or non-primitive) references of a class depending on the definition of the array.
- In the case of primitive data types, the actual values are stored in contiguous memory locations. In the case of class objects, the actual objects are stored in a heap segment.



Array Length = 9
First Index = 0
Last Index = 8

Syntax to Declare an Array in Java :

1. dataType[] arrayName; (or)
2. dataType []arrayName; (or)
3. dataType arrayName[];

- dataType - it can be primitive data types like int, char, double, byte, etc. or Java objects
- arrayName - it is an identifier

Instantiation of an Array in Java :

arrayName = **new** datatype[size];

Initialize Arrays in Java :

In Java, we can initialize arrays during declaration. For example,

```
//declare and initialize and array

int[] age = {12, 4, 5, 2, 5};
```

- ***Note that we have not provided the size of the array. In this case, the Java compiler automatically specifies the size by counting the number of elements in the array (i.e. 5).***

- We can also initialize arrays in Java, using the index number. For example,

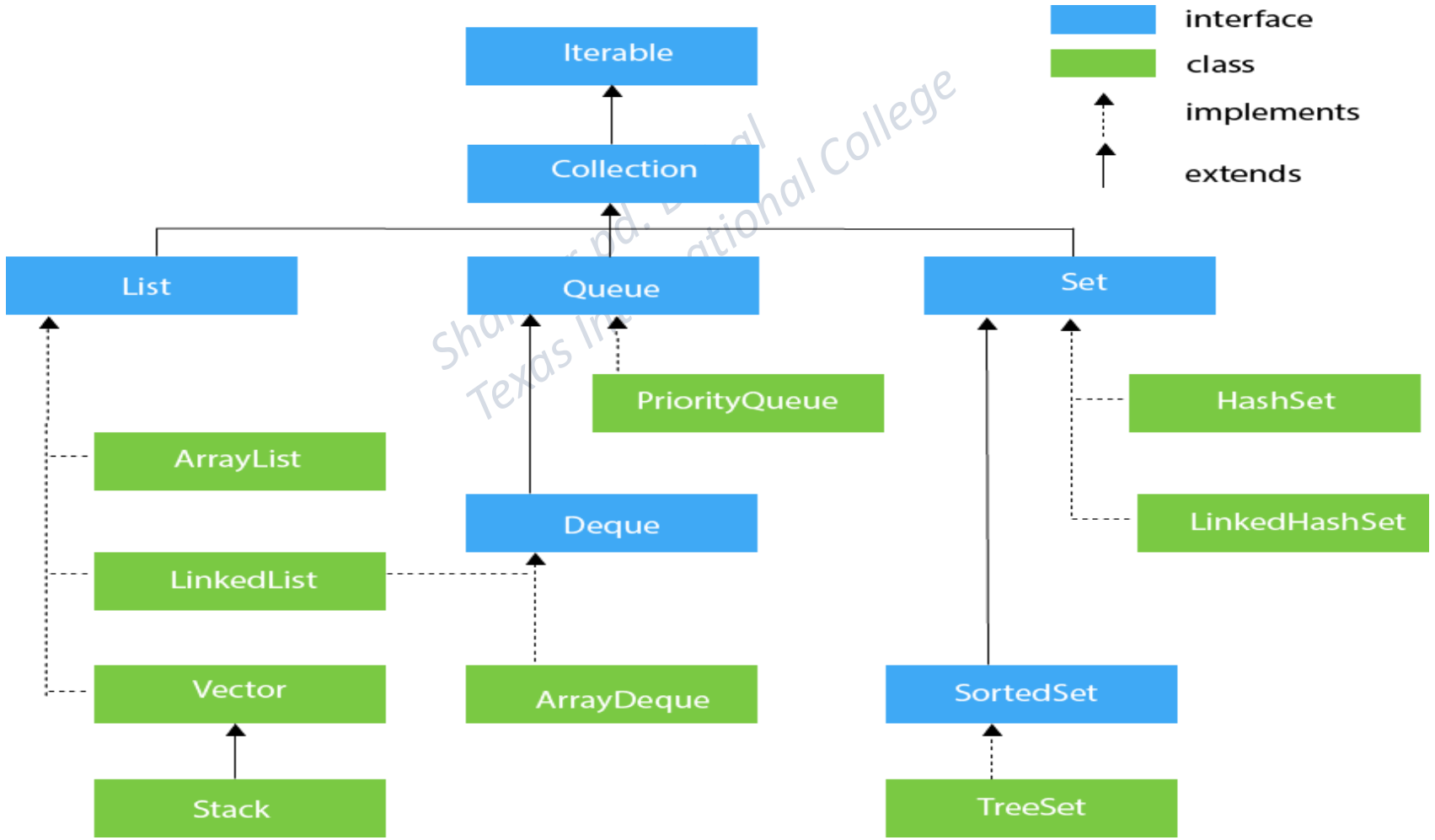
```
int[] age = new int[5];
// initialize array
age[0] = 12;
age[1] = 4;
age[2] = 5;
..
```

