=> Data Structure :-

-> A data structure is a particular way by which we organize, manage and store the data in the computer so that it can be used effectively

-> Data structure is not a programming langaue like c, c++, java or python. It is a set of algorithms which we can implment in any programming langauges

-> Operations that we can perform on data :-

1. Searching
2. Sorting
3. Insertion
4. Deletion
5. Updation

-> Advantages :-

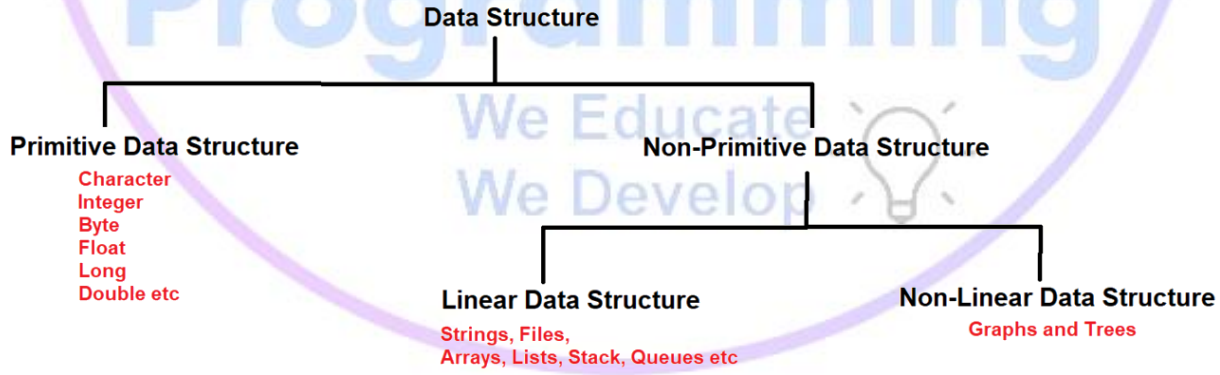
1. We can process the data easily

2. It stores the data very efficiently on the disk

3. Data structure is used to develop algorithms

4. It stores the data in a secured way

-> Types of Data Structure :-



1. Primitive Data Structure (eg Integer, Float, Long, Double etc)

2. Non-Primitive Data Structure

-> Linear DS (eg String, Files, Arrays, Lists, Stacks, Queues etc)

= In Linear DS the data is arranged in a sequential form i.e. one element is connected to only one other element in linear form

-> Non-Linear DS (eg Graphs & Trees)

= In non-linear DS, one element can connect with 'n' number of elements.

=> Arrays :-

1. Array is a collection of similar type of data (homogeneous data)

2. Array elements are stored in contigeous memory locations

3. Array can contain primitive or non-primitive elements

4. Array is index based data structure, first index position of an array is 0

5. Array length starts from 1

=> Types of Arrays :-

1. Single Dimensional Array
   -> 1D Array
2. Multi-Dimensional Array
   -> 2D Array
   -> 3D Array
   -> 4D, 5D, 6D.... Array
   -> Zic-Zac Array
3. Anonymous Array

=> Single Dimensional Array (for example 1D Array) :-

-> In this type of array, there is only one row or one column

-> Following points for each type of array :-

= Declaration :

1. Array can be declared normally like simple variables but we have to provide square([]) braces

2. When we declare an array, we dont need to provide the size of an array

= Creation :-

1. When we create an array by using new keyword, we have to provide the size of an array

2. When we create an array by using new keyword, all the index position will be initialized by its default values

=> We can declare and create an array within a single line

=Initialization :-

1. We can provide the values at particular index position

=> We can declare, create and initlize an array within single line

= Retrieve :-

1. We can retrieve the value by using for loop

⇨ Different cases for Declaration, Creation & Initialization of an array

=> Declaration :-

1. int[] a,b; //a and b are both arrays
2. int []a,b; //a and b both are arrays
3. int a[],b; //correct, a is an array but b is simple variable
4. int a,b[]; //correct, a is normal variable but b is an array
5. int a[], b[]; //both a and b are array
6. int []a, b[]; //a and b are both arrays

=> Creation :-

1. a=new int[5]; //correct
2. a=new int[]; //error

3. int[] a=new int[5]; //correct

4. int a[]=new int[5]; //correct

5. int []a=new [5]int; //error

6. int[] a=new int[0]; //it will successfully compile and run

7. int[] a=new int[-3]; //it will compile but provides runtime exception i.e. java.lang.NegativeArraySizeException

=> Initialization :-

1. int[] a=new int[3];

a[3]=100; //compile successfully but will throw runtime exception saying ArrayIndexOutOfBoundsException

2. If we dont initialize any proper index position value, then it will compile and run successfully

3. a[-1]=100; //compile successfully but will provide runtime exception saying ArrayIndexOutOfBoundsException

---

-> Points to remember :-

1. Array are Objects in java
2. Arrays are stored in "Heap Area"

Interview Questions :-

1. Difference between declaring, creating and initializing an array with and without using new keyword

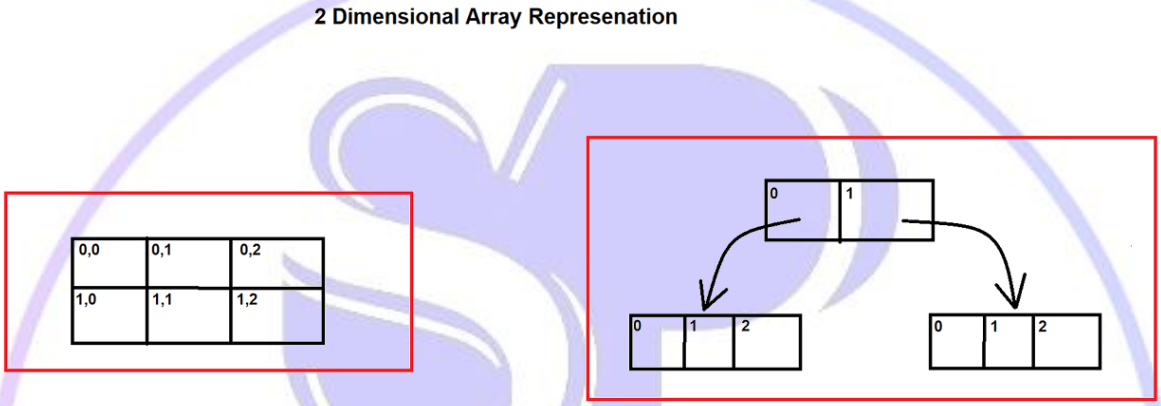
-> When we create an array without using new keyword then default value will not be initialized in the array, all the values will be initlized automatically in the array

---

=> Types of Arrays :-

1. Single Dimensional Array
   -> 1D Array
2. Multi-Dimensional Array
   -> 2D Array
   -> 3D Array
   -> 4D, 5D, 6D.... Array
   -> Zic-Zac Array
3. Anonymous Array

=> 2D Array :-



-> Following points for 2D array :-

= Declaration

-> For declaring 2D array, we have to use double square braces

= Creation :-

-> When we create 2D array, we have to provide the size of an array

-> a=new int[2][3]; // there are 2 rows and 3 columns

-> Whenever we create an array by using new keyword, then default values will be initialized in the array blocks

-> We can declare and create 2D array in a single line

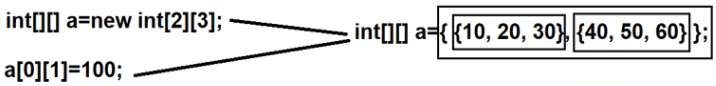
int[][] a=new int[2][3];

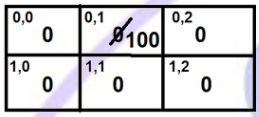
= Initialization

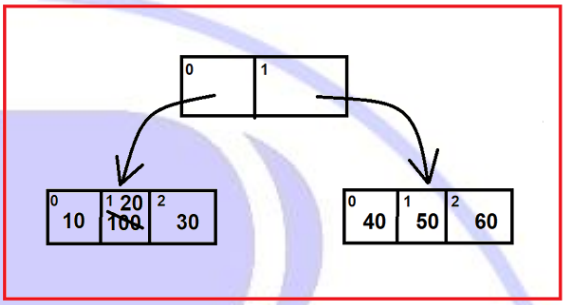
-> a[0][1]=100; //will initialize 100 value at 0,1 index position

-> We can declare, create and initialize an array in single line

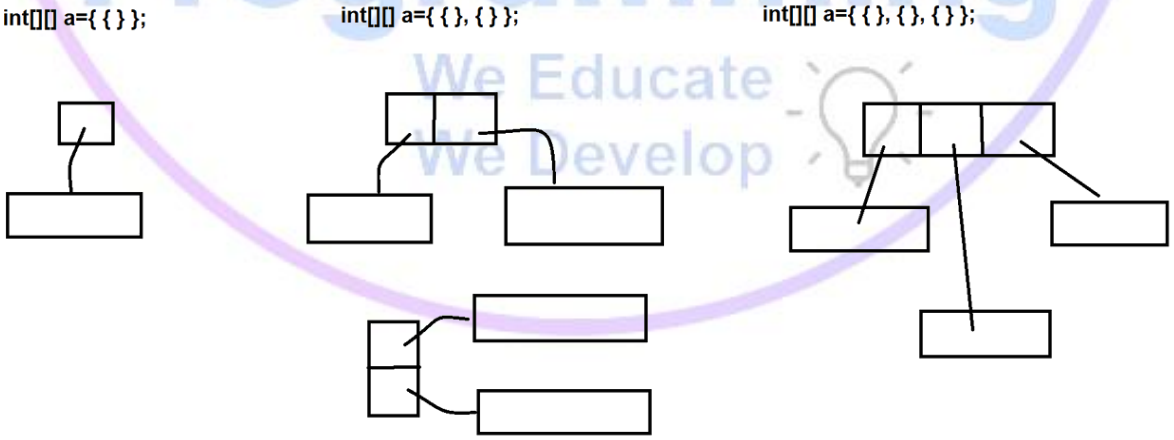
int[][] a={{10,20,30}, {40,50,60}};

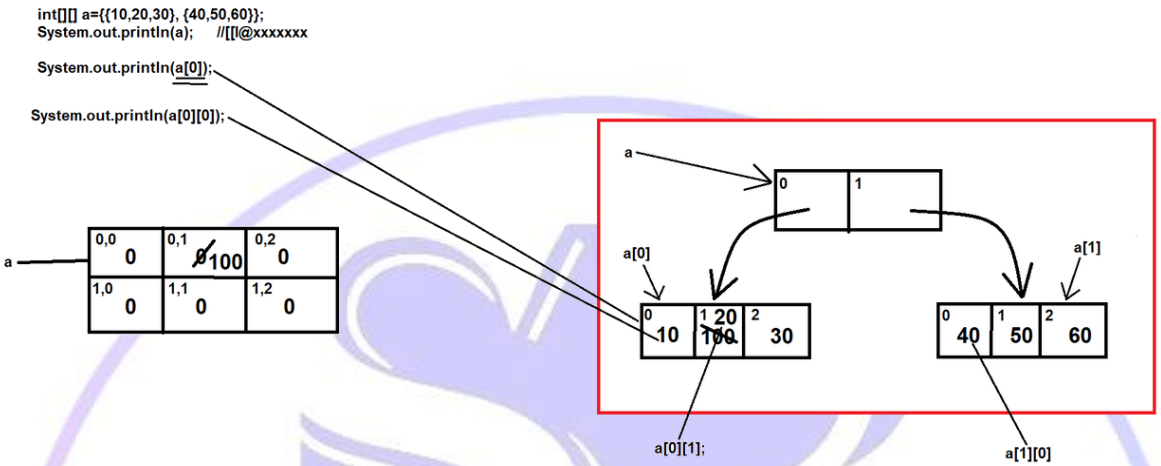

int[][] a=new int[2][3];
a[0][1]=100;
int[][] a={ {10, 20, 30}, {40, 50, 60} };


0,0
0
0,1
0
100
0,2
0
1,0
0
1,1
0
1,2
0


0
1
0
10
1
20
100
2
30
0
40
1
50
2
60


a
0
1
100
2
a[1]=100;
int[] a={10,20,30};


int[][] a={ { } };
int[][] a={ { }, { } };
int[][] a={ { }, { }, { } };



= Retrieve

-> We can retrieve elements by using for loop

=> Declaration :-

1. int[][] a; //prefered way
2. int [][]a;
3. int[][]a;
4. int [][] a;
5. int []a[];

=> Cases for Declaration :-

1. int [][]a, b; //a (2D array); b (2D array)
2. int a[][], b; //a (2D array); b (simple variable)
3. int []a[], b; //a (2D array); b (1D array)
4. int []a[], b[]; //a (2D array); b (2D array)

5. int [][]a, b[]; //a (2D array); b (3D array)

6. int [][]a, []b; //compile time error

=> Cases for Creation :-

1. a=new int[2][3]; //correct

2. a=new int[][]; //compile time error (array dimension missing)

3. a=new int[2][]; //correct

4. a=new int[][3]; //compile time error (']' expected)

=> Cases for creation within single line :-

1. int[][] a=new int[2][3]; //correct
2. int[][] a=new int[2][]; //correct
3. int[][] a=new int[][]; //error
4. int[][] a=new int[0][0]; //correct
5. int[][] a=new int[-2][3];//will compile successfully but provides runtime exception saying java.lang.NegativeArraySizeException

=> Types of Arrays :-

1. Single Dimensional Array
   -> 1D Array
2. Multi-Dimensional Array
   -> 2D Array
   -> 3D Array
   -> 4D, 5D, 6D.... Array
   -> Zic-Zac Array
3. Anonymous Array

=> 3D Array :-

-> Diagram representation

-> How to declare, create and initialize 3D array

= Declaration :-

-> We can declare 3D array by using 3 square braces.

