4/11/24

## Array:-

arraytype [ ] arrayname = new arraytype [size];

- new → operator
- arraytype [size] → array constructor
- arraytype object

Array
- Array of primitive types
  - Store only primitive datatype
- Array of class type
  - Store object of class

Ex:-

```
public class MainClass1 {
 public s.v.m [S[] a) {
   Sopln ("MMS");
 int [] arr1 = new int[5];
  Sopln (arr1);
 double [] arr2 = new double [5];
  Sopln (arr2);
 long[] arr3 = new long[5];
  Sopln (arr3);
 byte[] arr4 = new byte [5];
  Sopln (arr4);
 Sopln ("MME");
 } }
```

o/p

```
[I@
[D@
[L@
[B@
```

Ex:-

```
public class MainClass2 {
  p.s.v.m (S[] a) {
    Sopln ("MMS");
    // Array creation using new operator
    int [] arr1 = new int[5];
    for (int i = 0; i < arr1.length; i++) {
        Sopln (arr1[i]);
    }
    Sopln ("----------");
```

```
//Array Creationusing array initialiser.
int[] arr2= {10,20,30,40,50};
for(int i=0; i<arr2.length;i++) {
        sopln(arr2[i]);
    }
sopln("----------");
//Array creation using both new operator &
  array initialiser.
int[] arr3 = new int[] {12,22,44,55,56};
        // this is known as anonymous inner class.
for(int i=0; i<arr3.length;i++)
  {
   sopln(arr3[i]);
  }
Sopln("MME");
  }
}
```

Ex-3

```
→ public class Book {
    String title;
    int pages;
    String author;
    int edition;
public Book(String title, int pages, String author, int edition){
    this.title = title;
    this.pages = pages;
    this.author = author;
    this.edition = edition;
  }
@Override
  public String toString(){
  return "Book [title = "+title+", Pages = "+Pages+", author="
                +author+", edition = "+edition];
  } }
```

```
public class MainClass3 {
    p.s.v.m(sc[] a) {
    Sopln ("MMS");
Book b1 = new Book("Java Coding", 200, "Ramesh", 2);
Book b2 = new Book ("JS coding", 200, "Suresh", 2);
Book b3 = new Book ("SQL", 150, "Swetha/Sharath", 1);
Book b4 = new Book ("DSA", 200, "Darshan", 2);
Book [] bookarray = new Book [4]; //created array of book.
//Book type array
//Storing book objects in array
bookarray[0] = b1;
book array[1] = b2;
book array [2] = b3;
book array [3] = b4;
//Printing array elements
for (int i=0; i< bookarray.length; i++) {
    Sopln (bookarray[i]);
    sopln ("MME");
}
}
}
```

(or)

```
Book[] bookarray = new Book [ ] {b1, b2, b3, b4};
//printing book details.
Sopln ("----------");
Sopln ("Title \t Author \t Edition ");
Sopln ("---------");
for (int i=0; i< bookarray.length; i++) {
    Sopln (bookarray[i].title "\t" + bookarray[i].author +
        "\t" + book array[i].editor)
    }
}
}
```

5/11/24

~~Collections~~ Collections :-

```
public class BookStore {
    //Private Book[] bookArray = new Book[5];
    //Book type Array
    private Book[] bookArray;
    private int index;
//no arg constructor
/**
 * this constructor creates Bookstore with default capacity of:
 */
BookStore() {
    bookArray = new Book[5];
    }
//int parameter constructor
/**
* this constructor creates Bookstore with initial capacity
* specified in parameter of constructor
* @param capacity
*/
BookStore (int Capacity) {
  bookArray = new Book [int capacity];
  }
/**
* this method add the given book into the array based
* on the index
* @param book
*/
public void addBook (Book book) {
    if (index < bookArray.length) {
        bookArray [indexx++] = book;
        Sop/n ("added book in the store"); }
```

```
else
 {
  sopln ("Book Store filled with the Books");
 }
}
}.

public Class MainClass4 {
  public static void main (String [ ] args ) {
    Sopln ("main method Started");
    Book b1 = new Book ("Java Coding ", 200, "Ramesh", 2);
    Book b2 = new Book ("JS Coding", 200, "Suresh", 2);
    Book b3 = new Book ("SQL", 150, "SWETHA", 1);
    Book b4 = new Book ("DSA", 200, "Darshan", 2);
    Book b5 = new Book ("Web", 125, "Harish", 1);
    Book b6 = new Book ("SQL", 150, "Sharath ", 1);
    BookStore bs = new Bookstore ();
      bs. addBook (b1);
      bs. addBook (b2);
      bs. addBook (b3);
      bs. addBook (b4);
      bs. addBook (b5);
      bs. addBook (b6);
    Sopln ("MME");
  }
}
```

→ import java.util.Arraylist;

```
public class MainClass1 {
    P.S.V.M (S[] a) {
        Sopln ("MMS");
        ArrayList list1 = new Arraylist();
//Creating empty List, with default capacity of 10.
        ArrayList list2 = new Arraylist(20);
//creating empty list, with initial specified capacity of 20.
        Sopln (List 1);
        Sopln (List 2);
        Sopln ("MME");
    }
}
```

→ import java.util.ArrayList;

```
public class MainClass2 {
    P.S.V.M(S[] a) {
        Sopln ("MMS");
ArrayList numlist = new ArrayList();
    numlist.add(100);
    numlist.add(300);
    numlist.add(500);
    numlist.add(400);
    numlist.add(200);
int n = numlist.size();
    Sopln("List size is" +n);
    Sopln (numlist);
    Sopln ("removing an element");
~~Arraylist~~ numlist.remove(new ~~Integer~~(500));
        n = numlist.size();
```

```
Sopln("List size is"+n);
Sopln (numlist);
Sopln ("MME");
}
}
```

06/11/24

```
→ import java.util.ArrayList;
public class MainClass3 {
P.S.V.M (S[]a) {
Sopln ("MMs");
ArrayList numlist = new ArrayList();
//adding integer values
numlist.add(100);
numlist.add (300);
numlist.add (500);
numlist.add (400);
numlist.add (200);
//accessing element @ index 2.
Sopln (numlist.get(2));
//accessing all the elements & printing it based on index
Sopln ("List element using index");
for (int index=0 ; index< numlist.size(); index++) {
Sopln (numlist.get(index));
}
Sopln ("MME");
}
}
```

access

get() → to get the value

set() → to replace the ~~value~~ element with new value

remove() → to delete the element

clear() → Used to delete all the elements
ie., list becomes empty.

IsEmpty() → Used to check the list is empty (or) not.
→ It always returns in Boolean.

→ import java.util.ArrayList;

```
public class MainClass5 {
    P.S.V.M (S[] a) {
      Sopln ("MMS");
    ArrayList numlist = new ArrayList();
    //adding integer values.
     numlist.add (100);
     numlist.add (300);
     numlist.add (500);
     numlist.add (400);
     numlist.add (200);
     Sopln(" Is List empty ?"+numlist.isEmpty());
     Sopln ("List elements are");
     Sopln (numlist);
     Sopln ("replacing element");
     numlist.set(2,700); //remove the element at
      //given index & replace by new element.
     Sopln ("List elements are");
     Sopln(numlist);
     Sopln ("clearing all the elements");
     numlist.clear(); //delete all the elements, list becomes empty
     Sopln ("list elements are");
     Sopln (numlist);
     Sopln ("Is List empty ?"+numlist.isEmpty());
     Sopln ("MME");
    }
}
```

→ import java.util.Arraylist;

```
public class MainClass6 {
    P.S.V.M (String [] args) {
  Sopln ("..........");
  ArrayList strList = new Arraylist();
  StrList.add ("test");
  Strlist.add ("demo");
  Strlist.add (" Sample");
  Strlist.add ("run");
  Strlist.add ("execute");
  Sopln (" list elements: ");
  Sopln (StrList);
  String element = (String) strlist.get(2);
  Sopln (element.toUpperCase());
  Sopln ("----------");
  }
}
```

→ import java.util.Arraylist;

```
public class MainClass7{
    P.S.V.M (S[] a) {
    Sopln ("........");
  Arraylist Mixlist = new Arraylist();
    mixlist.add("test");
    mixlist.add (100);
    mixlist.add (5.6);
    mixlist.add ("demo");
    mixlist.add (true);
    mixlist.add ('t');
  Sopln ("List all elements");
```

```
for (int i=0; i<mixlist.size(); i++) {
    if (mixlist.get(i) instance of String) {
        Sopln (mixlist.get(i));
    }
Sopln ("----------")
}
}
```

```
import java.util.Arraylist;
public class MainClass & {
    p.s.v.m (s[] a) {
        Sopln.
```

→ ArrayList<String> strlist = new ArrayList<String>();
//generics.