Collections in Java :

- A Collection represents a single unit of objects, i.e., a group.
- The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.
- Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Hierarchy of Collection Framework :

- The **java.util** package contains all the classes and interfaces for the Collection framework.



Collections Interface :

- The Collection interface is the root interface of the Java collections framework.
- There is no direct implementation of this interface. However, it is implemented through its subinterfaces like List, Set, and Queue.
- For example, the ArrayList class implements the List interface which is a subinterface of the Collection Interface.
- The Collection interface includes subinterfaces that are implemented by various classes in Java. Following are the subinterfaces:

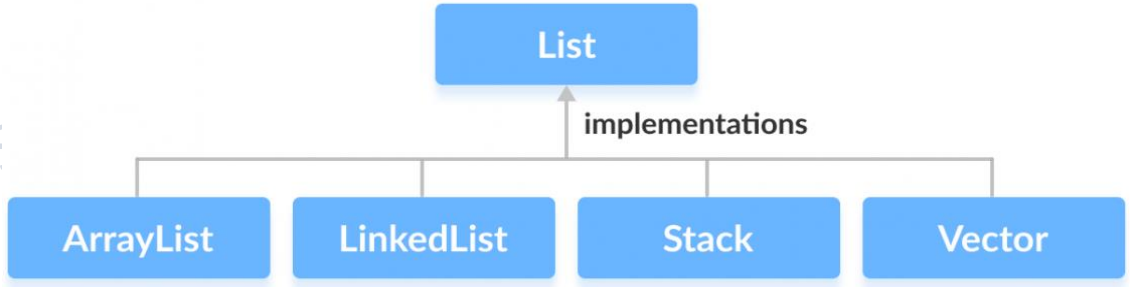
1. List Interface :

- List interface is an ordered collection that allows us to store and access elements sequentially. It extends the Collection interface.

Classes that Implement List:

- Since List is an interface, we cannot create objects from it. In order to use functionalities of the List interface, we can use these classes:

i. ArrayList
ii. LinkedList
iii. Vector
iv. Stack



Note : The ArrayList and LinkedList are widely used in Java programming.

Methods of List

- The List interface includes all the methods of the Collection interface.
- Its because Collection is a super interface of List.

Some of the commonly used methods of the Collection interface that's also available in the List interface are:

1. add() - adds an element to a list
2. addAll() - adds all elements of one list to another
3. get() - helps to randomly access elements from lists
4. iterator() - returns iterator object that can be used to sequentially access elements of lists
5. set() - change/update elements of lists
6. remove() - removes an element from the list
7. removeAll() - removes all the elements from the list
8. clear() - removes all the elements from the list (more efficient than removeAll())
9. size() - returns the length of lists
10. toArray() - converts a list into an array
11. contains() - returns true if a list contains specified element
12. sort - It is used to sort the elements of the list on the basis of specified comparator.