= Creation :-

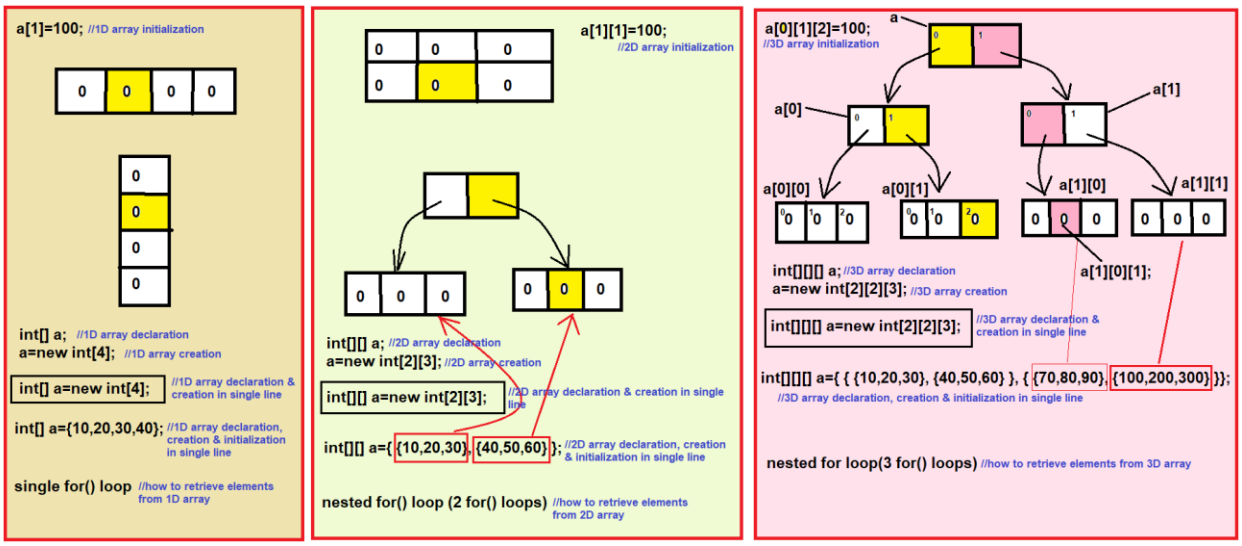
-> We can create an array by using new keyword

= Initialization :-

```
a[0][1][2]=100;
```

= Retrieve :-

We can retrieve the elements of 3D array by using 3 for loops



Declaration :-

1. int[][][] a;
2. int[] [][]a;
3. int [][][]a;
4. int []a[][];
5. int [][]a[];
6. int a[][][];

Cases for Declaration :-

1. int[][][]a, b; //a & b are 3D array
2. int [][]a[],b; //a is 3D array & b is 2D array
3. int [][]a[], b[]; //a and b are 3D array

4. int [][][]a, b[]; //a is 3D array & b is 4D array

5. int [][][]a, []b; //error

Cases for creation :-

1. a=new int[2][2][3]; //correct
2. a=new int[2][2][]; //correct
3. a=new int[2][][]; //correct
4. a=new int[][][]; //error
5. a=new int[2][][3]; //error

=> Types of Arrays :-

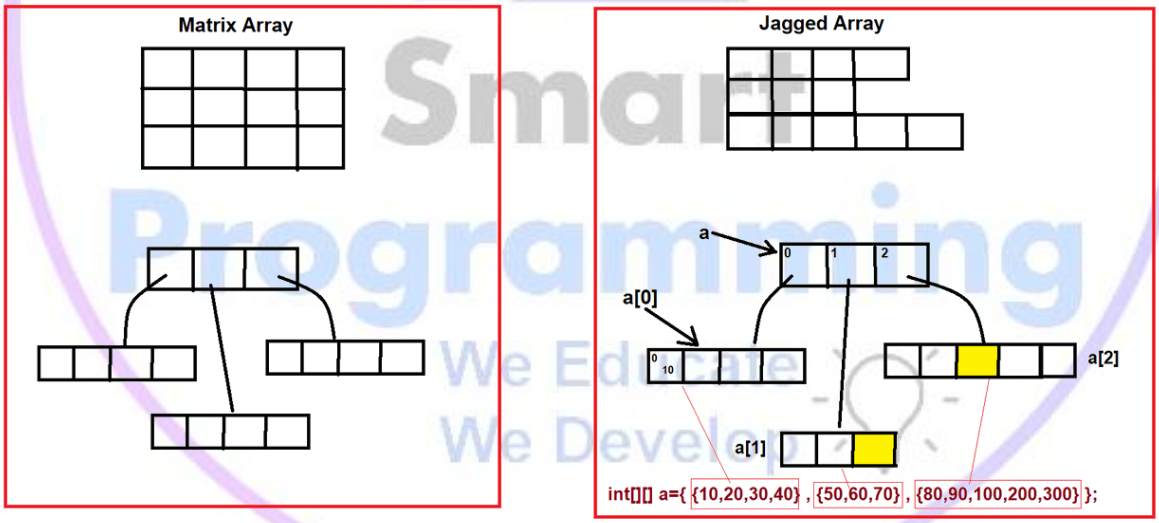
1. Single Dimensional Array
   -> 1D Array
2. Multi-Dimensional Array
   -> 2D Array
   -> 3D Array
   -> 4D, 5D, 6D.... Array
   -> Zic-Zac Array
3. Anonymous Array

=> Multi-Dimensional Array :

-> Multi-Dimensional array can be Matrix Array or Jagged Array

-> Matrix Array : The array in which there is fixed number of rows & columns

-> Jagged Array : The array in which number of columns can be changed



=> Anonymous Array :-

-> An array that does not have any name is known as Anonymous Array

-> For example :

new int[]{10,20,30,40};

-> Use : Anonymous arrays are created when we want to use them instantly or only for single time

-> Anonymous arrays are used as an argument in the method

-> Anonymous array can be single dimensional or multi-dimensional array

=> Different ways to print/iterate an array elements :-

-> for loop

-> while loop

-> for-each loop

```
for(type var:array)
{
    //print(var)
}
```

-> Limitations of for-each loop :-

1. It does not track the index position of an array

2. It only iterates the elements in forward direction

3. It cannot be used to modify the array

=> Arrays class :-

-> Java has created an "Arrays" class in "java.util" package

-> Arrays class directly inherits the Object class

-> "Arrays" class provides the static methods only

=> Types Of Data Structure :-

1. Primitive Data Structure

Examples are : boolean, char, byte, short, int, long, float, double

2. Non-Primitive Data Structure

Examples are : String, Files, Arrays, Collection Framework (Collection, Map)

=> Applications of Arrays :-

1. Arrays are used to store multiple data with single variable name which reduces the number of variables to be declared

2. Arrays are used to develop some algorithms like Bubble sort, Insertion sort, Selection sort etc

3. Arrays can be used to perform matrix operations

4. Arrays can be used for CPU scheduling

5. Array are used to implement data structure for example Stack, ArrayList, Queues etc

---

=> Point to remember for Array & Collection Framework :-

1. -> Array is java language feature inbuilt support provided by Sun Microsystems. We have to develop algorithms to sort or insert or delete etc

-> Collection Framework are API feature. It provides predefined classes and interfaces and methods by which we can easily iterate or delete or sort the elements

2. -> Array can store primitive (int, char etc) and non-primitive (objects) data types

-> Collection Framework can store only non-primitive data types (objects)

3. -> Array can store only homogeneous data types i.e. array can store only similar type of data

-> Collection Framework can store hetrogeneous data i.e. we can store different type of data

4. -> The size of an array cannot be increased or decreased according to our requirement at runtime

-> The size of collection can be increased or decreased according to our needs

5. -> Array are not good with respect to memory

-> Collection framework are very good with respect to memory

6. -> Arrays are good by performance wise

-> Collection are not good by performance wise

=> What is Collection Framework ?

-> Collection Framework consists of 2 words i.e. Collection and Framework

= Collection is a single entity or an object which contains multiple data

= Framework represents the library

-> Collection framework is the set of classes and interfaces that implement commonly reusable collection data structure

-> Collection framework contains 2 main parts :-

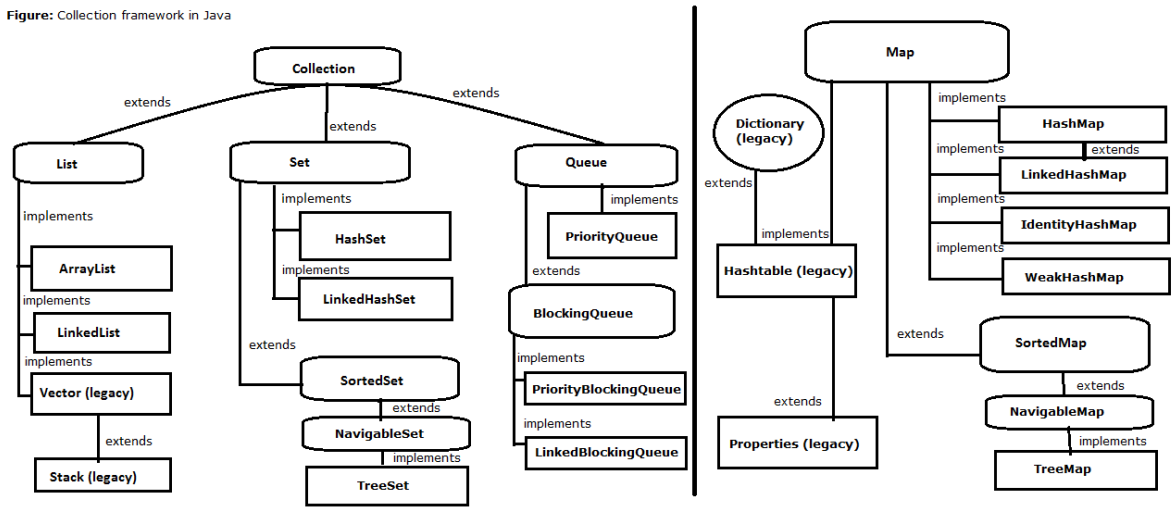
1. java.util.Collection
2. java.util.Map

-> "9 key interfaces" of Collection Framework

1. Collection
2. List
3. Set
4. SortedSet
5. NavigableSet
6. Queue

7. Map
8. SortedMap
9. NavigableMap

-> Hierarchy of Collection Framework :-



(image copyright to https://www.benchresources.net/collection-framework-in-java/)

-> In Collection, we can store the data directly but in Map we can store the data in key-value pairs

=> Collection :-

-> Collection is an interface which is present in java.util package

-> Syntax : public interface Collection<E> extends Iterable<E> { - }

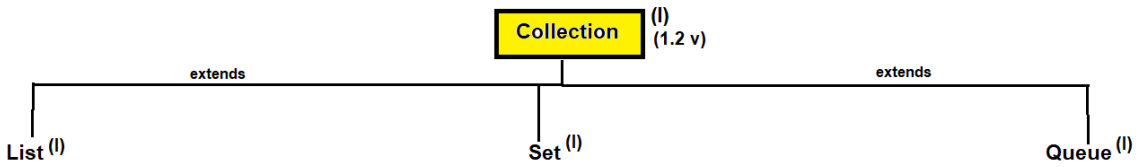
-> Collection was introduced in JDK 1.2 version

-> Collection is an object which is used to represent a group of individual objects as a single unit

-> Collection interface is the root interface of Collection Framework

-> There is no concrete class which implements the Collection interface directly but there are interfaces which inherit the Collection interface i.e. List, Set & Queue

-> Hierarchy of Collection interface :-



-> Collection interface contains most common methods which are applicable for any collection object

-> Methods of Collection Interface :-

1. boolean add(Object obj);
2. boolean addAll(Collection c);
3. boolean remove(Object obj);
4. boolean removeAll(Collection c);
5. default boolean removeIf(-) { - }
6. boolean retainAll(Collection c);

7. void clear();

8. boolean contains(Object obj);
9. boolean containsAll(Collection c);

10. boolean isEmpty();
11. int size();
12. Iterator iterator();
13. Object toArray();

14. boolean equals(Object obj);
15. int hashCode();

=> What is difference between Collection & Collections

1. -> Collection is an interface

   -> Collections is a utility class

2. -> Collection is an object which is used to represent a group of individual objects as a single unit

   -> Collections defines several utility methods that are used to operate on collection objects like sorting, searching etc

3. -> Collection interface contains default, abstract methods and static methods

   -> Collections class contains only static methods

=> What is Utility Class in Java ?