→ Arraylist<String> strlist = new Array list<String>();
//generics.

```
strlist.add ("test");
strlist.add ("demo");
strlist.add ("sample");
strlist.add ("run");
strlist.add ("execute");
String[] strarr = (String[]) strlist.toArray();
int[] intarr = {10,20,30,40,50};
List numlist = Arrays.asList(int arr);
```

→ WAP to add some heterogeneous elements into arraylist and count how many integer objects are present in arraylist.

```
-> import java.util.ArrayList;
   public class MainClass & {
       p.s.v.m(s[] a) {
       Sopln("MMS");
   ~~Array list~~   int count=0;
       Arraylist mixlist = new Arraylist();
           mixlist.add("Rama");
           mixlist.add(100);
           mixlist.add("Deepthi");
           mixlist.add(400);
           mixlist.add(5.6);
           mixlist.add(300);
           mixlist.add('c');
       Sopln(
       for(int i=0; i<mixlist.size();i++){
           if (mixlist.get(i) instance of Integer) {
               count = count++;
           }
       }
       sopln(count);
       }
   }
```

→ WAPT to create emp class and take properties called ID, name, salary & create some objects for employee → Add emp objects to arraylist & print the details whose empname starts with a.

```
import java.util.ArrayList;
→ public class Employee
  {
    int id;
    String name;
    ~~String s~~ double sal;

  Employee (int id, String name, double sal)
    {
      this.id = id;
      this.name = name;
      this.sal = sal;
    }

  @Override
    public String toString() {
    return "Employee (name = "+name+" id = "+id+" Sal:
                    sal);
  } }

public class MainClass {
        P.S.V.M (S[] a) {

    Employee e1 = new Employee (1042, "RAMESH", 55
    Employee e2 = new Employee (1043, "ABHIJEET", 45
    Employee e3 = new Employee (1044, "Damini", 60000
    Employee e4 = new Employee (1045, "RIYA", 50000

    ArrayList<Employee> emparray = new ArrayList<Emply
                                                  ();
      emparray.add(e1);
      emparray.add(e2);
      emparray.add(e3);
      emparray.add(e4);
```

```
for(int i=0; i< emparray. size(); i++)
  {
    int reg = "[A,a].*";
    Sopln (Pattern. matches (reg, emparray.get(i)));
    }
  }
}
```

(or)

```
if Employee e= al.get(i);
  if (e.name
```

11/11/24

```
⇒ public class MainClass {
    P.S.V.M (S[].a) {
     Sopln ("MMS");
   ArrayList <String> Strlist = new ArrayList <String>();
    Strlist . add ("Ramesh");
    Strlist . add (" Suresh");
            " ("Umesh");
            " ("Mahesh");
            " ("Mukesh");
            " ("Rajesh");
            " ("Ramalesh");
            " ("Jaggesh");
    Sopln (" Total names "+ Strlist . size());
    Sopln ("Given List element");
    Sopln (Strlist);
   List<String> partialist = strlist. Sublist (2, 5);
```

```
Sopln("partial list element");
sopln ("partialist");
Sopln ("Original List elements");
Sopln (Strlist);
Sopln ("Inserting new element @ Index 4");
Strlist.add(4, "Ramesh"); //insert- Shift the element from right
Sopln ("Original list elements");
sopln (Strlist);
Sopln ("MJME");
}
}
```

1. add() - used to add elements into list.
2. size() - returns total no. of elements in the list.
3. remove(element) - remove the element from the list.
4. get(index) - access the element present in the given index.
5. set(index, element) - replace
6. clear() - delete all element.
7. isEmpty() - check List is empty.
8. add(index, element). insert element @ given index.
9. Sublist(startindex, endindex) - partial list from startindex upto endindex, excluding element @ endindex.
10. remove(index) - remove element @ given index.

12/11/24

## ⇒ Collection:

What is collection?:-

→ Representing group of object as a single entity is known as collection.

→ We can perform operations like adding the element, removing the element, updating the element, aswell as accessing the element.

→ The collection allows to store objects maintain and manage it.

## ⇒ What is collection framework library

→ The collection framework library is a set of interfaces & implementation classes which are developed by using standard data structures like arrays, queue, linked list, Hashtable etc.,

→ The complete Library is present in "java.util Package".

→ The library is built on abstraction principle where we can access the functionalities without knowing the implementation details.

Important interfaces in CFL

1. Iterable
2. Collection
3. List
4. Queue
5. Set
6. Sorted set
7. Navigable Set
8. Dequeue
9. Map
10. Sorted Map
11. Navigatable Map
12. Iterator
13. List Iterator.

Implementation classes in CFL:-

1. ArrayList
2. LinkedList
3. Vector
4. Priority Queue
5. HashSet
6. LinkedHashSet
7. TreeSet
8. HashMap
9. LinkedHashMap
10. Hashtable
11. TreeMap.

Helper class

1. Collections

Helper Methods

1. public Iterator iterator()
2. public ListIterator listIterator()

## Methods of Iterable interface

1. public Iterator iterator()

## Methods of Collection Interface:

1. public boolean add(Object element)
2. public int size()
3. public boolean isEmpty()
4. public boolean remove(Object element)
5. public void clear()
6. public boolean contains (Object element)
7. public Object[] toArray()
8. public boolean addAll(Collection c)
9. public boolean removeAll (Collection c)
10. public boolean retainAll (Collection c)
11. public boolean containsAll (Collection c).

## Methods of List Interface:-

1. public boolean add(int index, Object element)
2. public Object get (int index)
3. public Object remove (int index)
4. public Object set (int index, Object element).
5. public int indexOf (Object element)
6. public int lastIndexOf (Object element)
7. public List subList (int startIndex, int endIndex)
8. public ListIterator listIterator()

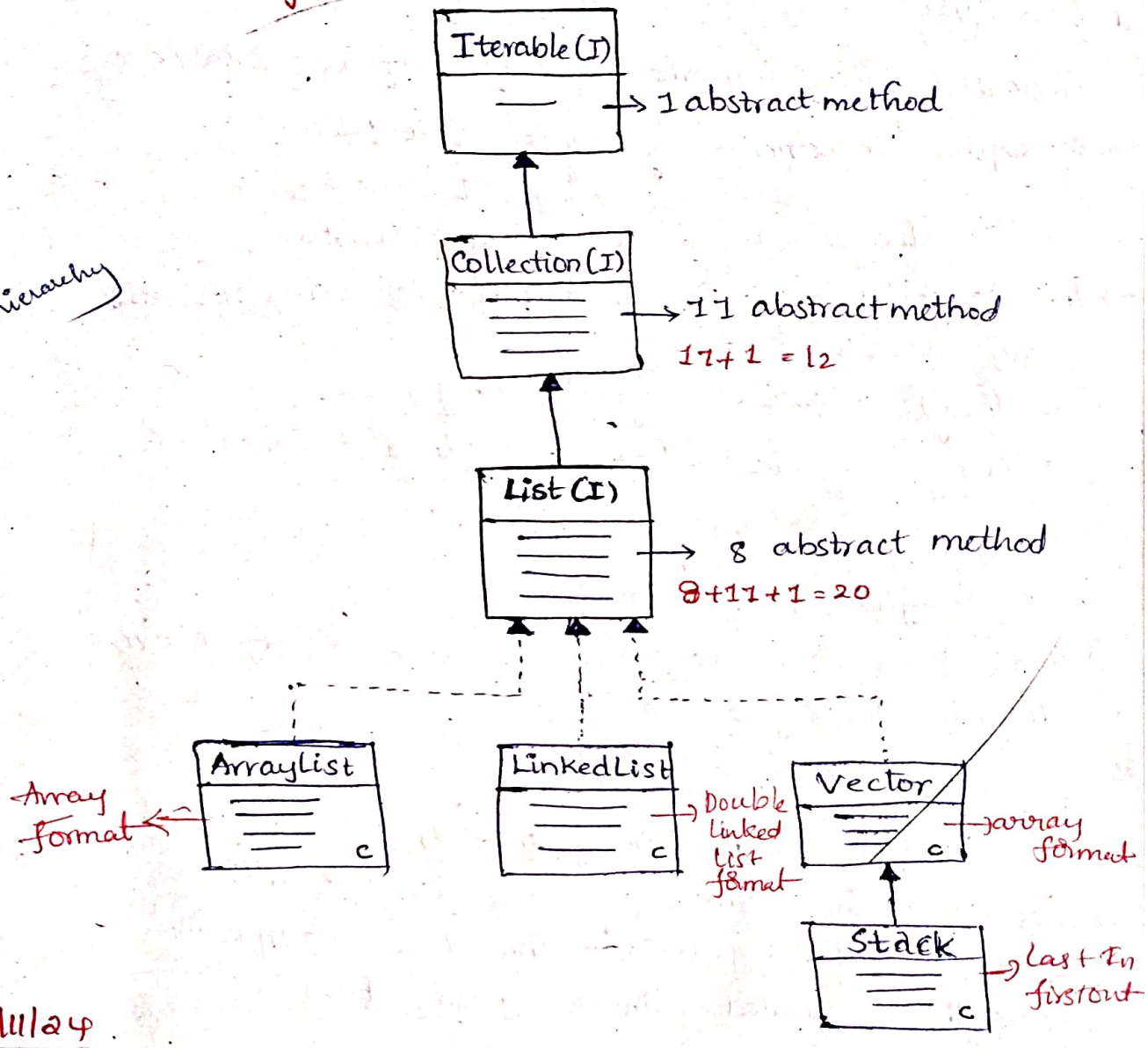
List specific methods

→ Using collection framework.

(i) We can store the data
(ii) we can update the data
(iii) we can delete the data
(iv) we can search the data.
(v) we can arrange the data in pre-defined order.

→ The complete collection framework library is available in java.util package

* List class diagram:-




13/11/24

Q) What are the types of Collection.

→ There are 3 types of Collections

1. List 2. Queue 3. Set.

List:-

→ List is a type of collection.

→ List stores the element sequentially.

→ The elements of the list are stored with index. Hence list is known as indexed Collection.

→ The elements of the list can be accessed based on the index.

→ The List maintains order of insertion.

→ List allows duplicate element.

→ List allows null as a element.

i, Arraylist:-

→ Arraylist is a implementation class of list interface.

→ Arraylist implements 3 marker interface.
↓
an empty interface.

1. Serialisable 2. Clonable 3. Random Access

→ The Arraylist class is implemented by using resizeable array datastructure.

→ The Arraylist default capacity is 10.

→ The Array List class has a overloaded constructor

1. Constructor without parameter:

ArrayList L1 = new ArrayList();

→ This constructor creates empty arraylist object with default capacity of 10.

(2) Int parameter constructor:-

ArrayList L1 = new ArrayList(50);

→ This constructor creates an empty arraylist with specified initial capacity, in the above example the arraylist is created with a capacity of 50.

(3) Collection type parameter constructor:

→ ArrayList L1 = new ArrayList(c1); for interconnection with another collection.
c1 ---> is a collection

→ This constructor creates an arraylist object with the elements of the specified collection.

v.v.imp
The Growing Mechanism of ArrayList:-

→ The Arraylist grows with by increasing the current capacity by 60%.