A. Creating List using ArrayList class:

Syntax:

```
List<Obj> list = new ArrayList<Obj> ();
```

Note: *Obj is the type of the object to be stored in List*

B. Creating List using LinkedList class:

Syntax:

```
List<Obj> list = new LinkedList<Obj> ();
```

Note: *Obj is the type of the object to be stored in List*

C. Creating List using Stack class:

Syntax:

List<Obj> list = new Stack<Obj> ();

Note: *Obj is the type of the object to be stored in List*

2. Set Interface :

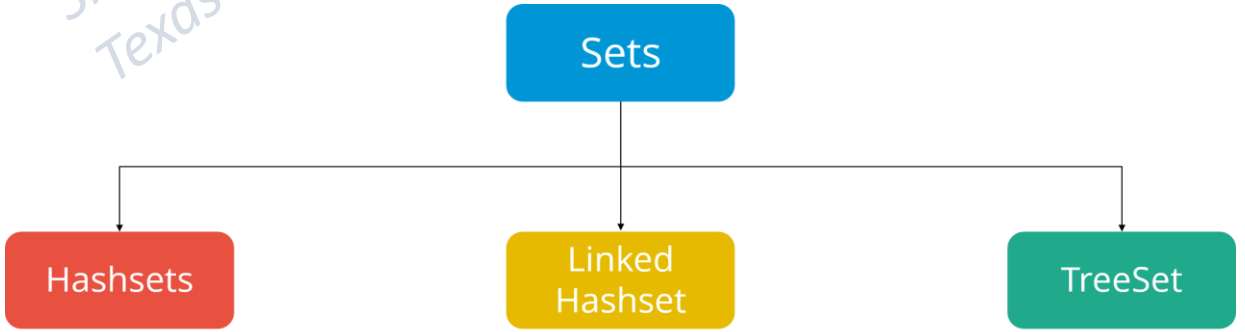
- The set interface is a part of the Java Collections Framework.
- The set interface is present in java.util package and extends the Collection interface.
- It is an unordered collection of objects in which duplicate values cannot be stored.
- This interface contains the methods inherited from the Collection interface and adds a feature that restricts the insertion of the duplicate elements.
- Set has its implementation in various classes such as HashSet, TreeSet and LinkedHashSet.
- On the Set, we can perform all the basic mathematical operations like intersection, union and difference.

A. Creating Set using `HashSet` class:

Syntax:

```
Set<Obj > setHash = new HashSet<Obj >();
```

Note: *Obj is the type of the object to be stored in Set.*



B. Creating Set using `LinkedHashSet` class:

Syntax:

```
Set<Obj > setLinkedHash = new LinkedHashSet<Obj>();
```

Note: *Obj is the type of the object to be stored in Set.*

C. Creating Set using TreeSet class:

Syntax:

```
Set<Obj> setTree = new TreeSet<Obj>();
```

Note: *Obj is the type of the object to be stored in Set.*

Some of the commonly used methods of the Collection interface that's also available in the Set interface are:

1. **add()** - adds the specified element to the set
2. **addAll()** - adds all the elements of the specified collection to the set
3. **iterator()** - returns an iterator that can be used to access elements of the set sequentially
4. **remove()** - removes the specified element from the set
5. **removeAll()** - removes all the elements from the set that is present in another specified set
6. **retainAll()** - retains all the elements in the set that are also present in another specified set
7. **clear()** - removes all the elements from the set
8. **size()** - returns the length (number of elements) of the set
9. **toArray()** - returns an array containing all the elements of the set
10. **contains()** - returns true if the set contains the specified element
11. **containsAll()** - returns true if the set contains all the elements of the specified collection
12. **hashCode()** - returns a hash code value (address of the element in the set)

3. Map Interface :

❖ The Map interface of the Java collections framework provides the functionality of the map data structure.

Working of Map

❖ In Java, elements of Map are stored in **key/value** pairs.