-> Utility class is also known as helper class which cannot be instantiated

-> Utility class contains only static methods

-> Examples are Arrays, Collections

-> How we can create utility class :-

1. declare the class as public and final

2. we have to declare private constructor to prevent object creation

3. class should contain only static methods and does not contain abstract methods

4. every method should have deep documentation

=> What is Utility Methods ?

-> Utility methods perform common, often reused methods.

-> Utility methods are always static type

-> Examples are sorting, searching, methods performing string manipulation, methods connecting to databases etc

=> List Interface :-

➔ List is a interface which is present in java.util package
➔ List is the child interface of Collection interface
➔ Syntax : public interface List extends Collection { - }
➔ List was introduced in JDK 1.2 version
➔ Hierarchy of List interface :-

➔ Properties of List Interface :-

1. List is an index based Data Structure which means that first element will be inserted at 0 index position
2. List can store different data types or heterogeneous elements
3. We can store duplicate elements in the List
4. We can store any number of null values in the List
5. List follows the insertion order which means the sequence in which we are inserting the

elements, in the same sequence we can retrieve the elements

6. List does not follow the sorting order

➔ Methods of List Interface :-

1. List contains all the methods of Collection interface

2. void add(int index, Object obj);
3. boolean addAll(int index, Collection c);
4. Object get(int index);
5. Object remove(int index);
7. 6. Object set(int index, Object newobj); //set method is used to replace the object at given index position
8. 7. int indexOf(Object obj); //it will return the index position of provided object and if object is not found then it will return -1
9. int lastIndexOf(Object obj);

=> ArrayList :-

- ➔ ArrayList is an implemented class of List interface which is present in java.util package
- ➔ Syntax : public class ArrayList extends AbstractList implements List, RandomAccess, Cloneable, Serializable
- ➔ The underline Data-Structure of ArrayList is resizable array or growable array
- ➔ ArrayList was introduced in JDK 1.2 version

➔ Properties of ArrayList :-

1. ArrayList is an index based Data Structure which means that first element will be inserted at 0 index position
2. ArrayList can store different data types elements or hetrogeneous elements
3. We can store duplicate elements in the ArrayList
4. We can store any number of null values in the ArrayList

5. ArrayList follows the insertion order which means the sequence in which we are inserting the elements, in the same sequence we can retrieve the elements
6. ArrayList does not follow the sorting order (above properties are same as List interface)

7. ArrayList is non-synchronized collection because ArrayList does not contain any synchronized method
8. ArrayList allows more than one thread at one time
9. ArrayList allows parallel execution
10. ArrayList reduces the execution time which in turn makes the application fast
11. ArrayList is not threadsafe
12. ArrayList does not gurantee for data consistency

➔ Working of an ArrayList :-

1. When we create default ArrayList, a new ArrayList with initial capacity 10 is created (but size is 0)
2. When the ArrayList capacity is full, a new ArrayList will be created with new capacity. The new Capacity is calculated by this formula:-
   a. (CurrentCapacity * 3 / 2) + 1
3. Then all the elements will be copied into the new ArrayList (and due this this reason performance of an ArrayList decreases)
4. When new ArrayList is created automatically, then reference variable will point to the new ArrayList
5. Then old ArrayList object will be not referenced by any reference and then garbage collection will delete that object

Note : There is no way by which we can find the capacity of an ArrayList

➔ Contructors of ArrayList :-

1. ArrayList al=new ArrayList();
   a. = In this arraylist, an ArrayList collection object is created whose capacity is 10
2. ArrayList al=new ArrayList(int initialCapacity);
   a. = In this arraylist, an ArrayList object is created with provided initialCapacity
3. ArrayList al=new ArrayList(Collection c);
   a. = In this arraylist, another collection object is copied into new arraylist object

➔ When we should use ArrayList ?

= When we use retrival operation mostly (Retrival operation is fast in case of ArrayList because it implements RandomAccess interface)

➔ When we should not use ArrayList ?

= When we have mostly insertion or deletion operation, then we should not use ArrayList

=> RandomAccess interface :-

-> RandomAccess interface is a marker interface that means it does not contain any methods or fields (variables)

-> The purpose of RandomAccess interface is to retrieve any random element in collection object at the same speed. For example we have collection object having 1 crore elements, we have to search 3rd element or middle element or last element then it will search with the same speed

-> There are only 2 classes which inherits the RandomAccess interface

1. ArrayList
2. Vector

=> Cloneable Interface :-

-> Cloneable interface is also a marker interface

-> It was introduced in JDK 1.0 version

-> It is used to clone the object without using the new keyword

---

=> LinkedList :-

- ➔ LinkedList is an implementation class of List interface which is present in java.util package
- ➔ Syntax : public class LinkedList extends AbstractSequentialList implements List, Deque, Cloneable, Serializable { - }
- ➔ The underline data structure of LinkedList is Double Linked List or Circular Linked List
- ➔ LinkedList was introduced in JDK 1.2 version

- ➔ Properties of LinkedList :-

1. LinkedList is an index based Data Structure which means that first element will be inserted at 0 index position
2. LinkedList can store different data types or hetrogeneous elements
3. We can store duplicate elements in the LinkedList
4. We can store any number of null values in the LinkedList

5. LinkedList follows the insertion order which means the sequence in which we are inserting the elements, in the same sequence we can retrieve the elements
6. LinkedList does not follow the sorting order

(above properties are same as List interface)

7. LinkedList is non-synchronized collection because LinkedList does not contain any synchronized method
8. LinkedList allows more than one thread at one time
9. LinkedList allows parallel execution
10. LinkedList reduces the execution time which in turn makes the application fast
11. LinkedList is not threadsafe
12. LinkedList does not gurantee for data consistency

-> Working of LinkedList :-

1. Types of LinkedList (all programming languages) :-

    a. Single Linked List

    b. Double Linked List

    c. Circular Linked List

2. Linked List are linear data structure in which elements are not stored in contigeous memory locations.

3. There is no capacity concept in LinkedList like ArrayList

-> Constructors :-

1. public LinkedList()

2. public LinkedList(Collection c)

-> Methods of LinkedList :-

1. Methods of Collection Interface
2. Methods of List Interface

3. public void addFirst(Object obj)
4. public void addLast(Object obj)

5. public Object getFirst()
6. public Object getLast()

7. public Object removeFirst()
8. public Object removeLast()

-> When we should use LinkedList ?

= LinkedList is best when we have to use insertion or deletion operations

-> When we should not use LinkedList ?

= LinkedList is worst in case of retrieval or searching operations (as LinkedList does not inherit RandomAccess interface)

-> What is difference between ArrayList & LinkedList ?

1. ArrayList underline data structure is dynamic array or resizable array

LinkedList underline data structure is double linked list or circular linked list

2. ArrayList stores the elements in contigeous memory locations

LinkedList does not store the elements in contigeous memory locations

3. ArrayList acts as List

   LinkedList can acts as List or Deque

4. ArrayList is good in case of retrieval operations

   LinkedList is good in case of insertion and deletion operations

5. ArrayList is worst in case of insertion or deletion operations

   LinkedList is worst in case of retrieval operations

=> Vector :-

- ➔ Vector is an implementation class of List interface which is present in java.util package
- ➔ Syntax : public class Vector extends AbstractList implements List, RandomAccess, Cloneable, Serializable { - }
- ➔ The underline data structure of Vector is resizable array or growable array
- ➔ Vector was introduced in JDK 1.0 version
- ➔ Vector class is also known as legacy class. (Legacy class is the class which was formed in previous version and was restructured or re-engineered in new version)

-> Properties of Vector :-

1. Vector is an index based Data Structure which means that first element will be inserted at 0 index position
2. Vector can store different data types elements or hetrogeneous elements
3. We can store duplicate elements in the Vector

4. We can store any number of null values in the Vector
5. Vector follows the insertion order which means the sequence in which we are inserting the elements, in the same sequence we can retrieve the elements
6. Vector does not follow the sorting order (above properties are same as List interface)

7. Vector is synchronized collection because Vector contains many synchronized method
8. Vector does not allow more than one thread at one time
9. Vector does not allow parallel execution or Vector allows sequential execution
10. Vector increases the execution time which in turn makes the application slow
11. Vector is threadsafe
12. Vector gurantee for data consistency

-> Working of Vector :-

1. When we create a vector, a vector of 10 initial capacity is created
2. When the vector is full, then new vector will be created automatically with new capacity = current capacity * 2;
3. When new vector is created then all the elements from old vector will copied to new vector and then the reference variable will point to the new vector and garbage collector will delete the previous vector from the memory

Note : In vector we can find the capacity

-> Constructors of Vector :-

1. public Vector()
2. public Vector(int capacity)
3. public Vector(int capacity, int incremental_ratio)
4. public Vector(Collection c)

-> Methods :-

1. Vector contains all the methods of Collection interface

2. Vector contains all the methods of List interface

1. public synchronized int capacity()

2. public synchronized void addElement(Object obj)

3. public synchronized Object firstElement()

4. public synchronized Object lastElement()

5. public synchronized boolean removeElement(Object obj)

6. public synchronized void removeElementAt(int index)

7. public synchronized void removeAllElements()

-> When we should use Vector ?

= We should use Vector in case of retrieval or searching operations

(Vector inherits the RandomAccess interface)

-> When we should not use Vector ?

= We should not use Vector in case of insertion or deletion of elements

-> What is difference between ArrayList & Vector :-

1. ArrayList was introduced in JDK 1.2 version

   Vector was introduced in JDK 1.0 version

2. ArrayList is not a legacy class

   Vector is legacy class

3. ArrayList is non-synchronized collection

Vector is synchronized collection

4. ArrayList allows more than one thread at one time

Vector does not allow more than one thread at one time

5. ArrayList allows the parallel execution

Vector does not allow parallel execution

6. ArrayList decreases the execution time which in turn makes the application fast

Vector increases the execution time which in turn makes the application slow

7. ArrayList is not threadsafe

Vector is threadsafe

8. ArrayList does not gurantee for data consistency

Vector gurantee for data consistency

9. In case of ArrayList new capacity = (present capacity * 3/2)+1

In case of Vector new capacity = present capacity * 2

10. In ArrayList we cannot find the capacity

In Vector we can find the capacity

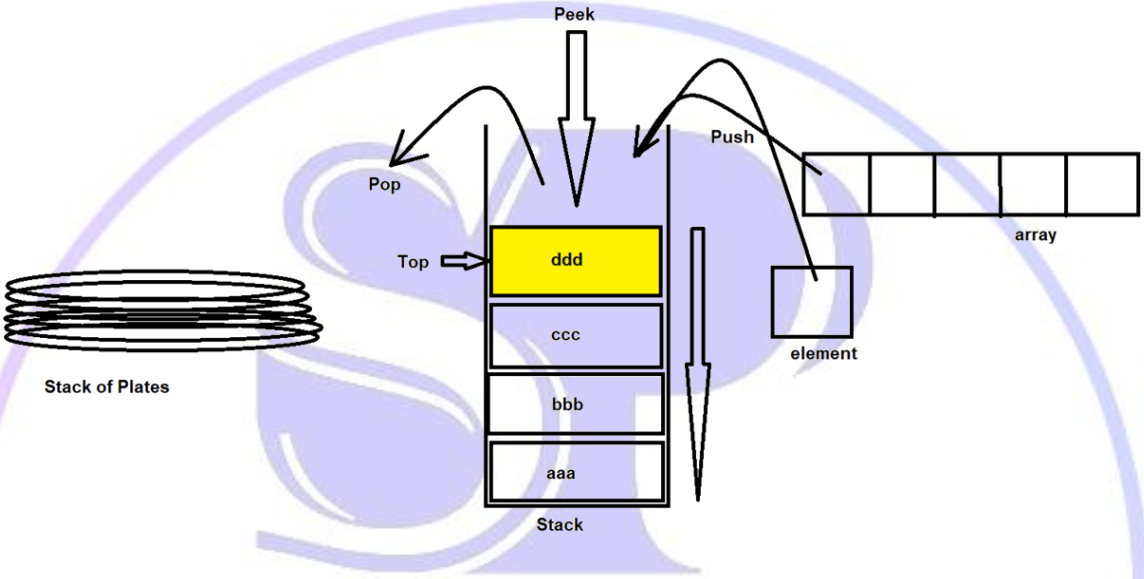
=> Stack :-

➔ Stack is the child class of Vector class present in java.util package
➔ Syntax : public class Stack extends Vector { - }
➔ Stack was introduced in JDK 1.0 version
➔ Stack is also known as legacy class (legacy class is the class which was re-engineered or restrctured in newer version)

-> Properties of Stack :-

1. Stack class was specially designed for Last In First Out (LIFO) but we can change this algorithm according to our requirement
2. Stack can be implemented using Array, ArrayList, LinkedList or Vector
3. Stack is also index based data structure

-> Working of Stack :-



In java stack is implemented by LIFO (Last In First Out) algorithm

-> Constructors :-

1. public Stack() { - }

-> Methods :-

1. Methods of Vector, List and Collections

2. public Object push(Object obj)
3. public synchronized Object pop()
4. public synchronized Object peek()
5. public synchronized int search(Object obj)
6. public boolean empty()

=> Cursors :-

➔ In java, whenever we print the object reference, internally JVM will call toString() method of Object class. In case of simple object it will print ClassName@referencevalue but in case of printing Collection object it will print the elements present in Collection object.