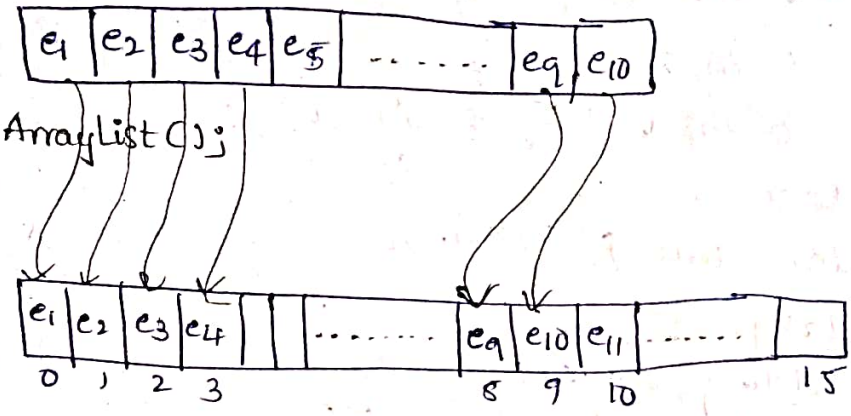
→ Whenever the array list starts resizing it internally
(i) it internally creates new Arraylist with a calculated capacity.
(ii) it copies the element from the old Arraylist into the newly created Arraylist index wise.

→ After copying the elements the reference is pointed to the newly created ArrayList.

→ Add that new element to the new Arraylist

Note:- The old Arraylist obj will be garbage collected.

v.v imp=




```
ArrayList l1 = new ArrayList();
l1.add(e1);
l1.add(e2);
l1.add(e3);
l1.add(e4);
l1.add(e5);
 :
l1.add(e10);
l1.add(e11);
```

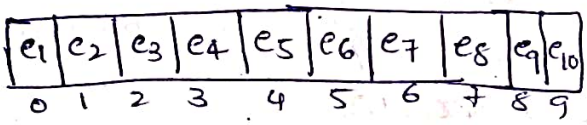
new Cap = (Current capacity × 3/2) + 1

= (10 * 3/2) + 1

⇒ 15 + 1 = 16

→ In ArrayList accessing any element randomly takes same time.

→ Hence, ArrayList is preferable wherever we have to access the data.




l1.get(0); → $t_1$ sec

l1.get(4); → $t_2$ sec

l1.get(9); → $t_3$ sec

$\boxed{t_1 \simeq t_2 \simeq t_3}$

→ The ArrayList is most preferable whenever we have more retrieval operations. Whereas ArrayList is not preferable whenever insertion, deletion operations are more.

## (ii) Vector :

→ Vector is an implementation class of List interface.

→ The functionality wise, Data Structure wise, implementation-wise vector is similar to Array list with the following minute difference.

1. The vector resizes by doubling the capacity.
2. The vector is thread safe.

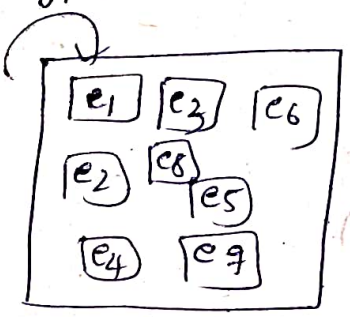
12/11/24 notes

## Fundamentals of Collection

1. add operation :-

adding an object of a class the object is upcasted to ~~upcasted~~ Object class type.

eg:- 1. add Student Object
2. add Employee Object




Box / container
- no size limit

2. Retrieve an element

→ Retrieved element shows object class properties downcast it to access specific properties.

3. Removing an element.

→ Removed element shows object class properties down cast it to access specific properties.

## Iterable Interface :-

→ root interface in collection framework library.

→ has only one abstract method.

→ public Iterator iterator() :-

↳ returns an Iterator type object to iterate / traverse elements "Iterable Collection". irrespective of order storage of object (Data ~~Storag~~ Structure).

→ What is iterable?

→ any entity which allows traverse from 1st element to last element is known as iterable.

→ Collection Interface?

→ The collection interface is a root of all type of interface

→ This interface defines the set of standard method to operate on any type of collection.

→ This interface is defined in java.lang package.

→ This interface is a child interface of iterable interface.

Collection interface methods.

1. public boolean add (Object element)

→ This method is used to add the element into the collection.

→ If the element added successfully the method returns true (otherwise the method returns false.

2. public int size():

→ This method returns the total no. of elements present in the collection.

3. public boolean remove(Object element)

→ This method is used to remove given element from the collection.

→ If the element is found in the collection it remove the element and returns true.

→ If the element is not found it returns false.

4. public void clear()

→ This method removes all the element from the collection and make the collection as empty.

5. public boolean isEmpty()

→ This method returns true if the collection is empty otherwise it returns false.

6. public boolean contains (Object element)

→ The method returns true if the element is found in the collection otherwise it returns false.

7. public Object [] toArray ()

→ This method returns the array of object class type containing the elements of the collection. In otherwords this method converts the collection Array type.

Bulk Operation Method

8. public boolean addALL (Collection c)

9. public boolean removeALL (Collection c)

10. public boolean retainALL (Collection c)

11. public boolean containsALL (Collection c)

// to get Iterator on Collection

12. public Iterator iterator ()

→ inherited from Iterable interface

List Interface :-

→ The List interface defines all the specific operations that can be performed on any type of list.

→ The List interface is a child interface of collection interface.

Imp

List Interface Methods :-

1. public boolean add (int index, Object element)

→ This method is used to insert the given element at the specified index.

→ If the element is inserted successfully it returns true otherwise false.

2. public Object get (int index).

→ This method returns the reference to the element present at given index.

3. public Object remove (int index)

→ This method removes the element present at given index, it returns the removed element.

4. public Object set (int index, Object element).

→ This method replaces the element present at the given index by new element.

5. public int indexOf (Object element)

→ The method returns the index of first occurence of given element.

6. public int LastIndexOf (Object element)

→ The method returns the index of ~~first~~ Last occurance of given element.

7. public List Sublist (int StartIndex, int End Index);-

The method returns the partialist b/w the start index and end index the element at the end index is excluded.

8. public List Iterator list Iterator()

The method returns List iterated type object through which the List element can be traversed

14/11/24

## (ii) Linked List

→ LinkedList is a implementation class of list interface.

→ LinkedList implements another interface Dequeue.

→ LinkedList implements 2 marker interfaces

1. Serialisable 2. Clonable.

→ The linkedlist is implemented by using doublee linked list data structures.

→ Each element added into the LinkedList will be stored as node format.

→ The node will have the address of next element and address of previous element.

→ One element is linked to next element as well as previous element.

→ Each element will have index. the LinkedList doesn't have default capacity.

→ It throws one element at a time.

→ Constructors of Linked List:-

→ The LinkedList has a overloaded constructor.

⇒ Constructor without parameter:-

```
LinkedList L1 = new LinkedList().
```

→ This constructor creates empty linkedlist.

⇒ Collection type parameter constructor:-

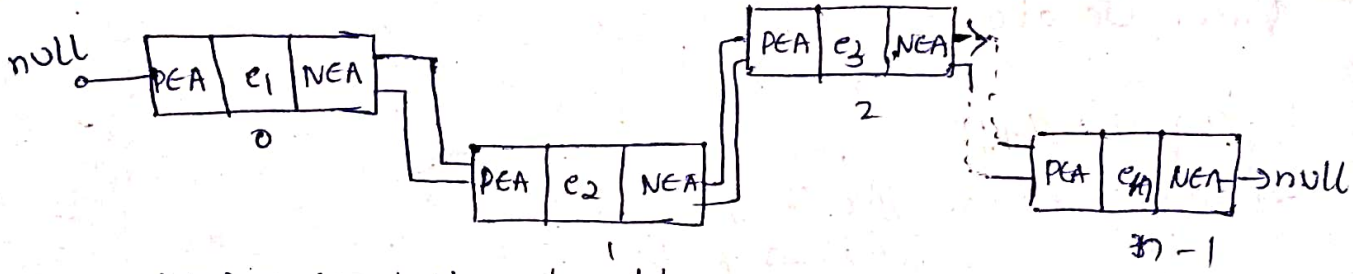
```
LinkedList L1 = new LinkedList(C1)
```

→ C1 → is a collection.

→ This constructor creates a linkedlist with the element of given collection.

```
Linked l1 = new LinkedList();
                ↳ empty

    l1.add(e1);
    l1.add(e2);
    l1.add(e3);
      ⋮
    l1.add(en);
```




NEA – Next Element Addr

PEA – Previous Element Addr

→ The LinkedList is not preferable whenever we are performing the retrieval operations. becoz the random access tym is not same for all the elements.

→ The LinkedList is more preferable whenever we have to perform delete & insertion operation.

| Retrival $t_3 > t_2 > t_1$ | Deletion | insertion; |
|---|---|---|
| l1.get(0) → t1sec | l1.remove(1); | l1.add |
| l.get (4) → t2sec | | |
| l.get (n) → t3sec | | |

## Queue:-

→ Arranging and processing the data/element in FIFO.

→ queue will have 2 ends

1. Head / Front
2. Tail / Rear

## Queue Operation:-

1. enqueue / offer:
   → adding element @ tail of the queue
2. Dequeue:- (poll)
   → removing element @ head of the queue.
3. Examine / Peek. Viewing the head element of the queue.

## Double Ended Queue:- (DeQueue):-

→ arranging & processing the data/element in FIFO.

→ Same

## Queue Operation:

1. enqueue/Offer.
   → adding element @ tail of the queue (OR) @ head of the queue.
2. Dequeue (pop).
   → removing element @ head of the queue (OR) @ tail of the queue.
3. Examine/Peek:
   → Viewing ~~element~~ the head element of the queue (or) tail element.

15/11/24

## Queue:-

→ Queue is a type of function where elements are processed ~~bef~~ with an order.

→ Queue is not a indexed collection hence the elements cannot be accessed based on the index.

→ Queue allows duplicate elements.

→ Queue doesn't support null as an element.

→ Generally a queue has two operations.

→ Adding an element into the queue is known as enqueue operation.

→ The enqueue operation always happen at tail end of the queue.

→ removing the element from the queue is known as Dequeue operation.

→ The Dequeue operation happens at head of the queue.

## Queue interface:-

→ A queue interface is a child interface of collection interface.

→ A queue inteface defines a standard operations that can be performed on any kind of queue.

→ The methods of the queue are)

→ The methods of queue are:

1. public boolean add (Object element):-

→ This method add the given element to the tail end of the queue.

→ If the element is added successfully returns true else false.

2. public boolean offer (Object element):-

→ This method add the given element to the tail end of the queue.

→ If the element is added successfully returns true else false.

3. public Object remove():-

→ This method removes the head element of the queue.

→ If the queue is empty this method throws NoSuchElement -Exception.