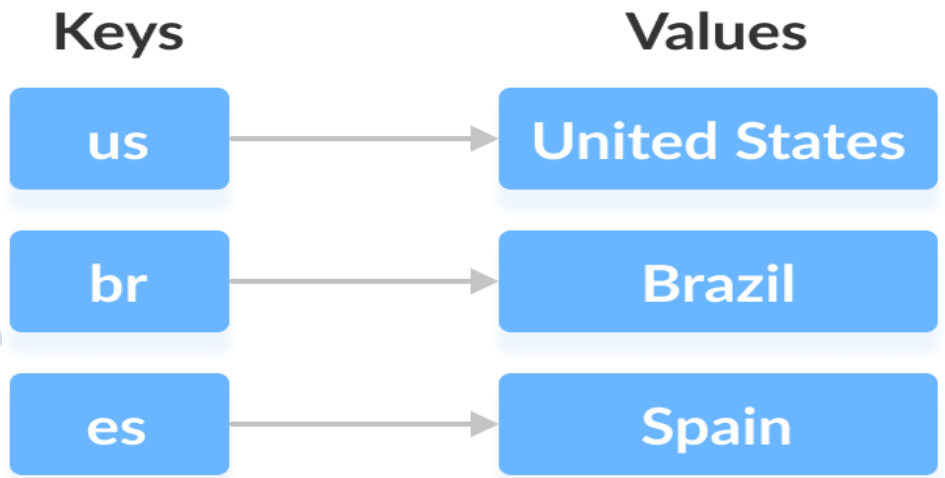
❖ **Keys** are unique values associated with individual **Values**.

❖ A map cannot contain duplicate keys and each key is associated with a single value.

❖ We can access and modify values using the keys associated with them.

❖ In the diagram, we have

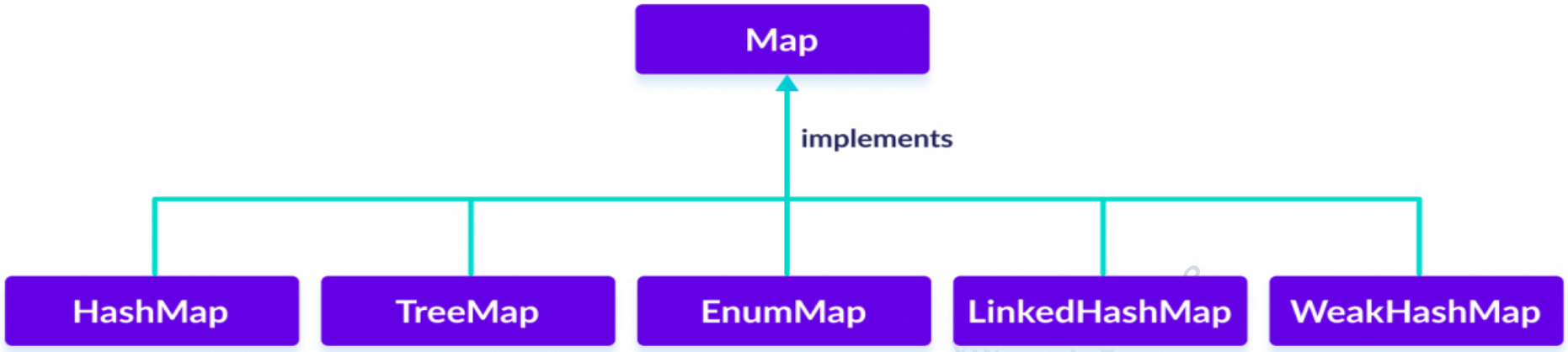
Values: ***United States, Brazil, and Spain*** and we have corresponding
Keys: ***us, br, and es.***



Now, we can access those values using their corresponding keys.