➔ When we print the collection object it will retrieve all the elements at one time but we want to retrieve the elements one by one then we have to use cursors

-> Types of cursors :-

1. Enumeration
2. Iterator
3. ListIterator

=> Enumeration :-

➔ Enumeration is the cursor which is used to get the elements one by one from the collection object

➔ Enumeration was introduced in JDK 1.0 version

➔ Enumeration is used only for legacy class

-> Steps "how to use" enumeration cursor :-

1. Create Enumeration Cursor Object

-> public Enumeration elements()

(this method is present in Vector & Stack legacy class)

2. Read one by one all the elements from Enumeration Cursor

-> public boolean hasMoreElements()

-> public Object nextElement()

(these methods are present in Enumeration interface)

-> Limitations of Enumeration :-

1. It can be used only with Legacy class and thus it is not universal cursor
2. By using enumeration cursor we can only perform read operation but not update or remove operation
3. It can be used to traverse the elements only in forward direction

---

=> Iterator :-

- ➔ Iterator is a cursor which is used to get the elements one by one from the collection object
- ➔ It is universal cursor which means that we can use it with all collection objects
- ➔ It can be used for read and remove operation
- ➔ It was introduced in JDK 1.2 version

-> Steps "how to use" iterator cursor :-

1. Create Iterator cursor object :-

-> public Iterator iterator()

(this method is present for every collection object)

2. Read one by one all the elements from iterator cursor :-

-> public boolean hasNext()

-> public Object next()

-> public void remove()

(these methods are present in Iterator interface)

-> Limitations of Iterator cursor :-

1. It can be used only for read and remove operation but not for replacement or addition operation

2. It can be used to iterate the elements only in forward direction

---

=> ListIterator :-

- ➔ ListIterator is a cursor which is used to get the elements one by one from collection object
- ➔ ListIterator is bi-directional cursor which means it can be used to traverse the elements in forward or backword direction
- ➔ It can be used to read, remove, insert and replace operations
- ➔ It was introduced in JDK 1.2 version

-> Steps "how to use" ListIterator cursor :-

1. Create ListIterator cursor object :-

-> public ListIterator listIterator();

(which is present in only List implementation classes)

2. Read one by one all the elements from ListIterator cursor :-

-> public boolean hasNext()

-> public Object next()

-> public int nextIndex()

(above methods are used to traverse the elements in forward direction)

-> public boolean hasPrevious()

-> public Object previous()

-> public int previousIndex()

(above methods are used to traverse the elements in backword direction)

-> public void remove()

-> public void add(Object obj)

-> public void set(Object obj)

(above methods are used to remove, add and replace operations)

-> Limitations of ListIterator cursor :-

1. It can be used only with List implemented classes thus it is not universal cursor

=> What is difference between Enumeration, Iterator & ListIterator :-

1. Enumeration can be used only with legacy classes

Iterator can be used for any collection object

ListIterator can be used only for List implemented classes

2. Enumeration can be used to traverse the elements only in forward direction

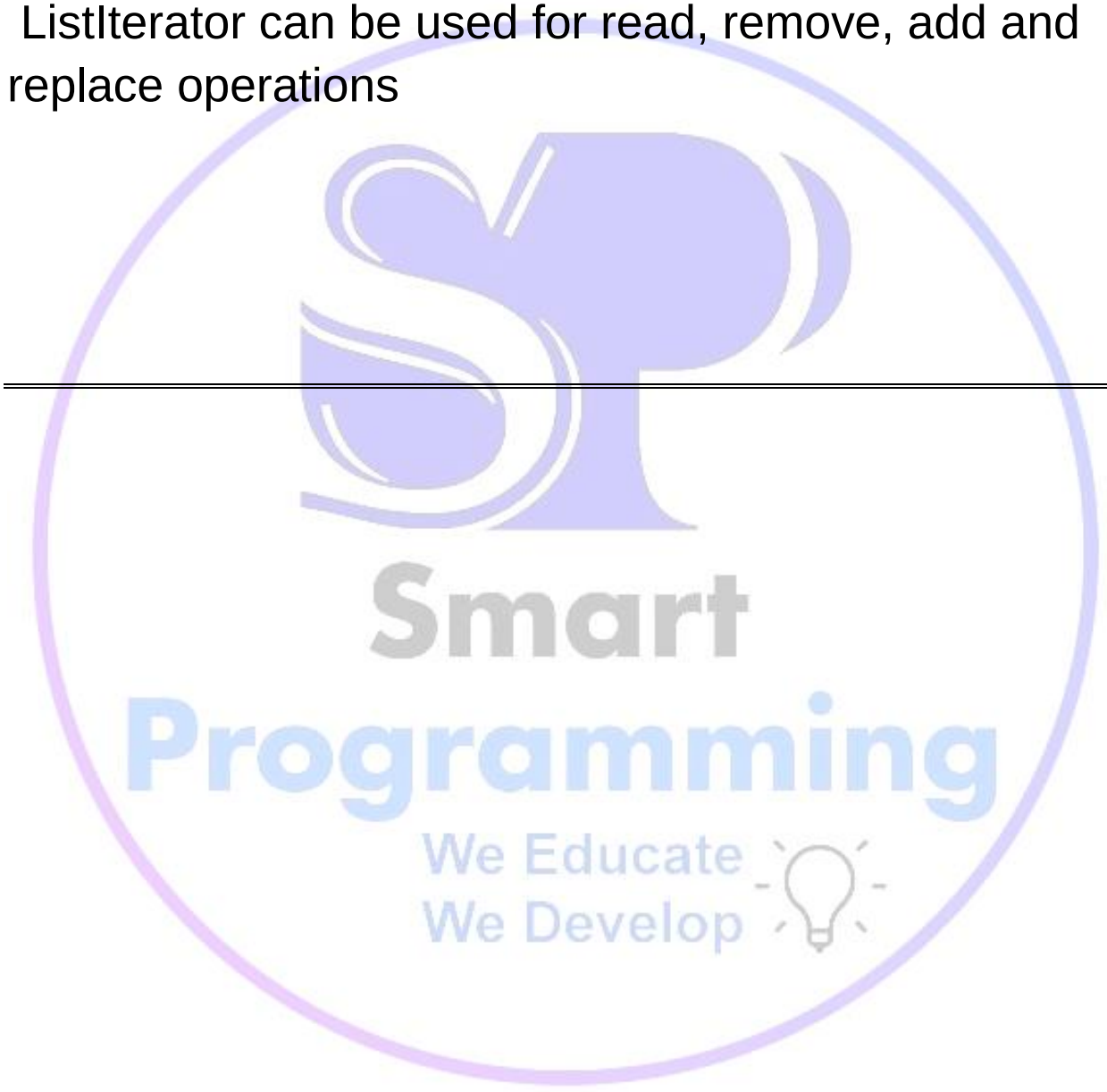
Iterator can be used to traverse the elements only in forward direction

ListIterator can be used to traverse the elements in forward and backword direction

3. Enumeration is used only for read operation

Iterator can be used for read and remove operation

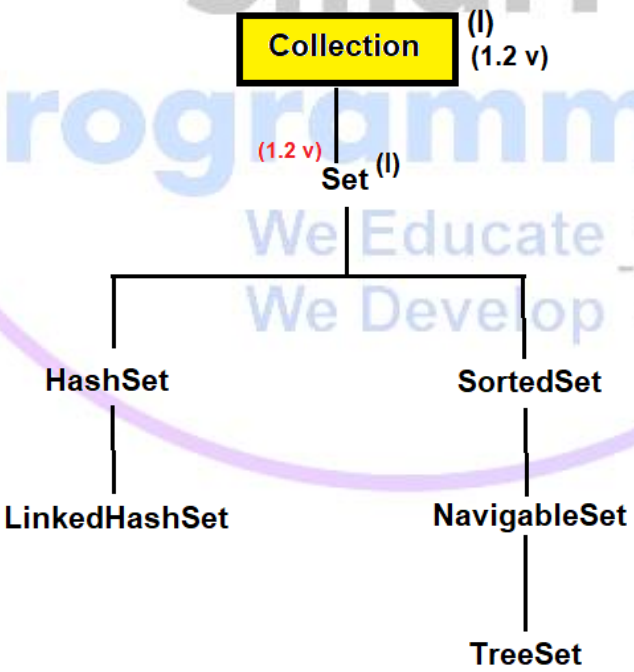
ListIterator can be used for read, remove, add and replace operations



=> Set :-

- ➔ Set is an interface which is present in java.util package
- ➔ Set is the child interface of Collection interface
- ➔ Syntax : public interface Set extends Collection { - }
- ➔ Set was introduced in JDK 1.2 version

- ➔ Hierarchy of Set interface :-



-> Properties of Set interface :-

1. Set is not an index based data structure, it stores the elements as per elements "hashcode" values
2. Set does not follow the insertion order (except LinkedHashSet)
3. Set does not follow the sorting order (except SortedSet, NavigableSet & TreeSet)
4. Set can store different data types or hetrogeneous elements (except SortedSet, NavigableSet, TreeSet)
5. We cannot store duplicate elements in Set
6. We can store only one null value in Set

-> Methods of Set interface :-

= approx same methods as that of Collection interface

=> Difference between List & Set :-

1. List is index based data structure which means in list data is stored by using index position

   Set is not an index based data structure, it stores the data according to hashcode values of the elements

2. List allows duplicate elements

   Set does not allow duplicate elements

3. List can store any number of null values

   Set can store only one null value

4. List follows the insertion order

   Set does not follows the insertion order

5. In case of List we can use Iterator & ListIterator cursor

In case of Set we can use only Iterator cursor

6. List is used in case of retrieveing the elements

Set is used when we does not want to allow duplicacy

=> HashSet :-

- ➔ HashSet is an implemented class of Set interface which is present in java.util package
- ➔ Syntax : public class HashSet extends AbstractSet implements Set, Cloneable, Serializable { - }

➔ The underline data structure of HashSet is Hashtable (HashSet is backed up by Map)
➔ HashSet was introduced in JDK 1.2 version