4. public Object poll():-

→ This method removes the head element of the queue.

→ If the queue is empty this method returns null.

5. public Object element():-

→ This method retrieve the head element of the queue.

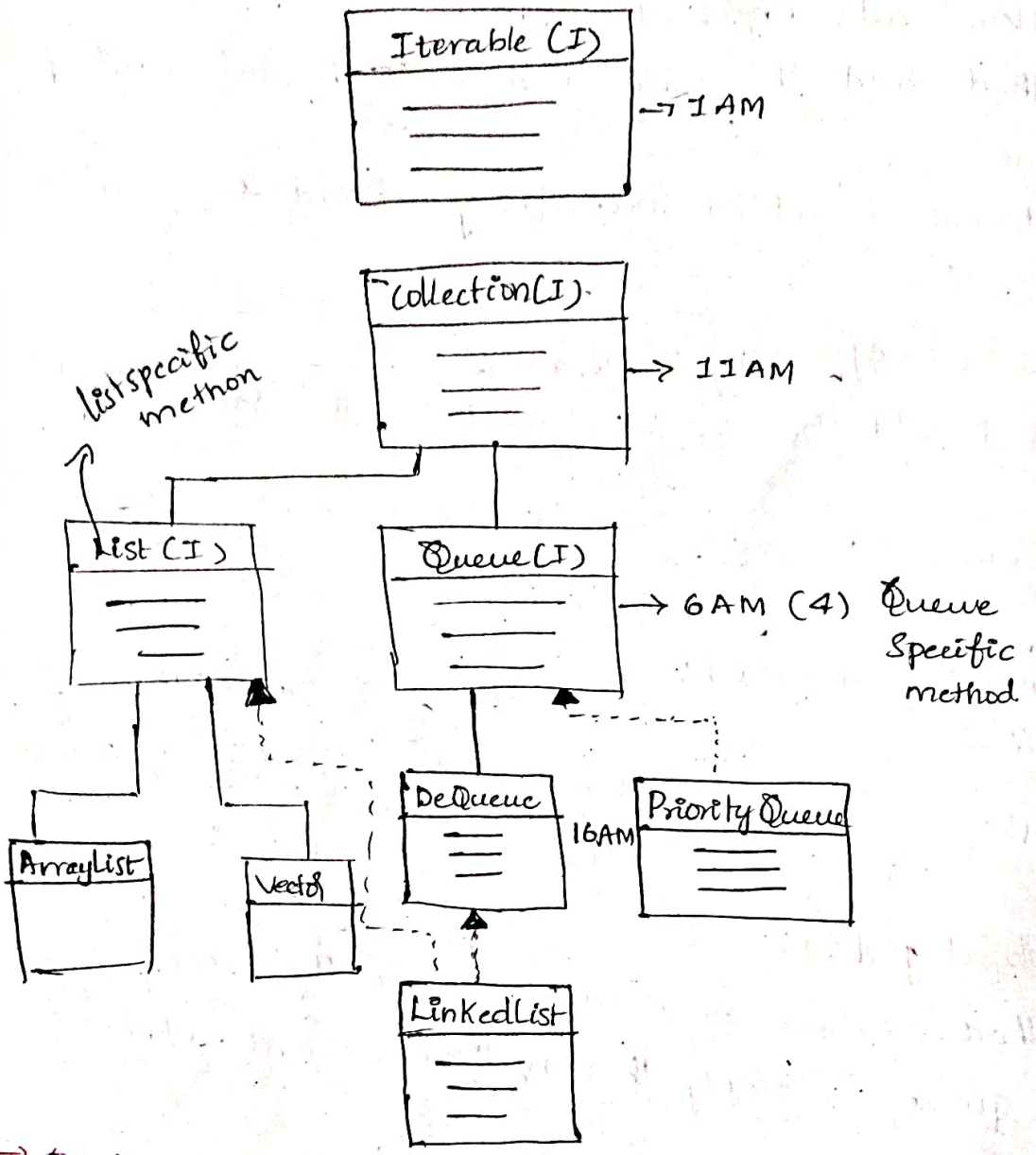
→ If queue is empty the method throws NoSuchElementException.

6. public Object peek():-

→ This method used to retrieve the head element of the queue.

→ If queue is empty the method returns null.

* Class Diagram of Queue




⇒ package learning queue;
import java.util.PriorityQueue;
public class MainClass1 {
P.S.V.M (S[]a) {
Sopln ("MMS");
PriorityQueue pq = new PriorityQueue();
boolean res = pq.isEmpty();
Sopln(res);
int n = pq.size();
Sopln (n);
pq.add (100);
pq.add (23);
pq.add (12);
pq.add (50);

```
pq.add(34);
pq.add(12);
res = pq.isEmpty();
Sopln(res);
n = pq.size();
Sopln(n);
Sopln("QueueElements:");
Sopln(pq);
Sopln("removing head element");
pq.poll();
Sopln("Queue elements:");
Sopln(pq);
Sopln("MAIN");
}
}
}
```

while

```
while(!pq.isEmpty()){
Object element = pq.poll();
Sopln(element);
Object element = pq.poll();
while(element != null)
Sopln(element);
element = pq.poll();
}
```

18/10/24

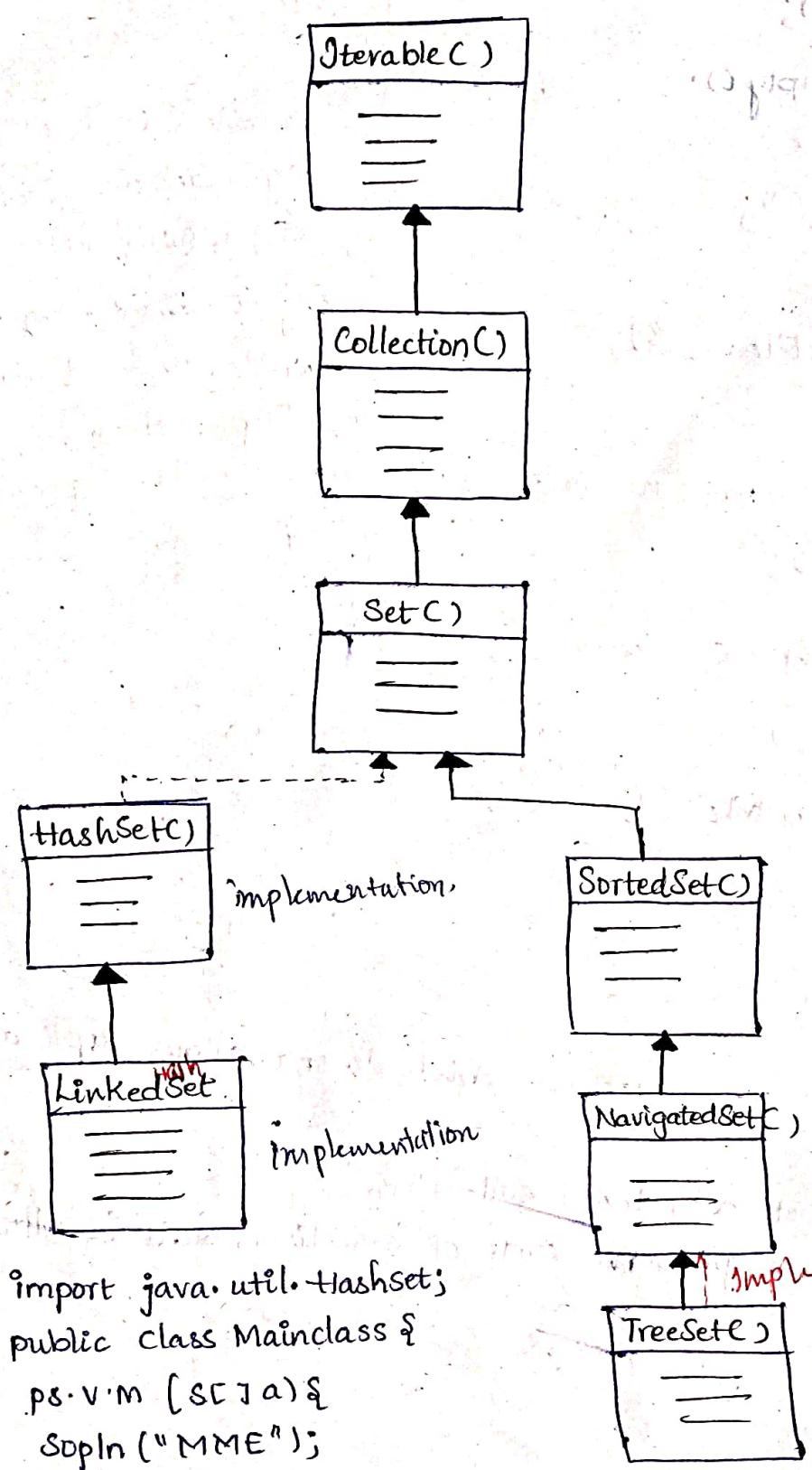
**Set:-**

→ Set is a type of collection which doesn't allow duplicate elements.

→ The set is not a indexed collection.