❖ Map is an interface so we cannot create objects from it.
❖ In order to use functionalities of the Map interface, we can use these classes:

a. HashMap
b. EnumMap
c. LinkedHashMap
d. WeakHashMap
e. TreeMap



A. Creating Map using `HashMap` class:

Syntax:

```
Syntax:
Map<Key, Value> hasMap = new HashMap<>();
```

B. Creating Map using TreeMap class:

Syntax:

```
Map<Key, Value> treeMap = new TreeMap<>();
```

C. Creating Map using `LinkedHashMap` class:

Syntax:

```
Map<Key, Value> linkedHashMap = new LinkedHashMap<>();
```

Collections Classes:

- Java Collections framework comes with many implementation classes for the interfaces. Most common implementations are Array List, Linked List, Hash Set, Tree Set and HashMap.

1. Array List :

- ArrayList is a part of the Java collection framework, and it is a class of java.util package.
- It provides us with dynamic arrays in Java.
- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because the array works on an index basis.
- In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.
- We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases.

Creating an ArrayList :

Syntax :

```
ArrayList<Type> arrayList= new ArrayList<>();
```

Here, **Type** indicates the type of an arraylist.

For example : `ArrayList<Integer> arrayList = new ArrayList<>();`

Basic Operations/Methods on ArrayList :

❖ The ArrayList class provides various methods to perform different operations on arraylists.

a. Add elements
b. Access elements
c. Change elements
d. Remove elements
e. size : Returns the length of the arraylist.
f. sort : Sort the arraylist elements.

a. Add elements

❖ To add a single element to the arraylist, we use the add() method of the ArrayList class.

b. Access elements

❖ To access an element from the arraylist, we use the get() method of the ArrayList class.

c. Change elements

❖ To change elements of the arraylist, we use the set() method of the ArrayList class.

d. Remove elements

❖ To remove an element from the arraylist, we can use the remove() method of the ArrayList class.

2. Linked List :

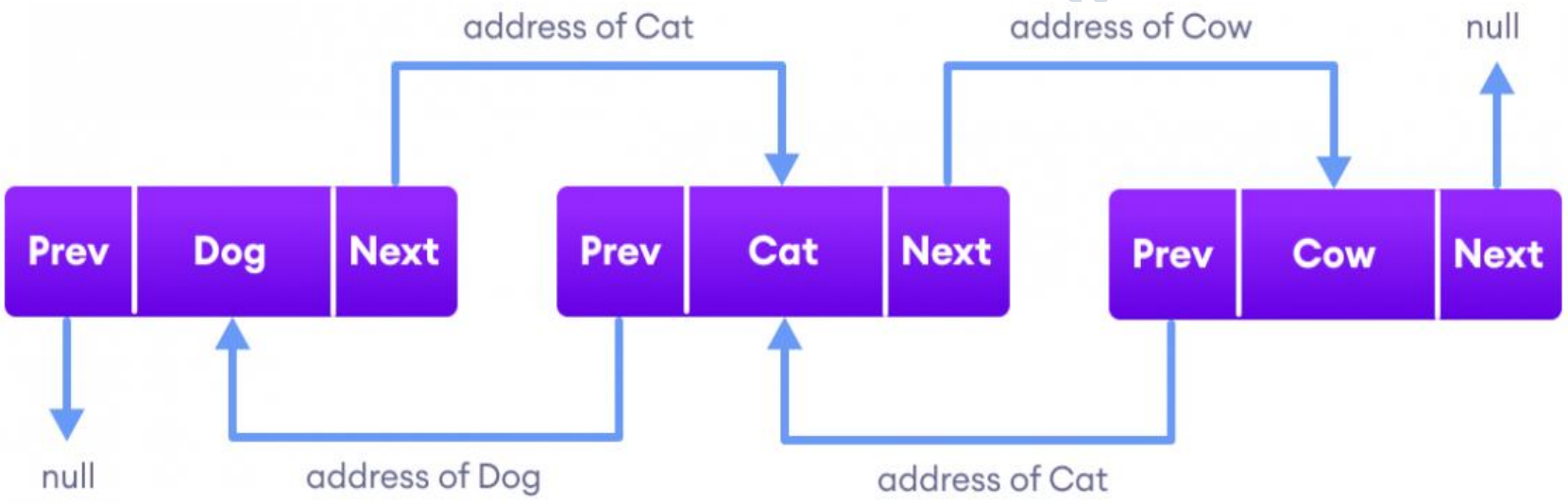
- The LinkedList class of the Java collections framework provides the functionality of the linked list data structure (doubly linkedlist).
- Each element in a linked list is known as a node. It consists of 3 fields:

1. **Prev** - stores an address of the previous element in the list. It is null for the first element
2. **Next** - stores an address of the next element in the list. It is null for the last element
3. **Data** - stores the actual data

Creating a Java LinkedList

```
LinkedList<Type> linkedList = new LinkedList<>();
```

Here, Type indicates the type of a linked list.



Basic Operations/Methods on **LinkedList** :

❖ The **LinkedList** class provides various methods to perform different operations on **LinkedList**.

a. Add elements
b. Access elements
c. Change elements
d. Remove elements
e. peek : returns the first element (head) of the linked list
f. poll : returns and removes the first element from the linked list
g. addFirst : adds the specified element at the beginning of the linked list
h. addLast : adds the specified element at the end of the linked list

a. Add elements

❖ To add a single element to the LinkedList, we use the add() method of the LinkedList class.

b. Access elements

❖ To access an element from the LinkedList, we use the get() method of the LinkedList class.

c. Change elements

❖ To change elements of the LinkedList, we use the set() method of the LinkedList class.

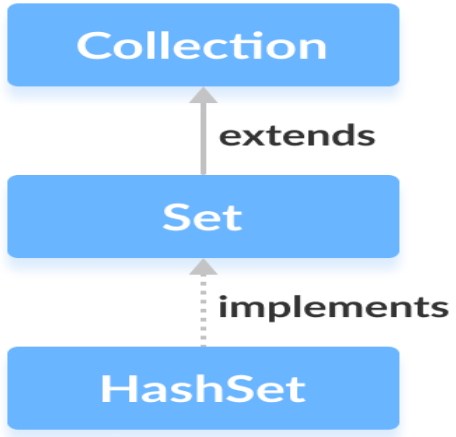
d. Remove elements

❖ To remove an element from the LinkedList, we can use the remove() method of the LinkedList class.

3. Hash Set :

❖ The HashSet class of the Java Collections framework provides the functionalities of the hash table data structure.

Creating a HashSet



❖ In order to create a hash set, we must import the java.util.HashSet package first.

❖ Once we import the package, here is how we can create hash sets in Java.

```
HashSet<Integer> numbers = new HashSet<>(8, 0.75);
```

Notice, the part new HashSet<>(8, 0.75).

Here, the first parameter is **capacity**, and the second parameter is **loadFactor**.

1. **capacity** - The capacity of this hash set is 8. Meaning, it can store 8 elements.
2. **loadFactor** - The load factor of this hash set is 0.6. This means, whenever our hash set is filled by 60%, the elements are moved to a new hash table of double the size of the original hash table.

Default capacity and load factor :

```
HashSet<Integer> numbers1 = new HashSet<>();
```

By default :

❖ the capacity of the hash set will be 16.

❖ the load factor will be 0.75.

Methods of HashSet

The HashSet class provides various methods that allow us to perform various operations on the set.

a. Add elements :

i. add() - inserts the specified element to the set
ii. addAll() - inserts all the elements of the specified collection to the set

b. Access elements :

❖ To access the elements of a hash set, we can use the iterator() method.

c. Remove elements :

i. remove() - removes the specified element from the set
ii. removeAll() - removes all the elements from the set

d. Union of Sets :

❖ To perform the union between two sets, we can use the addAll() method.