➔ Properties of HashSet :-

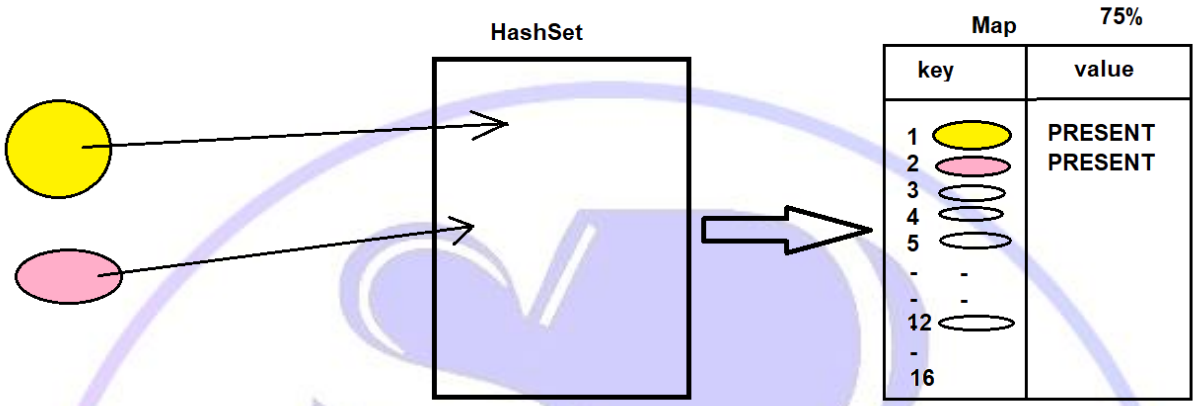
1. HashSet is not an index based data structure, it stores the elements according to elements hashcode values
2. HashSet can store different data types of hetrogeneous elements
3. HashSet cannot store the duplicate elements
4. HashSet can store maximum only one null value
5. HashSet does not follows the insertion order
6. HashSet does not follow the sorting order
   a. (same properties as Set interface)

7. HashSet is non-synchronized collection because HashSet does not contain any synchronized methods

8. HashSet allows more than one thread at one time
9. HashSet allows the parallel execution
10. HashSet reduces the execution time which in turn makes our application fast
11. HashSet is not threadsafe
12. HashSet does not gurantee for data consistency

-> Working of HashSet :-

1. HashSet internally works on the basis of Hashtable (it internally backed up by Map.)
2. When we insert any element in HashSet, it stores as a key inside the Map and at value position PRESENT reference variable is stored which is the reference variable of Object class
3. Initial capacity of HashSet is 16 elements
4. Its load factor or fill ratio is 75%
5. After load is filled then a new Map is created with its double capacity



-> Constructors of HashSet :-

1. public HashSet()
2. public HashSet(Object obj)
3. public HashSet(int initialCapacity)
4. public HashSet(int initialCapacity, float loadFactor)

-> Methods of HashSet :-

= approx same methods as that of Collection or Set interface

-> When we should use HashSet ?

= HashSet is good for searching or retriving operations

=> LinkedHashSet :-

➔ LinkedHashSet is the child class of HashSet which is present in java.util package

➔ Syntax : public class LinkedHashSet extends HashSet implements Set, Cloneable, Serializable { - }

➔ The underline data structure of LinkedHashSet is "Hashtable + LinkedList"

➔ LinkedHashSet was introduced in JDK 1.4 version

➔ Properties of LinkedHashSet :-

= All properties of LinkedHashSet is same as HashSet except LinkedHashSet follows the insertion order

➔ Constructors of LinkedHashSet :-

= same constructors as HashSet

➔ Methods of LinkedHashSet :-

= Same methods as HashSet

➔ When we should use LinkedHashSet :-

= If we have to create cache based applications then we can use LinkedHashSet

➔ Difference between HashSet & LinkedHashSet :-

1. HashSet was introduced in JDK 1.2 version

LinkedHashSet was introduced in J2SE 1.4 version

2. HashSet does not follow the insertion order

LinkedHashSet follows the insertion order

3. HashSet underline data structure is "Hashtable"

LinkedHashSet underline data structure is "Hashtable + LinkedList"

=> SortedSet :-

- ➔ SortedSet is a child interface of Set interface which is present in java.util package
- ➔ Syntax : public interface SortedSet extends Set { - }
- ➔ SortedSet was introduced in JDK 1.2 version

➔ Properties of SortedSet :-

1. SortedSet is not an index based data structure
2. SortedSet does not follows the insertion order
3. SortedSet follows the sorting order
4. SortedSet can store same data types or homogeneous elements. If we provide different data type element if will provide exception i.e. java.lang.ClassCastException
5. SortedSet cannot store the duplicate elements
6. We should not store null value in SortedSet because SortedSet follows the sorting order so while comparing the elements with null value, it will provide the java.lang.NullPointerException
7. SortedSet allows Comparable objects bydefault, but if we insert non-comparable objects, then it will provide an exception i.e. java.lang.ClassCastException

8. SortedSet is non-synchronized collection

➔ Methods of SortedSet :-

1. Object first();
2. Object last();
3. SortedSet headSet(Object toElement);
4. SortedSet tailSet(Object fromElement);
5. SortedSet subSet(Object fromElement, Object toElement);

=> NavigableSet :-

➔ NavigableSet is the child interface of SortedSet interface which is present in java.util package

➔ Syntax : public interface NavigableSet extends SortedSet { - }

➔ NavigableSet was introduced in Java SE 6 version

➔ Properties of NavigableSet :-

= NavigableSet has the same properties as that of SortedSet but it provides some extra navigable methods

➔ Methods :

1. public NavigableSet descendingSet()
2. public Object ceiling(Object obj)
3. public Object higher(Object obj)
4. public Object floor(Object obj)
5. public Object lower(Object obj)
6. public Object pollFirst()
7. public Object pollLast()

=> TreeSet :-

- ➔ TreeSet is the direct implementation for NavigableSet interface but it also provides the implementation for SortedSet, Set & Collection interface
- ➔ Syntax : public class TreeSet extends AbstractSet implements NavigableSet, Cloneable, Serializable { - }
- ➔ The underline data structure of TreeSet is "Balanced Tree"
- ➔ TreeSet was introduced in JDK 1.2 version

➔ Properties of TreeSet :-

1. TreeSet is not index based data structure
2. TreeSet does not follows the insertion order
3. TreeSet follows the sorting order, it may be default sorting order or customized sorting order
4. TreeSet stores the same data type elements or homogeneous elements. If we provide different data type elements in TreeSet then it will provide java.lang.ClassCastException

5. TreeSet cannot stores the duplicate elements
6. TreeSet can store only one null value.

7. TreeSet is non-synchronized collection
8. TreeSet allows more than one thread at one time
9. TreeSet allows the parallel execution
10. TreeSet reduces the execution time which makes our application fast
11. TreeSet is not threadsafe
12. TreeSet does not provide gurantee for data consistency

➔ **Working of TreeSet :-**

1. When we insert the first element, it will become the root node.
2. Then when we insert next element, it will compare with the root node ("next element".compareTo("root element");), if the return value is -ve, then it will goes to the left

side and if the return value is +ve then it will goes to the right side

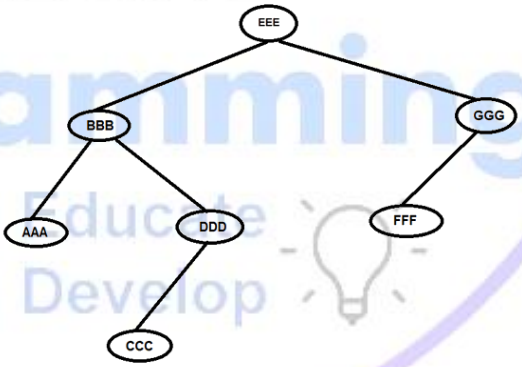
3. Then again it will check wether there is any parent node or not, if there is any parent node, then again it will compare and check the return value, then if return value is +ve, it will goes to right side and if return value is -ve, it will goes to the left side

4. When we retrieve the elements, it will retrieve as "LEFT - ROOT - RIGHT"

```
TreeSet ts=new TreeSet();

ts.add("EEE");  //first element will be inserted as root element
ts.add("GGG");  //"GGG".compareTo("EEE")          +ve
ts.add("BBB");  //"BBB".compareTo("EEE");         -ve
ts.add("DDD");  //"DDD".compareTo("EEE");         -ve
                //"DDD".compareTo("BBB");         +ve
ts.add("CCC");  //"CCC".compareTo("EEE");         -ve
                //"CCC".compareTo("BBB");         +ve
                //"CCC".compareTo("DDD");         -ve
ts.add("AAA");  //"AAA".compareTo("EEE");         -ve
                //"AAA".compareTo("BBB");         -ve
ts.add("FFF");  //"FFF".compareTo("EEE");         +ve
                //"FFF".compareTo("GGG");         -ve

System.out.println(ts);   [AAA, BBB, CCC, DDD, EEE, FFF, GGG]
```



➔ Constructors :

1. public TreeSet() //it will create an empty TreeSet object where the elements are inserted according to natural default sorting order
2. 2. public TreeSet(Comparator comparator) // it will create an empty TreeSet object where the elements will be inserted according to the customized sorting order
3. public TreeSet(Collection c) // we can pass any other collection object
4. public TreeSet(SortedSet s)

-> Methods of TreeSet :-

= contains all the methods of Collection, Set, SortedSet & NavigableSet interfaces

-> When we should use TreeSet :-

= When we want to store the large number of elements in sorting order and due to this retrival operation is fast

=> Cases for "null" value insertion in TreeSet :-

1. We can store only one null value

2. We can insert null value only at first position, but then if we insert any other element then it will provide "java.lang.NullPointerException"

3. NOTE : Until 1.6 version we can successfully insert the null value at first position but after 1.6 version we cannot store the null value even at first position

=> What is difference between HashSet, LinkedHashSet & TreeSet :-

1. HashSet underline data structure is "Hashtable"

LinkedHashSet underline data structure is "Hashtable + Linked List"

TreeSet underline data structure is "Balanced Tree"

2. HashSet does not allow insertion order

LinkedHashSet allow the insertion order

TreeSet does not allow the insertion order

3. HashSet does not allow the sorting order

LinkedHashSet does not allow sorting order

TreeSet allows the sorting order

4. HashSet allows the hetrogeneous objects

LinkedHashSet allows the hetrogeneous objects

TreeSet does not allow the hetrogeneous objects

5. HashSet allows the null value insertion

LinkedHashSet allows the null value insertion

TreeSet allows the null value but at first position (but this is applicable till java 1.6 version)

=> Comparable :

➔ Comparable is an interface which is present in java.lang package

➔ It contains only one method i.e. compareTo()

- Prototype : public int compareTo(Object obj)
  - obj1.compareTo(obj2);
  - +ve - if obj1 is greater than obj2
  - -ve - if obj2 is greater than obj1
  - 0 - if obj1 is equals to obj2

➔ String and all wrapper classes (Integer, Long, Float etc) implements Comparable interface

➔ Problems with Comparable interface :-

1. By implementing Comparable interface, the properties of Original class will get changed
2. By this way we can sort only for one entity for example we can sort the student object either with name or rollno at one time

(To remove above problems java provided one interface i.e. Comparator)

=> Comparator :-

-> Comparator is an interface which is present in java.util package

-> Comparator interface contains 2 methods :-

1. public int compare(Object obj1, Object obj2)
2. public boolean equals(Object obj)

=> TASK : WAP to compare and add different elements according to length and alphabetical order

For example :

A, B, AA, BB, AAA, BBB, AAAA

=> What is difference between Comparable & Comparator interface :-

1. Comparable interface contains compareTo() method

   Comparator interface contains compare() method

2. Comparable interface is present in java.lang package

   Comparator interface is present in java.util package

3. By using Comparable interface original class properties will get changed

By using Comparator interface original class properties will not be changed

4. Comparable interface can sort only one entity

   Comparator interface can sort multiple entities

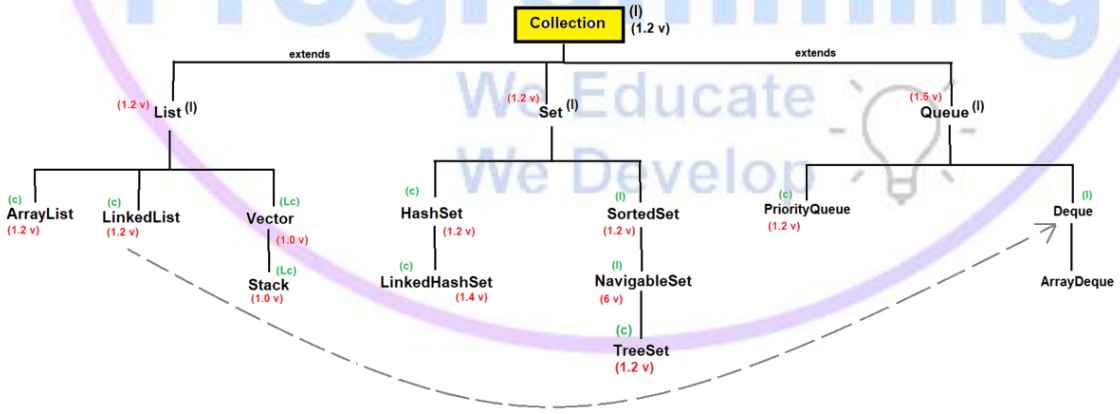
5. Comparable interface is used to implicit sorting