→ The set doesn't preserve order of insertion. Null is allowed

## Class diagram of set:-




Ex:-
```
import java.util.Hashset;
public class Mainclass {
  ps.v.m (S[] a) {
   Sopln ("MME");
Hashset set1 = new Hashset();
  boolean res = set1.isEmpty();
   int n = set1.size();
   sopln ("Is Set Empty "+res);
   sopln ("set size: "+n);
   sopln ("set elements: "+set1);
   sopln ("adding elements");
   set1.add(100);
```

```
Set1.add(2.5);
Set1. add ("JSPIDERS");
Set1. add ('t');
Set1. add (true);
Set 1. add (null);
res = Set1. isEmpty();
  n = Set1.size();
Sopln ("is SetEmpty ? "+res);
Sopln ("Set Size" +n);
Sopln ("Set element : "+set1);
Sopln ("Adding duplicate elements");
    Set1.add (100);
     n. set1.size ();
    Sopln ("Set size:" +n);
    Sopln (" Set elements : "+ Set1);
    } }
```

⇒

```
Package LearningSet;
import java. util. hashSet;
public class MainClass2 {
   P.S.V. M (S[] a) {
hashSet  Set1 = new hashSet();
boolean res = Set1. isEmpty();
int n = Set1. size ();
Sopln ("If set empty "+res);
Sopln ("Set Size :"+n);
Sopln ("Set elements: "+Set1);
Sopln ("adding elements");
  Set 1. add (100);
  Set1. add (2.5);
  Set1. add ("JSPIDERS");
  Set1. add ('t');
  Set1. add (true);
  Set 1. add (null);
  Sopln ("elements of Set");
  for (Object element: Set 1){
      Sopln (element);
   }
 }
}
```

main

```
→ hashset <String> Set = new HashSet<String>( );
    Set.add ("test");
    Set.add ("demo");
    Set.add ("run");
    Set.add ("Sample");
    Sopln ("Set elements");
    for (String elements : Set) {
      Sopln (elements);
    }
```

- Main

```
→ TreeSet < Integer> Set1 = new TreeSet <Integer>( );
    Set1.add (100);
    Set1.add (23);
    Set1.add (12);
    Set1.add (34);
    Set1.add (9);
    Set1.add (65);
    Set1.add (12);
    Sopln ("Set Size :" + Set1.size());
    Sopln ("Set elements");
    Sopln (Set1);
    Sopln ("Using enhanced for loop");
    for (Integer element : Set1) {
      Sopln (elements);
    }
```

## * Map:-

→ The Map is a very special Collection present in the collection framework library.

→ In map the elements are stored in a Keyvalue pair structure.

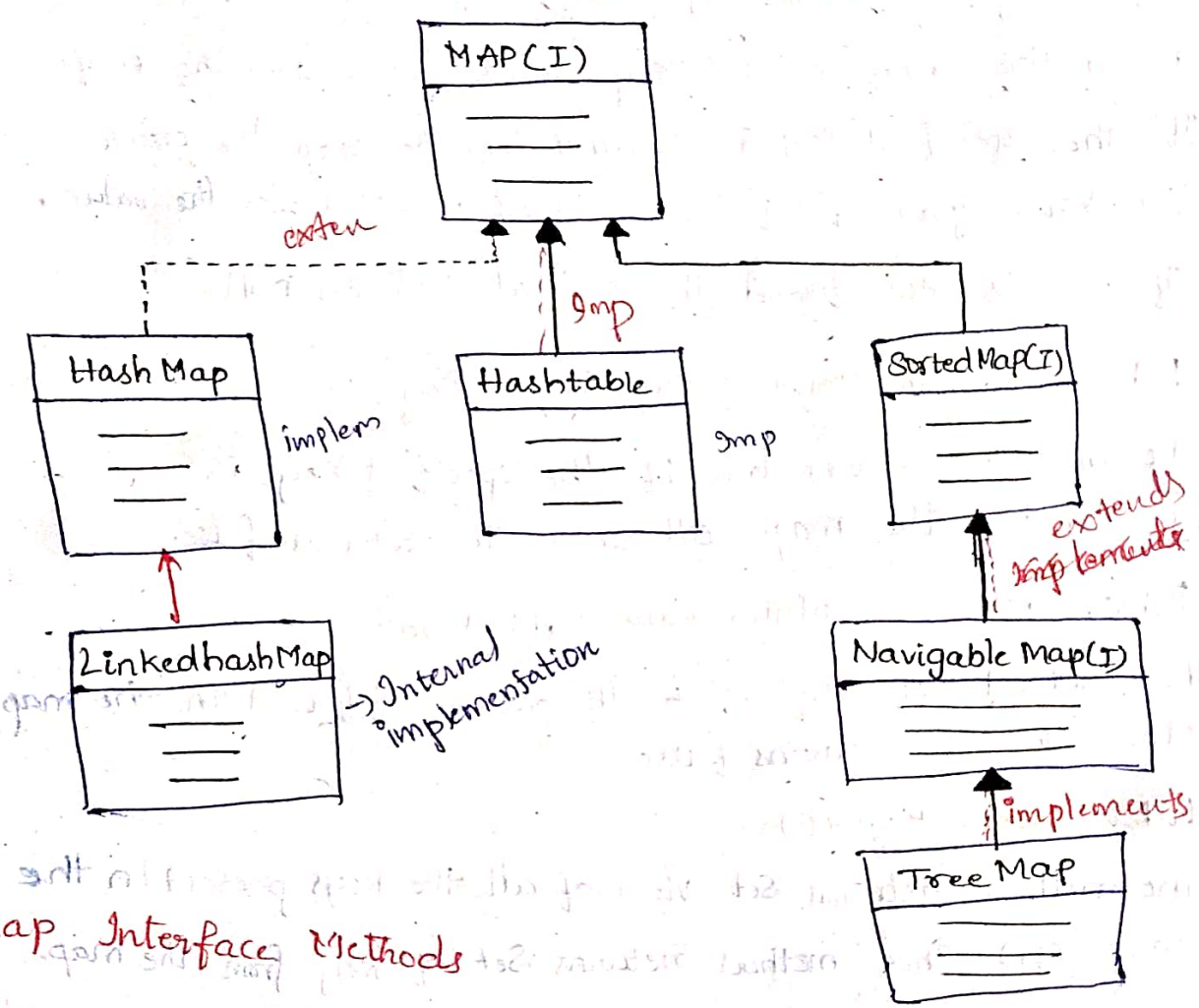
→ The key can be any type of object, value can also be any type of object.

→ The value is mapped to specified key.

→ The value is accessible only based on key.

→ Key must be unique, values can duplicate, without a key value cannot be accessed.

* Class diagram of map:-




Map Interface Methods

→ The map interface define a set of standard methods that can be operated by any type of map.

→ The methods are 1) public Object put (Object key, Object value)

→ This method associates the specified value to the specified key and store inside the map.

→ If the key is already existing it replacing the existing value with new value.

→ The first parameter becomes the key Second parameter is value.

2) Public Object get (Object key):

→ The method is used to access the value associated to the specified key. If that is found the method return the associated value. If the key is not found method returns null.

3) Public Object remove (Object key):

→ The method remove the key value pair from the map

→ If the specified key is found in the map the entire key value pair will be removed and returns the value.

→ If key is not found the method returns null.

4) Public boolean Contains key (Object key)

→ The method returns true if the specified key is found in the map otherwise it returns false.

5) Public boolean Contains value (Object value)

→ The method returns true if the value is found in the map otherwise it returns false.

6) Public Set KeySet().

→ The method returns Set view of all the keys present in the map (or) This method returns Set of key from the map.

7) Public Collection Values()

→ The method returns the collection of all the values from the map.

— Main

```
HashMap map1 = new HashMap();
Sopln ("is mapempty ? "+ map1.isEmpty());
Sopln (" Map size : "+ map1.Size());
Sopln (" map elements : ");
Sopln (map 1);
map1. put (12, "test");
map1. put (3.6, true);
map1. put (" demo", 4565);
map1. put (false, null);
map1. put (null, 't');
map1. put (12, "sample");  → old value
```

util. HashMap;
util. Set;
Util. Collections;

→ Old value is replaced with the new value if the key is same the value cant be changed

```
sopln ("is mapEmpty ? "+ map1.isEmpty());   false
Sopln (" Map Size : "+ map1. Size ());   5
Sopln (" map elements : ");   mapelements.
Sopln (map1);   {null=t, 3.6=true 12 test
                demo=4565, false=NULL}
Sopln (" map Size : "+map1.Size());   6
Sopln (" map elements: ");
Sopln (map 1);   {except NULL='t' An the elements
                will be there}
```

→ Set keys = map1. KeySet(); // getting all the keys from map

Collection values = map 1. ~~KeySet~~ Values(); // getting all the values from map

Sopln (" Printing all the keys of map); → Pring all the keys of map

```
for (Object key : Keys) {
    Sopln (Key);
}
```

3.6
null
false
12
demo

→ Sopln ("Printing all the values of the map");

```
for (object value : values)
{
Sopln (value);
}
```

Printing all the values of map
True
t
NULL
Test
4565

## DataStructures

22/11/24

### Hashtable :-

→ The hashtable datastructures stores the data in a table format, which has two columns.

1. key 2. value.

→ The Key will be always unique generated based on the hash algorithm.

→ We can make no. of entries with a unique key.

→ The value is fetched based on the key.

### HashMap :-

→ HashMap is implementation class of map interface.

→ It implements two marker interfaces.

(i) Serialisable (ii) Clonable

→ The hashmap is implemented by using hashtable datastructure

→ The hashmap class has overloaded constructor.

1. No parameter Constructor :-

```
Hashmap m1=new Hashmap();
```

→ This constructor creates an empty hashmap with the default capacity of 16 & load factor of 0.75.

2. Int parameter Constructor :-

```
Hashmap m2= new Hashmap(n);
```

→ This constructor creates empty Hashmap with the specified initial capacity n, with default load factor of 0.75.

3) Int and float parameter Constructor:

Hashmap m3 = new Hashmap (n,m)

→ This constructor creates empty hashmap with specified capacity and specified Load factor.

Resizing concept in Hashmap:-

→ The Hashmap resize based on the load factor.

→ When the Hashmap is filled upto load factor of capacity the Hashmap starts growing

Ex:- Hashmap m1 = new HashMap ();

→ This Hashmap default capacity is 16 and the Load factor is 0.76

→ This map starts growing when the capacity is filled.

load factor * capacity  fill ratio = Load factor x capacity

∴ 0.76 x 16 = 12 pair of elements

→ This is also known as fill ratio → "Fill ratio".

→ This Hashmap allows null as a key & value

→ The Hashmap do not preserve order of insertion.

Linked-Hashmap:

→ The LinkedHashmap is a child class of Hash Map.

→ The LinkedHashmap is built on hybrid datastructure Hashtable and Linkedlist.

→ The Hashtable is used to stress in key-value pair where as the linkedlist is used to pressure the order of insertion. Hence Linked Hashmap maintains the order of insertion.

## HashTable class:-

→ HashTable class is a implementation class of map interface.

→ It is implemented by using Hashtable Datastructure

→ Hashtable class and Hashmap class implementation wise they are same. The difference are

1) Hashtable doesnot supports null as key and value.

2) The method of Hashtable are synchronised. Hence they are thread safe.