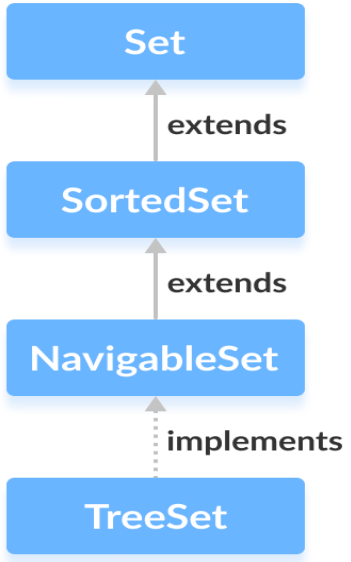
e. Intersection of Sets :

❖ To perform the intersection between two sets, we can use the retainAll() method.

f. Difference of Sets

❖ To calculate the difference between the two sets, we can use the removeAll() method.

3. Tree Set :



- The TreeSet class of the Java collections framework provides the functionality of a tree data structure.
- Java TreeSet class contains unique elements only like HashSet.
- Java TreeSet class access and retrieval times are quiet fast.
- Java TreeSet class doesn't allow null element.
- Java TreeSet class is non synchronized.
- Java TreeSet class maintains ascending order.
- Java TreeSet class contains unique elements only like HashSet.
- Java TreeSet class access and retrieval times are quite fast.
- Java TreeSet class doesn't allow null elements.
- Java TreeSet class is non-synchronized.
- Java TreeSet class maintains ascending order.

Constructors of TreeSet Class are as follows:

1. TreeSet():

This constructor constructs an empty tree set that will be sorted in an ascending order according to the natural order of its elements.

2. TreeSet(Comparator):

This constructor constructs an empty tree set that will be sorted according to the given comparator.

3. TreeSet(Collection):

This constructor builds a tree set that contains the elements of the collection **c**.

Methods of TreeSet

The HashSet class provides various methods that allow us to perform various operations on the set.

a. Add elements :

i. add() - inserts the specified element to the set
ii. addAll() - inserts all the elements of the specified collection to the set

b. Access elements :

❖ To access the elements of a hash set, we can use the iterator() method.

c. Remove elements :

i. remove() - removes the specified element from the set
ii. removeAll() - removes all the elements from the set

d. size() :

Returns the number of elements in this set (its cardinality).

e. clear():

Removes all of the elements from this set.

f. contains():

Returns true if this set contains the specified element.

Accessing Collections:

- To access elements of a collection, either we can use index if collection is list based or we need to traverse the element.
- There are three possible ways to traverse through the elements of any collection.

1. Using Iterator interface
2. Using ListIterator interface
3. Using for-each loop

1. Using Iterator interface

- Iterator is an interface that is used to iterate the collection elements.
- It is part of java collection framework.
- It provides some methods that are used to check and access elements of a collection.
- Iterator Interface is used to traverse a list in forward direction, enabling you to remove or modify the elements of the collection.
- Each collection classes provide iterator() method to return an iterator.

Iterator Interface Methods :

a. **hasNext() :** Returns **true** if there are more elements in the collection. Otherwise, returns false.
b. **next() :** Returns the **next element present** in the collection. Throws NoSuchElementException if there is not a next element.
c. **remove() :** Removes the current element. Throws IllegalStateException if an attempt is made to call remove() method that is not preceded by a call to next() method.

Comparator:

- In Java, Comparator interface is used to order(sort) the objects in the collection in your own way.

- It gives you the ability to decide how elements will be sorted and stored within collection and map.

- Comparator Interface defines compare() method. This method has two parameters. This method compares the two objects passed in the parameter. It returns 0 if two objects are equal. It returns a positive value if object1 is greater than object2. Otherwise, a negative value is returned.

Rules for using Comparator interface :

1. If you want to sort the elements of a collection, you need to implement Comparator interface.

2. If you do not specify the type of the object in your Comparator interface, then, by default, it assumes that you are going to sort the objects of type Object. Thus, when you override the compare() method ,you will need to specify the type of the parameter as Object only.

3. If you want to sort the user-defined type elements, then while implementing the Comparator interface, you need to specify the user-defined type generically. If you do not specify the user-defined type while implementing the interface, then by default, it assumes Object type and you will not be able to compare the user-defined type elements in the collection