   Comparator interface is used for explicit sorting

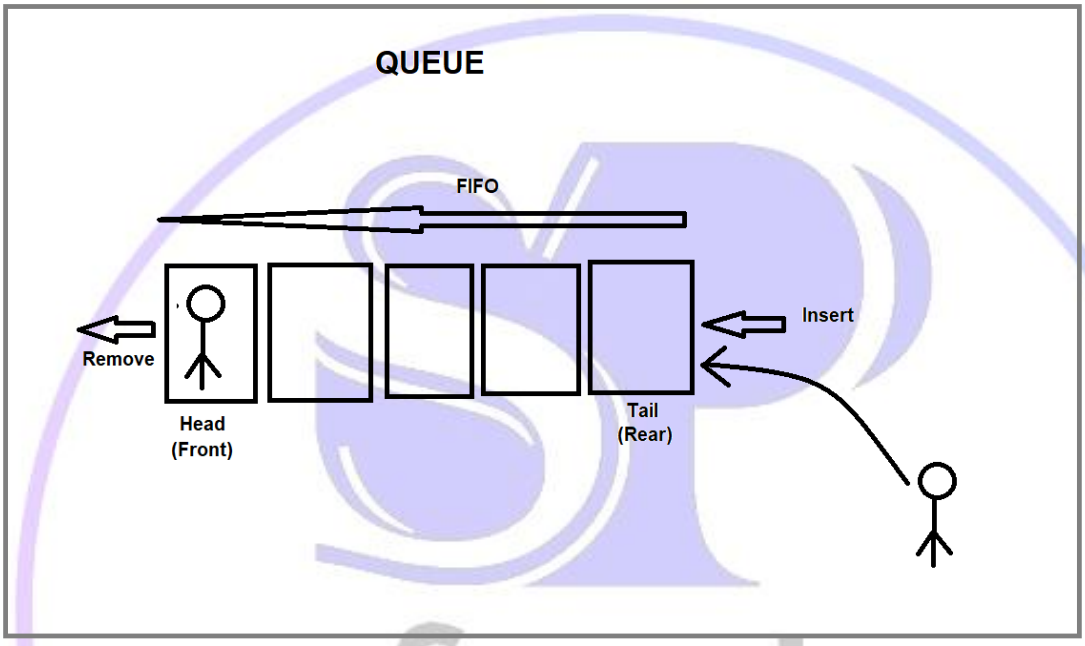
=> Queue :-

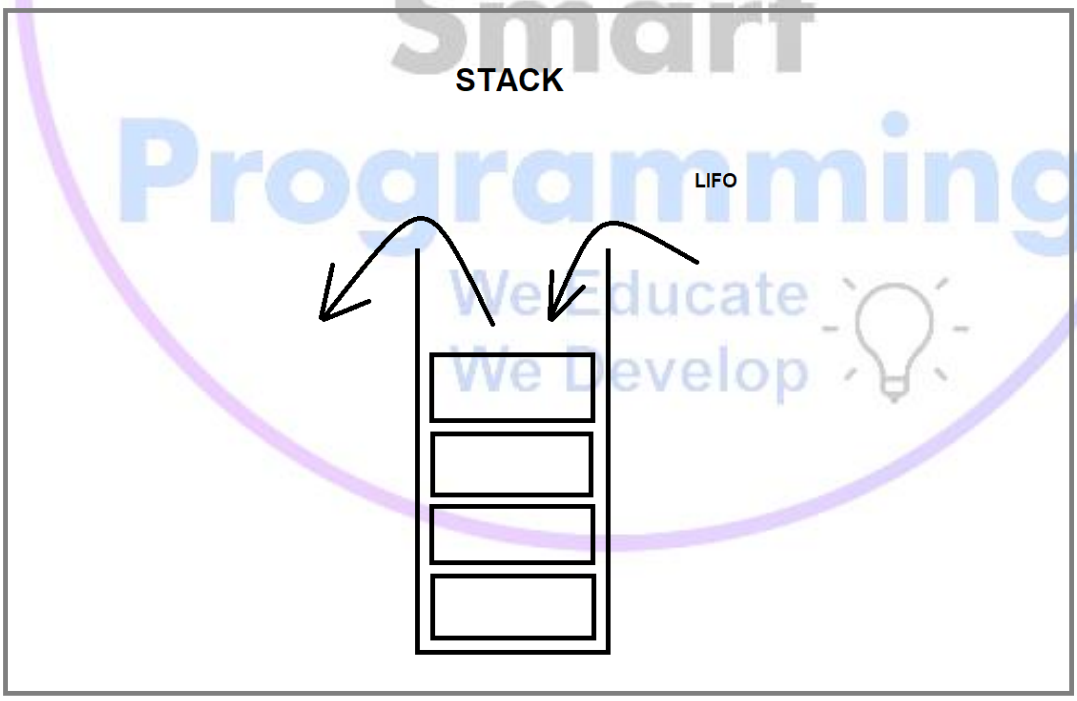
- ➔ Queue is the child interface of Collection interface
- ➔ Syntax : public interface Queue extends Collection { - }
- ➔ Queue was introduced in JDK 1.5 version
- ➔ Queue orders the elements in FIFO(First In First Out) manner, but we can change this algorithm according to our requirements

- ➔ Hierarchy of Queue :-



➔ Implementation of Queue :-





➔ Properties of Queue :-

1. Queue does not follow the insertion order
2. Queue follows the sorting order
3. Queue stores the same data type elements or homogeneous elements. If we try to store different elements then it will throw an exception saying "java.lang.ClassCastException"
4. Queue can store the duplicate elements
5. Queue does not store any null value. If we try to store null value then it will throw an exception i.e. "java.lang.NullPointerException"

➔ Methods of Queue :-

1. boolean offer(Object obj) - to add the elements in the queue
2. Object peek() - It will return the head element of the queue. If no element is found in the queue it will return null value
3. Object element() - It will return the head element of the queue. If no element is found, it

will throw an exception i.e. "java.util.NoSuchElementException"

4. Object poll() - It is used to remove the head element and also it will return that element, If no element is found, then it will return null value
5. Object remove() - It is used to remove the head element and also it will return that element. If no element is found, it will throw an exception i.e. "java.util.NoSuchElementException"

=> PriorityQueue :-

- ➔ PriorityQueue is an implementation class for Queue (but not direct implementation)
- ➔ Syntax : public class PriorityQueue extends AbstractQueue implements Serializable { - }
- ➔ PriorityQueue was introduced in JDK 1.2 version

➔ PriorityQueue may not support on windows platform

➔ It is able to process all the elements prior to processing as per priorities

➔ Properties of PriorityQueue :-

1. PriorityQueue does not follows the insertion order
2. PriorityQueue does not follows the sorting order
3. PriorityQueue stores the same data type elements or homogeneous elements. If we try to store different data type elements then it will throw an exception i.e. "java.lang.ClassCastException"
4. PriorityQueue can stores the duplicate elements
5. PriorityQueue cannot store the null values
6. PriorityQueue is non-synchronized collection

7. PriorityQueue allows more than one thread at one time
8. PriorityQueue allows the parallel execution
9. PriorityQueue reduces the execution time which makes our application fast
10. PriorityQueue is not thread-safe
11. PriorityQueue does not provide guarantee for data consistency

➔ Constructors :-

1. public PriorityQueue() - When we use default PriorityQueue constructor its initial capacity is 11
2. public PriorityQueue(int capacity)
3. public PriorityQueue(Comparator c)
4. public PriorityQueue(int capacity, Comparator c)
5. public PriorityQueue(SortedSet ss)
6. public PriorityQueue(PriorityQueue pq)
7. public PriorityQueue(Collection c)

➔ Methods of PriorityQueue :-

= Contains the methods of Queue and Collection interface

➔ When we should use PriorityQueue :-

= We can use PriorityQueue in SMS (JMS - Java Message Service), mail, offers, prime users etc

=> Deque :

➔ It is also known as "double ended queue"
➔ In Deque we can add or remove the elements on both side
➔ Deque is the child interface of Queue interface
➔ Syntax : public interface Deque extends Queue { - }

➔ Methods of Deque :-

1. void addFirst(Object e);
2. void addLast(Object e);
3. boolean offerFirst(Object e);
4. boolean offerLast(Object e);
5. Object removeFirst();
6. Object removeLast();
7. Object pollFirst();
8. Object pollLast();
9. Object getFirst();
10. Object getLast();
11. Object peekFirst();
12. Object peekLast();

=> ArrayDeque :-

- ➔ ArrayDeque is an implemented class for Deque interface
- ➔ Syntax : public class ArrayDeque extends AbstractCollection implements Deque, Cloneable, Serializable { - }
- ➔ ArrayDeque is used to provide the facility of Deque and Resizable-Array.

- ➔ Properties of ArrayDeque :-
- ➔ In this also we can add and remove the elements from both side
- ➔ Null is not allowed in ArrayDeque
- ➔ ArrayDeque is not synchronized collection
- ➔ ArrayDeque has no capacity concept

➔ Advantage :-

= ArrayDeque is faster as compared to LinkedList or Stack

=> Abstract Design Pattern :

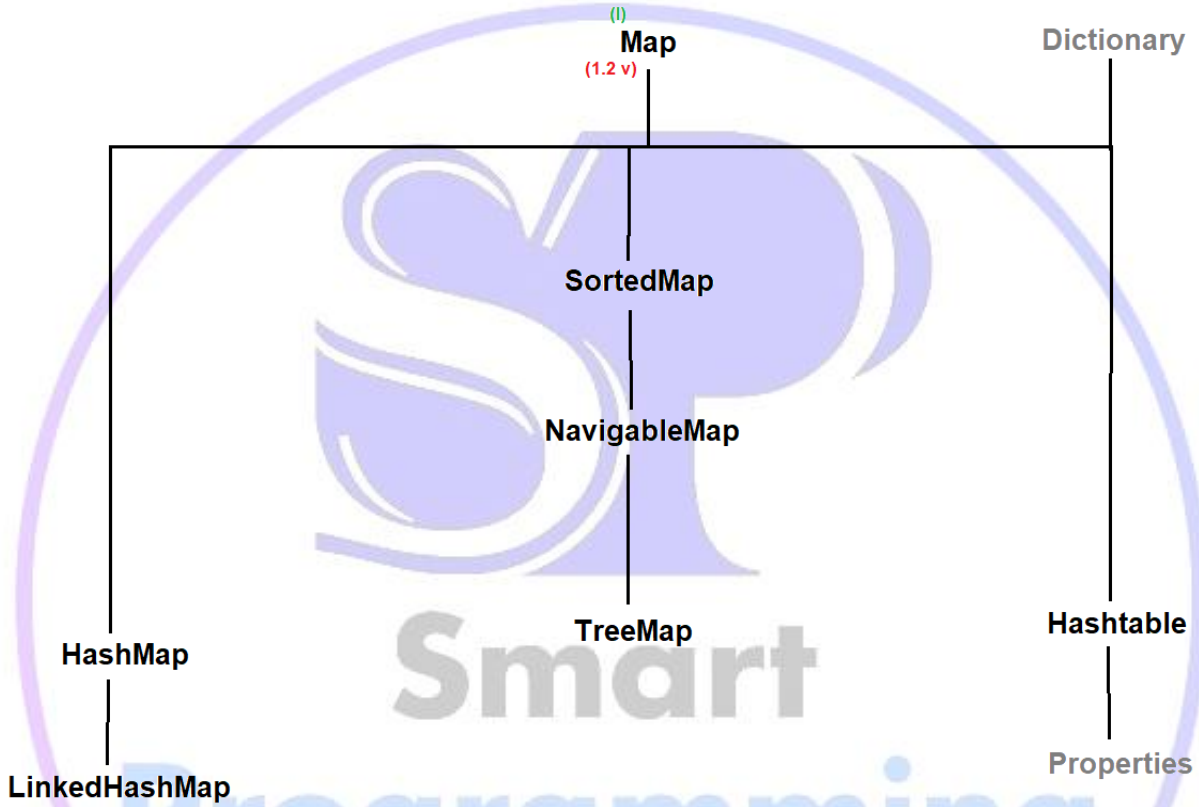
-> It provides a way by which we dont have to implement all the methods of an interface, we just have to declare an abstract class which inherits the interface and in that abstract class we will override the methods of an interface. When we create a class which inherits the abstract class, then we dont need to override all the methods of interface

==========
==========

=> Map :-

- ➔ Map is an interface which is present in java.util package
- ➔ Map does not inherit Collection interface
- ➔ Syntax : public interface Map { - }
- ➔ Map was introduced in JDK 1.2 version