Ex:-
```
import java.util.HashMap;
import java.util.Set;
public class Mainclass1
{
 P.S.V.M (S[] a) {
Hashmap<integer,string> Subjects = new Hashmap<integer,string>;
        subjects.put (101, "Java");
        Subjects.put (102, "SQL");
          "      "   (103, "DSA");
          "      "   (104, "Web");
          "      "   (105, "APTI");
        sopln (subjects);
  Set <Integer> subjcode = subjects.Keyset();
   for (integer code : subj-code) {
      String subjname = subjects.get(code);
        Sopln (code + " : " + subjname);

      }
    }
  }
```

Serialisable – converting java code into text format

22/11/24

## HashSet:

→ HashSet is a implementation classe of set interface.

→ It implements 2 marker interfaces.

1. Serialisable  2. Clonable

→ HashSet is built on HashMap (table) datastructure.

→ While adding the element into the set the hashcode of the element goes under key column and element goes under value column.

→ Since any two objects will not have same hashcode no duplicates are allowed in the set.

⇒ HashSet Set1 = new HashSet();

Set1.add(e1);

① get the hashcode of element

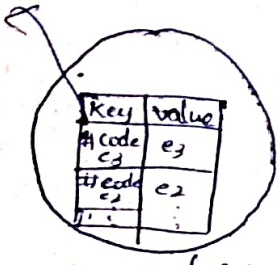
② make a entry in the table

hashcode → Key

element → value

Set1.add(e2);

Set 1




HashSet Object

⇒ HashSet Set1 = new HashSet();

Set1.add(e1);

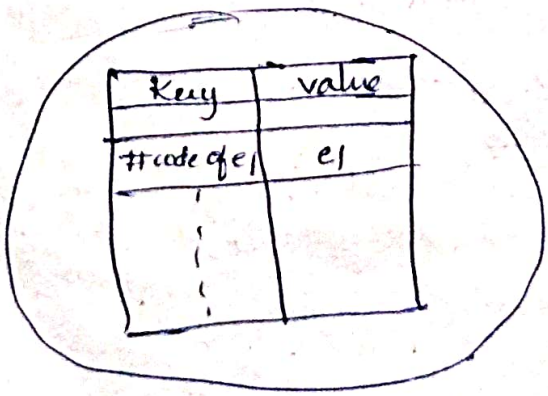
Set1.add(e2);

e2.equals(e1);

1. true → #codes are same, will not add

2. false → #codes different add the element

(this.hashcode() = e1.hashCode();)




HashSet object.

Example:

→ public class Textbook {

```
    int id;
    String title;
    String author;
    int edition;
    double price;
public Textbook (int id, String title, String author, int edition,
                double price)
  {
    this.id = id;
    this.title = title;
    this.author = author;
    this.edition = edition;
    this.price = price;
  }
public String toString ( )
  {
return "Textbook [id=" +id+", title="+ title + ", author="+
        author+", edition="+ edition+", Price="+
        price+"]";
  }
}
```

⇒ import java.util.ArrayList;

```
public class MainClass1 {
    public Static void main (String [ ] args) {
Textbook b1 = new Textbook (192, "Java", "Abhi", 3, 150.00);
Textbook b2 = new Textbook (322, "SQL", "Sharath", 2, 120.00);
Textbook b3 = new Textbook (232, "Web", "Harish", 4, 120.00);
Textbook b4 = new Textbook (343, "DSA", "Darshan", 2, 200.00);
Textbook b5 = new Textbook (4413, "Spring", "Vinod", 1, 250.00);
```

```
// Store all the books in ArrayList

ArrayList<Textbook> booklist = new ArrayList<Textbook>();
    booklist.add(b1);
    booklist.add(b5);
    booklist.add(b3);
    booklist.add(b4);
    booklist.add(b2);
    Sopln("BookDetails s");
    Sopln("-".repeat(45));
    Sopln("ID\t Title\t Author\t Edition\t Price");
    Sopln("-".repeat(45));
    ~~Sopln~~
    for (TextBook book: bookList) {
    Sopln( book.id+"\t"+ book.title+"\t"+ book.author+"\t"+
            book.edition+"\t"+ book.price);
    }

}

=> //filtering Based on title
    Sopln("Book Details"); //
    Sopln("-".repeat(45));
    sopln("ID\t Title\t Author\t Edition\t Price");
    Sopln("-".repeat(45));
    for (TextBook book:booklist) {
    ~~sop~~ if (book.title.equalsIgnoreCase("SQL")) {
    Sopln(book.id+"\t"+book.title+"\t"+book.author+"\t"+book.
            edition+"\t"+book.price);
    }
}
```

```
// filter based on author
sopln ("-".repeat (45));
Sopln ("ID\t Title \t Author \t Edition\t Price");
Sopln ("-".repeat (45));
for (TextBook Book : book list )
  {
    if (book.author.equalsIgnoreCase ("Darshan")) {
    Sopln (book.id +"\t"+ book.title+"\t"+book.author+"\t"+
          book.edition+ "\t"+ book.price);
  }
}
```

23/11/24

Sorting

ascending descending

Comparing Technique

① $n_1$ & $n_2$ are int value

if ($n_1 > n_2$)

"USING Relational Operator".

2) Using Arithimetic Operator.

$n_1$ & $n_2$ are two int value

$n_1 - n_2$ eg: 1) 10-20 → -ve diff

2) 20-10 → +ve diff

3) 10-10 → 0

Result : +ve, $n_1$ greater than $n_2$

-ve, $n_1$ lesser than $n_2$

0, $n_1$ same as $n_2$

* Sorting Booklist based on Id:-

```
→ import java.util.Comparator;
  public class SortByID implements Comparator<TextBook> {
    public int compare (TextBook b1, TextBook b2) {
        if (b1.id > b2.id)
          return +1;
        else if (b1.id < b2.id)
          return -1;
        else
          return 0;
      }
  }
```

import java.util.Collections;

→ SortByID idWise = new SortByID();

Collections.sort(bookList, idWise);

---

```
import java.util.Comparator;

public class SortByPrice implements Comparator<Textbook>{
    public int compare(TextBook b1, TextBook b2) {
        if (b1.price > b2.price)
            return +1;
        else if (b1.price < b2.price)
            return -1;
        else
            return 0;
    }
}
```

⇒

```
Package com.jspiders.p2;
//Textbook class is mutually comparable based on bookid
public class TextBook implements Comparable<TextBook>{
    int id;
    String title;
    String author;
    int edition;
    double price;
public TextBook(int id, String title, String author, int edition,
                double price){
    this.id = id;
    this.title = title;
    this.author = author;
    this.edition = edition;
    this.price = price;
}
```

@ Overside

```
public String toString(){
    return "TextBook: [ id = "+id+ ", title = "+ title +", author="
            + author + ", edition = "+edition +", price = "
            + price + "]";
}

public int compareTo(TextBook b){
    if (this.id > b.id)
        return -1;
    else if(this.id < b.id)
        return +1;
    else
        return 0;
    }
  }
}
```

⇒ Collections.sort(bookList); //mutually comparable & sorted

→ Tree Set is a Sorted Set

```
//Store all the books in TreeSet
TreeSet<Textbook> bookList = new TreeSet<TextBook>();
    booklist.add(b1);
    booklist.add(b5);
    bookList.add(b3);
    bookList.add(b4);
    bookList.add(b2);
    bookList.add(b10);
    booklist.add(b7);
    booklist.add(b8);
    booklist.add(b9);
    bookList.add(b6);
```

## TreeSet:

→ TreeSet is a implementation class of navigable set.

→ Navigable Set is child interface of SortedSet, Sorted-Set is a child interface of Set.

→ The Tree Set implements 2 marker interface

1. Serialisable 2. Clonable.

→ Tree Set is implemented by using binary tree data structure.

→ The Tree Set allows only comparable type objects by default.

→ The elements of the treeset are sorted in a ascending order by default.

→ The Treeset constructors are overloaded.

→ (i) No parameter Constructor;

```
TreeSet Set1 = new TreeSet()
```

→ This constructor create an empty treeset object which allows only comparable type objects.

→ These objects are mutually compared and sorted in ascending order by default.

(ii) Comparator type parameter Constructor;-

```
TreeSet Set1 = new TreeSet (cmp1);
```

cmp1 ---> is a type of Comparator.

→ This constructor creates an empty TreeSet which allows non-comparable objects.

→ The Objects of this Treeset sorted in ascending order by default based on the comparator specified.

→ This TreeSet follows non-mutual comparison.

26/11/24

## * Comparable Interface:

→ The Comparable interface is defined in java.lang package
→ This interface has one abstract method.

Public int compareto (Object o)

→ This method is used to compare the current object with the specified object based on the attribute.
→ The method returns int type, the method should return either positive (or) negative (or) zero.
→ This method is called method sorting of the collection happens.
→ This interface is used to achieve mutual comparision.

## * Comparable type objects:

→ Any class which implements the comparable interface is known as comparable type.
→ The objects of comparable type class is known as Compara-ble type Object.
→ The class must be implement the compare to) (compareto()) method to compare the same type of object based on the attribute.
→ In the java library String is a comparable String

Ex:-
```
public class MainClass3{
    P.S.V.M (S[] a) {
    Sopln ("MM S");
    String Str1 = "AAB";
    String Str2 = "AA";
  int n1 = Str1.CompareTo(Str2); //lexicographically
    Sopln(n1); //1
  int n2 = Str2.CompareTo(Str1);
    Sopln(n2); //-1
    Sopln ("MM E");
  }}
```

→ In the java library the wrapper classes are also comparable type.

```
Ex:- Integer i1 = 10;
     Integer i2 = 5;
     int n1 = i1.CompareTo(i2);
     int n2 = i2.CompareTo(i1);
     Sopln (n1); // 1
     Sopln (n2); // -1
     Double d1 = 12.54;
     //int n3 = d1.compareTo (i1); // Cannot compare double with integer
```

* Comparator Interface :-

→ This interface is defined in java.util package.

→ This interface contains one abstract method ie.,

method :- public int Compare(Object 1, Object 2)

→ This interface is used to achieve non-mutual comparison.

Comparator type Objects:-

→ A class which implements the comparator interface is known as comparator type & Objects of the class is known as comparator object.

→ The comparator Objects are developed to provide multiple sort option based on the attributes.

* ID wise :-

```
Ex:- import java.util.Comparator;
     public class SortByID implements Comparator<TextBook>
     {
     public int compare (TextBook b1, TextBook b2)
       {
         return b1.id.CompareTo (b2.id);
       }
     }
```

2. Price Wise :-

```
public int Compare (TextBook b1, TextBook b2) {
    return b1.price.CompareTo (b2.price); }
```

3. Based On Author :-

```
public int Compare (TextBook b1, TextBook b2) {
    return b1.author.CompareTo (b2.author); }
```

## * Priority Queue:-

→ The priority class is a implementation class of queue interface

→ It implements two marker interfaces.

1. Serialisable 2. clonable.

→ The Priority Queue is implemented by using priority Queue data structure.

→ The default capacity is "11".

→ The priority queue process the element based on the priority by default the priority is given for the least element. The least element will be removed first. The next priority is given to the next least element in the queue.

## Constructors of Priority Queue:-

→ The priority queue contains overloaded constructor.

### 1. No parameter Constructor:-

→ This constructor creates a empty queue with the default capacity of 11.

```
PriorityQueue pq = new PriorityQueue();
```

→ The Queue allows only comparable type objects because the queue is internally sorted in a ascending order.

### 2. Int parameter Constructor:-

→ Priority-Queue Pq = new priority queue (n).

n → int type

⇒ This constructor creates an empty priority queue with specified initial capacity.

⇒ The Queue allows only comparable type objects.

3. Comparator type parameter constructor:-

PriorityQueue Pq = new priority Queue (comp);

comp → comparator type object

→ This constructor creates an empty queue with a default capacity and allows non-mutual comparison based on the specified comparator.

4. int and comparator type parameter constructor:-

Priority Queue pq = new priorityQueue (n, comp);

n → int type comp → comparator type object.

→ This constructor creates an empty queue with the specified initial capacity and allows non-mutual comparison based on the specified comparator.

5. Collection type parameter constructor :-

Priority Queue pq = new priority Queue (C1)

C1 → any type of collection.

→ This constructor creates a queue with the elements of the specified type of collection. The collection should contain comparable type objects.

27/11/24

Cursor:-

→ A Cursor is a object used to travel elements of any types of collection.

→ There are 3 types of Cursors.

1. Enumerator → This cursor is not in use.
2. Iterator
3. ListIterator

Iterator:-

→ An Iterator is a type of cursor used to traverse elements of the collection in unidirection.

→ In the collection framework Library iterator is a interface which defines the methods to traverse the elements of the collection.

→ The methods of the interface are

1. public Object next():-

→ This method moves the cursor to the next element in the Collection.

→ If the collection doesn't have next element the method throws NO_Such_element_exception.

2. public boolean hasNext():-

→ This method returns true if the collection has next element otherwise the method returns false.

3. public object remove():-

→ This method removes the elements where the cursor is pointing.

→ The Iterator can be applied on anytype of Collection list, queue and set. Iterator is not a bidirectional cursor.

Helper method:-

→ In General framework development helper methods returns the object without disclosing the implementation class name one such helper method is public Iterator iterator().

→ On Calling this method it returns the object of Iterator type which can iterate the elements of the Collection.

List Iterator:-

→ The List Iterator is a bidirectional cursor which can traverse the elements in both forward and backward direction.

~~List Interf~~

List Iterator interface:-

→ This interface defines the standard methods of List Iter-ator.

→ This methods of list Iterator Interface is:-

1. public object Next():-

→ This method moves the cursor to the next element in the list. If the List doesn't have next element the method throws NoSuchElementException.

2. public boolean hasNext():-

→ This method returns true if the list has next element otherwise the method returns false.

3. public object remove():-

→ The method removes the element where the cursor is pointing.

4. public object previous():

→ This method moves the cursor to the previous element of the list, If previous element is not present the method throws no_such_element_exception.

5. public boolean hasprevious():-

→ This method returns true if the list contains previous element otherwise it returns false.

→ The List Iterator is applied only on List type collection. It cannot be applied on queue and set.

Helper Method:-

public ListIterator listIterator():-

→ This method returns an object of ListIterator type.

Note:- List Iterator is a child interface of Iterator.

Ex:-
```
import java.util.ArrayList;
import java.util.Iterator;
public class MainClass1 {
    p.s.v.m (S[] a) {
        Sopln ("MMS");
ArrayList <String> fruitlist = new Arraylist <String> ();
    fruitlist.add ("Apple");
    fruitlist.add ("Grape");
      "        "   ("Pineapple");
      "        "   ("Mango");
```

```
fruitlist.add ("Orange");
"        "    ("Kiwi");

Iterator <String> itr1 = fruitlist.iterator();
while (itr1.hasNext()) {
  String fruit = itr1.next();
    Sopln (fruit);
}
Sopln ("Using second iterator");
Iterator <String> itr2 = fruitlist.iterator();
  while (itr2.hasNext()) {
      String fruit = itr2.next();
        Sopln (fruit);
    }
    Sopln (" MME");
  }
}
```

→ If one (iterator) is used then again we can't use it back. So another (iterator) is creator.
cursor cursor

for ListIterator :-

```
List Iterator <String> itr1 = fruitlist.list Iterator();
  Sopln ("List items in forward direction");
  while (itr1.hasNext()) {
        String fruit = itr1.next();
          Sopln(fruit);
    }
    Sopln ("List items in reverse direction");
  while (itr1.hasprevious()) {
      String fruit = itr1.previous();
      Sopln (fruit);
  }
```

→ from jdk-1.5 onwards the enhanced forloop is introduced as a replacement of iterator syntax. It is a simplified version of iterator.

→ The cursor is applicable only on iterable Type coll the cursors cannot be applied on map because map is not a iterable.

For priority Queue:

```
import java.util.Iterator;
import java.util.priorityQueue;
public class MainClass {
    P.S.V.M (S[] a) {
    Sopln ("_ - - - - - - - - - - - - - - ");
priorityQueue <String> pq = new PriorityQueue <String> ();
    pq.add ("Apple");
    pq.add ("Grape");
    pq.add ("Pineapple");
    pq.add ("Kiwi");
    pq.add ("orange");
  Sopln("Queue elements using for-each loop");
  for (String fruit : pq) {
      Sopln(fruit);
    }
  Sopln ("Queue elements using iterator loop");
  Iterator <String> itr = pq.iterator();
  while (itr.hasNext())
   {
    String fruit = itr.next();
    Sopln (fruit); }
  Sopln ("polling the element of PQ");
  while (! pq.isEmpty()) {
     String fruit = pq.poll();
     Sopln (fruit);
    }
  Sopln ("NME");
  } }
```

⇒ Add all elements of one list to another

```
ArrayList L1 = new ArrayList();
  l1.add(10);
  l1.add(20);
   "     (30);
         (40);
         (50);
ArrayList L2 = new ArrayList();
  l2.add(100);
  l2.add(200);
  l2.add(300);
Sopln("First list");
Sopln(l1);
Sopln("Second List");
Sopln(l2);
Sopln("adding all elements");
l1.addAll(l2);
Sopln("First List");
Sopln(l1);
```

⇒

```
ArrayList L1 = new ArrayList();
  l1.add(10);
  l1.add(20);
  l1.add(30);
   "     (40);
   "     (50);
ArrayList L2 = new Arraylist();
  l2.add(100);      l2.add(20);
  l2.add(200);      l2.add(30);
  l2.add(300);      l2.add(10);
l1.removeAll(l2); // (40,50)
l1.retainAll(l2); // (10,20,30)
```

```
Sopln (l1.containAll(l2)); //false
Sopln (l2.containAll(l1)); //true
Sopln (l1); // [10,20,30]
```

Remove All :- remove elements in l1 that are common in

Retain All :- ~~Uncommon~~ Uncommon elements will be removed & common elements will be there

28/11/24

# Multithread

## Threads:-

Def:- A thread is a light weight process ie, An operation or program that is currently under execution and consumes less amount of CPU and memory.

→ Threads can be used by programmer to perform multitasking at a program level.

⇒ Default threads in java:-

→ Whenever a program begins execution the JVM internally creates the following 3 threads.

1. main thread
2. Garbage Collector
3. Thread Scheduler.

→ The main thread is responsible for executing the code written by the programmer.

→ The garbage collector is a thread created by JVM which is responsible for memory management in the heap area.

→ The thread scheduler is responsible for managing the life cycle of threads in java.

→ The thread scheduler and garbage collector are "DEMON THREDS"

** ⇒ DAEMON Threads are low priority ~~thread~~ background threads that are not under the programmers control.

## Creating threads in java:-

→ As a java programmer we can create threads in 2 ways.

1. By extending thread class.
2. By implementing Runnable interface.

### 1. By extending thread class:-

Steps:-

1. create a class that extends "java.lang.Thread".
2. Override "run()" method of thread class.

[Write the code to be executed by new thread inside() method]

3. Create an object of target class

4. Call "Start()" method to begin thread execution.

Ex:-

```
class Demonstratet
class Demothread extends thread {// 1
  @ Override
   public void run() //2
    {
    for (int a=1; a<=5; a++)
      {
      sopln ("demo thread ------ ");
      }
    }
  }
public class MainClass 1 {
    Public Static void main (S[] a) {
     sopln ("Program Starts");
Demothread dt = new Demothread(); //3
     dt. Start(); //4
for (int a=1; a<=5; a++)
  {
  sopln ("main thread ------ ");
  }
sopln ("Program ends");
  }
}
```

2) By implementing Runnable interface:-

Steps:-

① Create a class that implements "java.Lang.Runnable"

② Override "run()" method of Runnable interface.

[Write the code to be executed by new thread inside run() method].

3. Create an Object of target class.
4. Create an object of thread class by passing instance of target class.
5. class "Start()" method to begin thread execution.

Ex:-

```
class SampleThread implements Runnable //1
{
@Override
  public void run() { //2
   for (int a=1; a<=5; a++)
    {
     Sopln ("Sample thread ----");
    }
  }
}
public class Mainclass 2 {
  public static void main (String[] args)
   {
    Sopln ("MMS");
   SampleThread st = new SampleThread ();  //3
    Thread th = new thread (st); //4
      th.Start(); //5
   for (int a=1; a<=5; a++)
    {
     Sopln ("main thread ----");
    }
   Sopln ("Program Ends");
```

***
Note

→ Implementing runnable interface is considered as the recommended approach to create thread as it offers more flexibility for inheritance.