➔ Hierarchy of Map interface :-



➔ Properties of Map :-

1. Map stores the data in key-value pair. Each key-value pair is known as Entry
2. In Map, keys should always unique but values can be duplicate
3. Map can store heterogeneous elements or different type of elements

4. In Map keys we can store maximum one null value but in values we can store any number of null values
5. Map does not follows the insertion order by default
6. Map does not follows the sorting order by default

➔ **Methods of Map interface :**

1. public Object put(Object key, Object value)
2. public void putAll(Map map)
3. public Object get(Object key)
4. public Object remove(Object key)
5. public void clear()
6. public boolean isEmpty()
7. public int size()
8. public boolean containsKey(Object key)
9. public boolean containsValue(Object value)
10. public Set keySet();
11. public Collection values();

=> Entry :-

- ➔ Entry is a one key-value pair in Map
- ➔ Without Entry Map is always empty or we can say Map does not contain any key-value pair
- ➔ Entry is an interface which is present in Map interface (or we can say that Entry is a sub-interface of Map interface)
- ➔ Syntax :

```
interface Map
{
    //methods
    interface Entry
    {
        1. Object getKey()
        2. Object getValue()
        3. Object setValue(Object obj)
    }
}
```

=> HashMap :-

➔ HashMap is a direct implemented class of Map interface which is present in java.util package

➔ Syntax : public class HashMap extends AbstractMap implements Map, Cloneable, Serializable { - }

➔ HashMap was introduced in JDK 1.2 version

➔ The underline data structure of HashMap is "Hashtable"

➔ **Properties of HashMap**

1. HashMap stores the values in key-value pair and each key-value pair is known as Entry
2. In HashMap, keys should always unique but values can be duplicate
3. HashMap can store hetrogeneous elements or different type of elements

4. In HashMap keys we can store maximum one null value but in values we can store any number of null values
5. HashMap does not follows the insertion order by default
6. HashMap does not follows the sorting order by default

7. HashMap is non-synchronized Map because HashMap does not contain any synchronized methods
8. HashMap allows more than one thread at one time
9. HashMap allows the parallel execution
10. HashMap reduces the execution time which in turn makes our application fast
11. HashMap is not threadsafe
12. HashMap does not gurantee for data consistency

-> Working of HashMap :-

1. Whenever we create HashMap, its inital capacity is 16 elements
2. HashMap load factor is 75%

➔ Constructors of HashMap :-

1. public HashMap()
2. public HashMap(int capacity)
3. public HashMap(int capacity, float loadFactor)
4. public HashMap(Map m)

➔ Methods of HashMap :-

= same methods as that of Map interface

➔ When we should use HashMap:-

= HashMap is good for searching or retrival operations

➔ How to get synchronized version of HashMap:-

= By default HashMap is non-synchronized but if we want to get synchronized version of HashMap then we have to use synchronizedMap() method of Collections class

---

=> LinkedHashMap :-

➔ LinkedHashMap is the child class of HashMap which is present in java.util package

➔ Syntax : public class LinkedHashMap extends HashMap implements Map { - }

➔ LinkedHashMap was introduced in JDK 1.4 version

➔ LinkedHashMap underline data structure is "Hashtable + LinkedList"

➔ Properties of LinkedHashMap :-

= LinkedHashMap has the same properties as that of HashMap but one difference is LinkedHashMap follows the insertion order

➔ Constructors of LinkedHashMap :-

= Same constructors as that of HashMap

➔ **Methods of LinkedHashMap :-**

= Same methods as that of HashMap

➔ **When we should use LinkedHashMap :-**

= If we have to create cache based applications then we can use LinkedHashMap

=> What is difference between HashMap & LinkedHashMap :-

1. HashMap was introduced in JDK 1.2 version
   LinkedHashMap was introduced in JDK 1.4 version

2. HashMap does not follows the insertion order
   LinkedHashMap follows the insertion order

3. HashMap uderline data structure is "Hashtable"
   LinkedHashMap underline data structure is "Hashtable + LinkedList"

=> Difference between == & .equals() method :-

== is used for reference comparision or address comparision

equals() method is used for content comparision

=> IdentityHashMap :-

➔ IdentityHashMap is an implemented class of Map interface which is present in java.util package

➔ Syntax : public class IdentityHashMap extends AbstractMap implements Map, Serializable, Cloneable { - }

➔ IdentityHashMap was introduced in JDK 1.4 version

➔ **Properties, Constructors & Methods are same as HashMap**

➔ **Point to remember :-**

1. Map does not contain duplicate keys but can contain duplicate values
2. If we insert key-value pair in HashMap internally .equals method will be called and keys will be compared, if the value is true, it will treat as duplicate key and it will not insert that key

3. If we insert key-value pair in IdentityHashMap, then internally == operator will be called, and keys will be compared, if the value is true, then it will replace the value or if the value is false then it will insert that key-value pair

⇨ **Difference between HashMap & IdentityHashMap :-**

1. HashMap was introduced in JDK 1.2 version
   IdentityHashMap was introduced in JDK 1.4 version

2. In case of HashMap to check for duplicate keys, internally .equals () method is called
   In case of IdentityHashMap, to check for duplicate keys, internally == operator will be called

=> How to delete an object in java ?

➔ There are 2 steps to delete an object in java :-

1. Provide null value to the reference
2. Call gc() method

=> WeakHashMap :-

➔ WeakHashMap is a class which implements the Map interface

➔ Syntax : Public class WeakHashMap extends AbstractMap implements Map { - }

➔ WeakHashMap was introduced in JDK 1.2 version

➔ **Properties, Constructors and methods are same as HashMap**

=> Difference between HashMap & WeakHashMap :-

1. HashMap does not allow the garbage collector to delete its elements

   WeakHashMap allows the garbage collector to delete its elements

=> SortedMap :-

➔ SortedMap is a child interface of Map interface which is present in java.util package

➔ Syntax : public interface SortedMap extends Map { - }

➔ SortedMap was introduced in JDK 1.2 version

➔ Properties of SortedMap :-

1. SortedMap stores the data in key-value pair where key must be unique but values can be duplicate
2. SortedMap does not follows the insertion order w.r.t. keys
3. SortedMap follows the sorting order w.r.t. keys
4. SortedMap can store homogeneous and heterogeneous keys :-
    a. If we are depending on default natural sorting order then the keys should be homogeneous and Comparable otherwise it will provide an exception saying "java.lang.ClassCastException"

b. If we are defigning our own sorting order by Comparator, then the keys can be heterogeneous.

5. We cannot store null values in SortedMap

6. SortedMap is non-synchronized Map

➔ Methods of SortedMap :-

1. Object firstKey();
2. Object lastKey();
3. SortedMap headMap(Object key);
4. SortedMap tailMap(Object key);
5. SortedMap subMap(Object key2, Object key2);
6. Comparator comparator();

=> NavigableMap :-

➔ NavigableMap is the child interface is SortedMap which is present in java.util package

➔ Syntax : public interface NavigableMap extends SortedMap { - }

➔ NavigableMap was introduced in Java SE 6 version

➔ Properties of NavigableMap :-

= NavigableMap is same as SortedMap but it provides some extra navigable methods

➔ Methods of NavigableMap :-

1. public NavigableMap descendingMap()
2. public Object ceilingKey(Object key)
3. public Object higherKey(Object key)
4. public Object floorKey(Object key)
5. public Object lowerKey(Object key)
6. public void pollFirstEntry()
7. public void pollLastEntry()

=> TreeMap :-

➔ TreeMap is a clas which provides the implementation for NavigableMap, SortedMap & Map interface

➔ Syntax : public class TreeMap extends AbstractMap implements NavigableMap, Cloneable, Serializable { - }

➔ TreeMap was introduced in JDK 1.2 version

➔ The underline data structure of TreeMap is "Red-Black tree"

➔ Properties of TreeMap :-

1. TreeMap stores the data in key-value pair which key must be unique but value can be duplicate
2. TreeMap does not follows the insertion order wrt keys
3. TreeMap follows the sorting order wrt keys

4. We can store homogeneous & heterogeneous elements :-
    a. If we are depending on default natural sorting order then the keys must be homogeneous and comparable
    b. If we are defining our own sorting by Comparator then the keys can be heterogeneous
5. We cannot store the null values in TreeMap

6. TreeMap is non-synchronized Map
7. TreeMap allows more than one thread at one time
8. TreeMap allows the parallel execution
9. TreeMap reduces the execution time which makes our application fast
10. TreeMap are not thread-safe
11. TreeMap does not provide guarantee for data consistency

➔ Constructors :-

1. public TreeMap() { - }
2. public TreeMap(Comparator comparator) { - }
3. public TreeMap(Map m) { - }
4. public TreeMap(SortedMap m) { - }

➔ Methods :-

= implements all the methods of Map, SortedMap & NavigableMap interface

➔ When we should use TreeMap :-

= When we want to store a large number of elements in sorting order then we have to use TreeMap which makes the retrival operation fast

➔ "null" value insertion in TreeMap :-

1. Till 1.6 version we were able to insert null value at first key position but if we insert any other value then it will provide "NullPointerException"
2. If we insert null values after first position then it will throw an exception saying "NullPointerException"
3. From 1.7 version we cannot insert the null value even at first position

---

=> What is difference between HashMap, LinkedHashMap & TreeMap :-

1. HashMap was introduced in JDK 1.2 version
   LinkedHashMap was introduced in JDK 1.4 version
   TreeMap was introduced in JDK 1.2 version

2. HashMap underline data-structure is "Hashtable"
   LinkedHashMap underline data-structure is "Hashtable+LinkedList"
   TreeMap underline data-structure is "Red-Black Tree"

3. HashMap does not follows the insertion order
   LinkedHashMap follows the insertion order
   TreeMap does not follows the insertion order

4. HashMap does not follows the sorting order
   LinkedHashMap does not follows the sorting order
   TreeMap follows the sorting order

5. HashMap allows the heterogeneous objects by default
   LinkedHashMap allows heterogeneous objects by default

TreeMap allows homogeneous objects by default

6. HashMap allows the null value insertion
   LinkedHashMap allows the null value insertion
   TreeMap does not allows the null value insertion

=> Hashtable :-

- ➔ Hashtable is a direct implemented class of Map interface which is present in java.util package
- ➔ Syntax : public class Hashtable extends Dictionary implements Map, Cloneable, Serializable { - }
- ➔ Hashtable was introduced in JDK 1.0 version
- ➔ Hashtable is also known as legacy class
- ➔ The underline data structure of Hashtable is "Hashtable"

- ➔ Properties of Hashtable :-

1. Hashtable stores the values in key-value pair and each key-value pair is known as entry
2. In Hashtable, keys should always unique but values can be duplicate
3. Hashtable can store heterogeneous elements at key position
4. In Hashtable we cannot insert null values at key or value position

5. Hashtable does not follows the insertion order by default
6. Hashtable does not follows the sorting order by default

7. Hashtable is synchronized map
8. Hashtable does not allows more than one thread at one time
9. Hashtable allows the sequential execution
10. Hashtable increases the execution time which in turn makes our application slow
11. Hashtable is thread-safe
12. Hashtable provides the guarantee for data consistency

➔ Constructors :-

1. 1. public Hashtable() { - } //it will create a Hashtable having 11 as capacityand default fill ratio is .75%
2. public Hashtable(Map t) { - }
3. public Hashtable(int initialCapacity) { - }

4. public Hashtable(int initialCapacity, float loadFactor) { - }

➔ Methods :-

= same methods as that of Map interface

➔ When we should use Hashtable :-

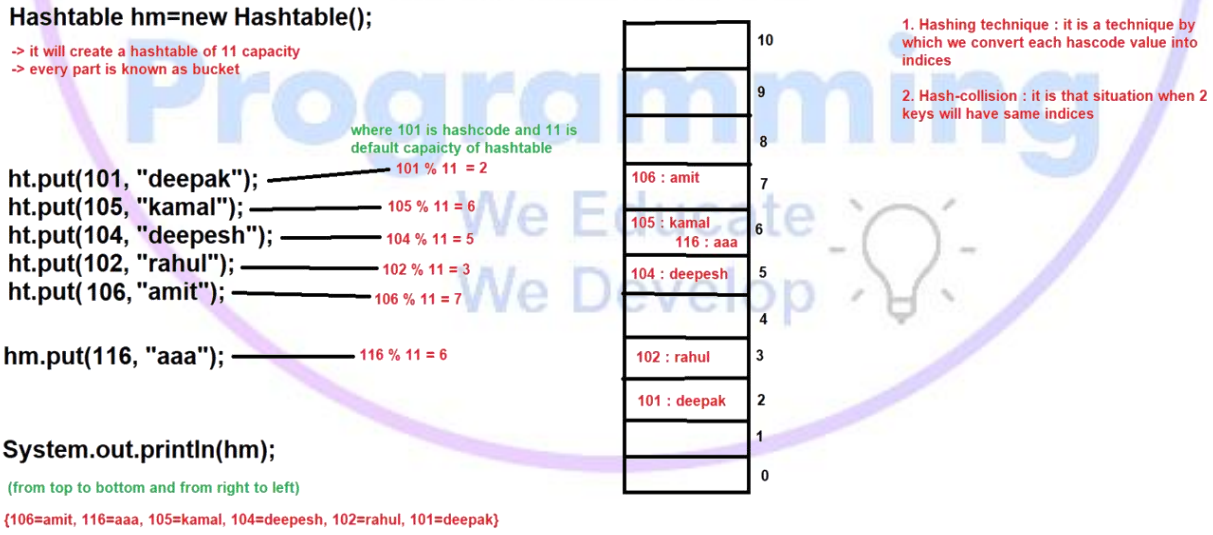
= Hashtable is good for searching or retrieval operation

=> Working of Hashtable :-

➔ "hashcode" is the unique integer value of each and every object that is provided by JVM

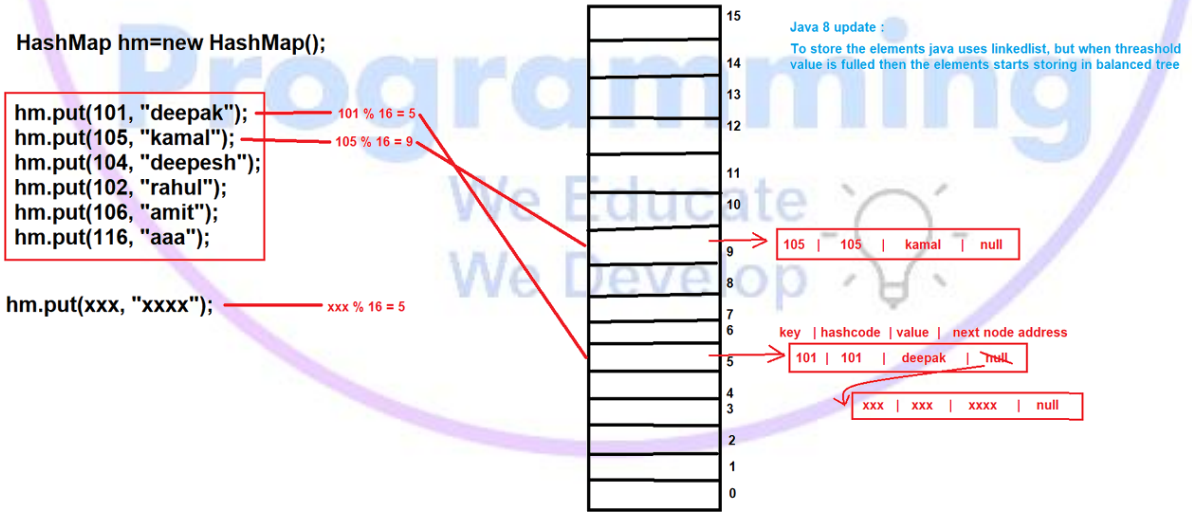
➔ Hashtable initialCapacity is 11

- ➔ Then for each and every key hashcode value will be generated and its index position will be calculated by using hashing technique
- ➔ And at that index position that key-value pair or entry will be inserted
- ➔ If two elements have same index position, then that entry will be inserted at right side of previous entry
- ➔ When the values are traversed then they are traversed from top to bottom and right to left



=> Working of HashMap :-

- ➔ HashMap initialCapacity is 16
- ➔ For every entry it will calculate the index position and store the element there.
- ➔ If multiple entries have same index position, then it will create linked list and starts storing in that linked-list
- ➔ In java 8 updates, after threshold value is filled then it starts storing the elements in balanced tree



=> Difference between HashMap & Hashtable :-

1. HashMap was introduced in 1.2 version
   Hashtable was introduced in 1.0 version

2. HashMap is not a legacy class
   Hashtable is a legacy class

3. In HashMap we can store the null values
   In Hashtable we cannot store the null value at key or value position

4. HashMap is non-synchronized Map because HashMap does not contain any synchronized methods
   Hashtable is synchronized Map because it contains synchronized methods

5. Multiple points related to synchronization

=> Dictionary :-

➔ Dictionary is an abstract class which is present in java.util package
➔ Dictionary represents the key-value pair similar to Map interface
➔ Syntax : public abstract class Dictionary { - }
➔ As Dictionary is an abstract class, thus we cannot create an object of it

➔ Methods :-

1. abstract public int size();
2. abstract public boolean isEmpty();
3. abstract public Enumeration keys();
4. abstract public Enumeration elements();
5. abstract public Object get(Object key);
6. abstract public Object put(Object key, Object value);
7. abstract public Object remove(Object key);

- ➔ Dictionary was introduced in JDK 1.0 version
- ➔ Since java does not support multiple inheritance, if any class extends the Dictionary class, then it will not be able to inherit any other class and due to this reason, Map interface was created.
- ➔ Dictionary class is not used that much as Map interface.

=> Properties :-

- ➔ Properties is the child class of Hashtable which is present in java.util package
- ➔ Syntax : public class Properties extends Hashtable { - }
- ➔ Properties is the file which stores the data in key-value pair
- ➔ Properties can contain "key-value pair in the form of String type only"

➔ When we should use Properties :-

- Whenever there is any data which can change frequently in future then we should not embed that data in our java application. Instead of that we have to create properties file and provide the data in key-value pair so that we dont need to recompile, rebuild, redeploye and even no need to restart the server
- For example validations, database connection, exception handling etc

➔ Constructors :-

1. public Properties() { - }
2. public Properties(Properties defaults) { - }

➔ Methods :-

1. void load(InputStream is)
2. String getProperty(String pname)
3. String setProperty(String pname, pvalue);
4. void store(OutputStream os, String comments)

=> Concurrent Collections :-

➔ Why Concurrent Collections were introduced :-

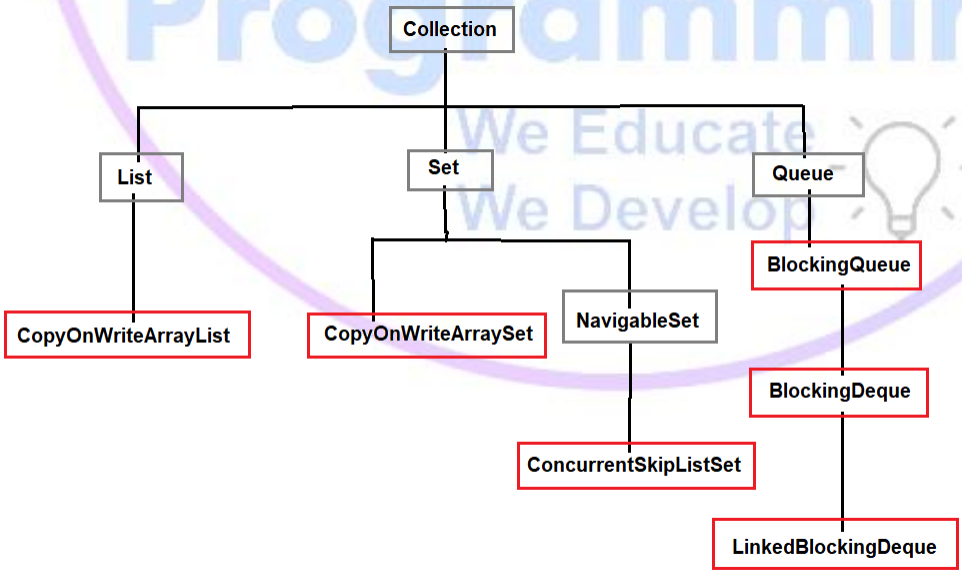
- Most of the classes that we have used till now (traditional collections) are non-synchronized for example ArrayList, LinkedList, HashSet, HashMap etc
- If any class is non-synchronized then it does not guarantee for data consistency which can ruin our application
- Some of the synchronized collection like Vector or Stack have low performace
- In traditional collections, if a thread is iterating a Collection object and if another thread try to add new element in that iterating object, then it will provide an exception saying "ConcurrentModificationException"

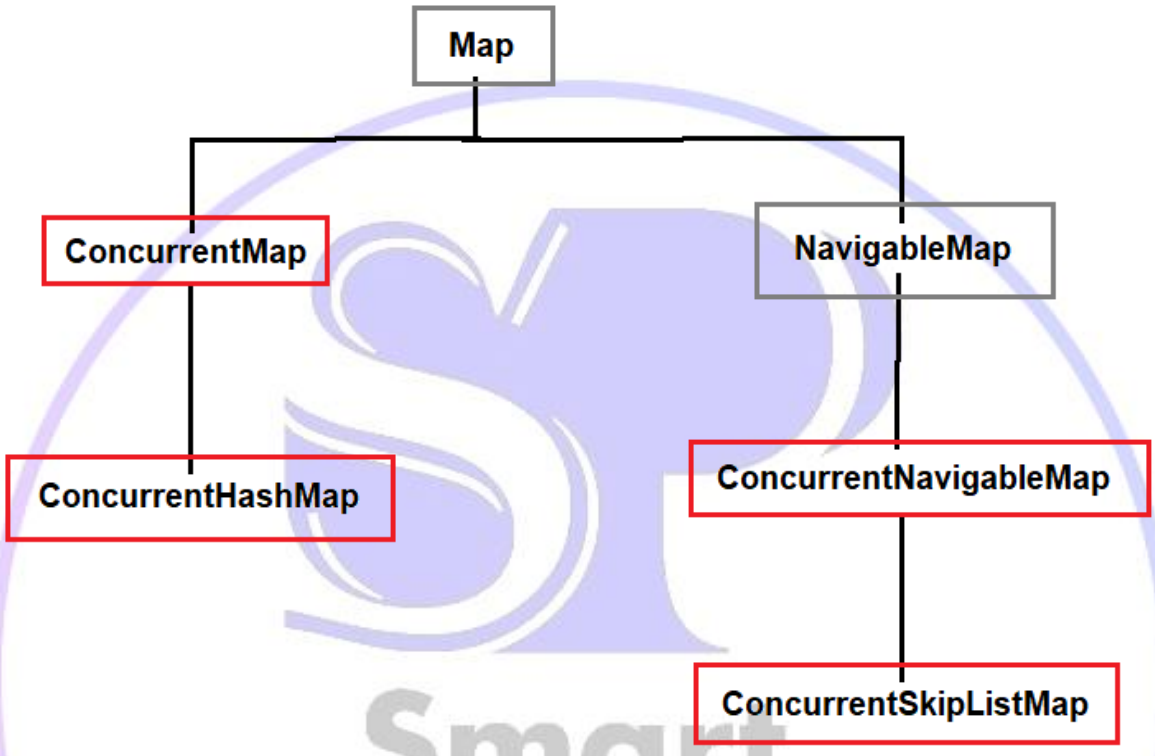
➔ Due to above problems, traditional Collections were not good for multithreaded applications and due to this, concurrent collections were introduced

➔ The concurrent Collections were introduced in JDK 1.5 version

➔ These Concurrent Collection classes are present in java.util.concurrent package

➔ Hierarchy of Concurrent Collection :-





=> Generics :-

➔ Generics means parametrized types which means that we can provide any type of parameter to the classes, interfaces or methods

➔ Generics were introduced in JDK 1.5 version

➔ Generics are represented by angular braces - < >

➔ The main objective of Generics are :-

1. To provide type safety
2. To resolve type casting problem

➔ By default "arrays are type safe"

➔ Now for collections, till JDK 1.4 version, collections were not generic types

➔ In JDK 1.5 version, Generic Collections were introduced

➔ NOTE : We can only provide non-primitive values in generics

=> Generic Classes :-

➔ If any class is declared with type parameters then it is known as Generic Class

➔ Generic classes can be user-defined classes or predefined classes (collections classes)

➔ Generic type can be any valid identifier name

➔ We can provide any number of parameters in generics

=> Generic Bounded Types :-

➔ We can bound the type parameter for a particular range by using extends keyword. And

this concept is known as Generic Bounded Type Concept

➔ Syntax : class A<T extends X>
(X can be any class or interface)

➔ Cases :-

1. We can only use extends keyword, not implements keyword
2. We can only use Non-Primitive data types
3. class A<T extends X & Y>