→ An actual thread is created in java only when we call the start() method.

29/11/24

Roll of Start() method :-

→ An actual thread is created ~~be~~ in java only when we call the start method.

→ The start method will internally perform the following 3 steps:-

1. Register with JVM.
2. Allocate ~~Register~~ Resources.
3. Internally call "run()" method.

⇒ 1. It registers thread object with JVM.
2. It allocates resources ~~method~~ needed for execution.
3. It internally calls run method.

What happens if we call run method directly instead of calling start method.

→ The code present inside run method will execute as a part of the main thread and no new thread will be created.

Thread.currentthread();

→ It is a static method of thread class.

→ It returns an object of thread class to represent the currently executing thread.

Syntax:-

```
Thread th = Thread.currentthread();
```

Thread properties :-

→ All ~~the~~ Threads in java will have the following 3 properties.

1) ID 2) Name 3) Priority.

ID:-

→ It is an integer value assigned by the JVM to keep track of the thread.

Name:

→ It is a string value which can be assigned either by the JVM (or) by the programmer.

Priority:-

→ It is an integer value that ranges b/w 1 to 10.

1 - lowest priority, 5 - average priority, 10 - highest priority.

→ We can access the properties of thread by using the following getter methods of thread class.

Id → getID

Name → getName

Priority → getPriority

→ We can assign thread properties by using the following setter methods of thread class.

Name → setName (String)

Priority → setPriority (int).

Note:-

→ ID can only be accessed by the programmer but it cannot be assigned.

Ex:-
```
public class Program1
{
    P.S.V.M (S[] a) {
    Sopln("Program Starts");
    for (int a=1; a<=5 ; a++)
    {
        Thread th = Thread.CurrentThread();
Sopln("ID:"+th.getID()+"Name:"+th.getName()+
    "Priority:"+th.getPriority()); }
```

```
Sopln ("Program Ends");
  }
}
```

Ex:-

```
class Example thread implements Runnable
{
@Override
    public void run() {
      for(int x=1; x<=5; x++) {
    thread th = thread. currentthread();
Sopln ("ID: "+ th.getID() + "Name: "+ th.getName() + "Priority
      → th.getpriority());
    } }
  }
public class MainClass3 {
    p.s.v.m (S[] a) {
    Thread th1 = new thread (new Example Thread());
            th1.setName ("a25");
            th1.Set Priority (10);
    thread th2 = new thread (new Example thread());
            th2.setName ("M17");
            th2.Set priority (1);
            th1.Start();
            th2.Start();
        }
    }
```

→ The priority of a thread comes into picture only when there is a resource ~~crutk~~ Crunch.

## Synchronization:-

→ Synchronisation is the concept of making a thread (Resource threadsafe.

→ When a method is declared with synchronised keyword it can only be accessed by one thread at a time.

→ We can perform synchronization by using the keyword synchronized.

→ If all the methods in a class are declared with synchro-nized keyword then we can say the class is thread safe.

Ex:-1

```
class Document {
    void read() {
    }
    Synchronized void write()
    {
    }
}
```

2)

```
class BookMyShow
{
    void browse()
    {
    }
    Synchronized void Payment
    ()
    {
    }
    void Selectseat()
    {
        ≡
        ≡
        Synchronised
        {
        ≡
        }
    }
}
```

## Thread.Sleep():-

→ Sleep is a static method present in thread class.

→ It can be used by the programmer to pause program execution for a certain time interval.

→ Sleep method is risky and may cause interrupted exception.

Ex:-

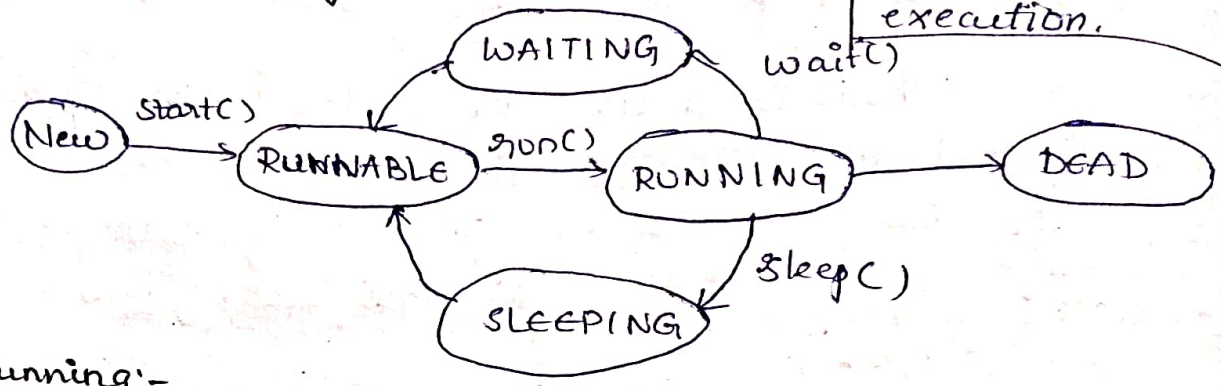
```
public class program1
{
    p.s.v.m (S[] a) {
        Sop ln ("Program Starts");
        try {
            Thread.Sleep(5000);
        }
```

```
Catch (InterruptedException e) {
        e.printStackTrace();
    }
    Sopln ("Program ends");
  }
}
```

New :- In this state the thread is a newly created object.

Runnable :- In this state the thread is registered with the JVM & ready to begin execution.

Thread Life Cycle :-




Running:-

→ In this state the thread is currently getting executed

Waiting:-

→ In this state the thread execution is paused by the JVM.

Sleeping:

→ In this state thread execution is paused by the programmer.

DEAD:-

→ In this state the thread has completed execution.

⇒ RACE CONDITION :-

→ Multi threads competing for same resources is called as race condition.

→ Race condition may lead to data inconsistency.

→ We can prevent Race condition scenario by using Synchronized Keyword.

⇒ DEAD LOCK:

→ In a dead lock scenario two / more threads are mutually waiting for eachother to complete execution.

→ In a deadlock scenario as no thread is accessing the resource the resource become ideal.

→ We can overcome deadlock scenario by using inter process thread communication.

2/12/24

F

# FILE HANDLING

File:-

A file is a container that holds information and has a well known extension.

Folder / Directory:

A folder is a container that holds files & sub folders.

File System:

It is an arrangement of files and folders in a computer.

→ file handling is the concept of manupulating the files & folders present in a computer.

→ The java library provides several readymade programs which can be used by the programmer to perform file handling operations.

→ All the predefined programs related to file handling can be found inside java.io package.

Creating a folder:

```
import java.io.File;
public class Program1 {
    P.S.V.M (S[] a) {
File ref = new File("C:/FILE IO");
boolean flag = ref.exist();
```

```
if (flag == false) {
    ref.mkdir();
    Sopln ("folder created");
    }
else {
    Sopln ("folder already exist");
    }
}
}
```

Creating a new file using java

```
import java.io.File;
import java.io.IOException;
public class Program2{
  P.S.V.M (  ) {
  File ref = new File ("C:/FILEIO/Demo.txt");
   boolean flag = ref.exists();
   if (flag == false) {
     try {
       ref.CreateNewFile();
       Sopln ("file created");
```

Deleting a file/folder:-

```
import java.io.FILE;
public class Program3 {
    P.S.V.M (S[] a) {
  File reg = new File ("C:/FILEIO/Demo.txt");
   boolean flag = reg.exists();
    if (flag == true) {
      reg.delete();
        Sopln ("deleted");
      }
```

```
else {
    sopln ("file folder does not exist");
    }
}
```

Eg:- finding path & length:-

```
import java.io.file;
public class program 4 {
    p.s.v.m (s[] a) {
    File ref = new ref
```

Java.io.File:-

→ File is a predefined program present in java.io.package.

→ It provides several predefined methods that can be used by the programmer to perform basic file manipulations.

1. exists():-

→ This method returns a Boolean value that indicates whether a file/folder is present in the given location.

true: present

false: absent

2. mkdir():

→ This method creates a folder/directory in the given location.

3. CreateNewFile():-

→ This method creates a new file in the given location, it is a risky method and may cause exception.

4. delete():-

→ This method deletes a file/folder in the given location.

5. getAbsolutePath():-

→ This method returns a string value that provides the location referenced by the current file object.

6. length():-

→ This method returns the character count in the current file as a long type value.

7. List():-

→ This method returns a string [ ] which contains the names of all the files and folders in current location.

8. isFile():-

→ This method returns a Boolean value which indicates true when the file object is assigned to a physical file.

9. is Directory ():-

→ This method returns a Boolean value which indicates the when the file object is assigned to a folder.

P-10

3/12/24 \n - escape sequence

## ⇒ Writing character Data to a file:

~~i) P~~

→ We can write character data from java prg to a text file using the following pre-defined programs in java.io package

(i) FileWriter (ii) Buffered Writer (iii) Print Writer

(i) java.io.FileWriter:

Steps:-

① Create an object of FileWriter.
② Write data to fileWriter ~~class~~ object using "write()" method.
③ Transfer data to physical file by calling "flush()" method
④ Close FileWriter object by calling "close()" method.

```
import package com.jspiders.filehandling;
import java.io.FileWriter;
import java.io.IOException;
public class Mainclass1{
    P.S.V.M(S[] a) {
    FileWriter ref=NULL;
      try {
        ref=new FileWriter("C:/FILEIO/log.txt");
        ref.write("JAVA");
        ref.write("\n");
        ref.write("SQL");
        ref.write("\n");
        ref.write("WEB");
        ref.write("\n");
        ref.write(65);
        ref.write("\n");
```

→ Here the drawback is it directly won't goes to the next line it will print in same line we have to do manually

```
char[]ch = {'J', 'S', 'P', 'I', 'D', 'E', 'R', 'S'};
ref.write(ch);
ref.flush();
Sopln("Written Completed");
}
catch (IO Exception e) {
        e.printStackTrace();
}
finally {
    try {
        ref.close();
    }
    catch (IO Exception e) {
        e.printStackTrace();
    }
}
}
}
```

Here String data is only supported as character

(ii) java.io.BufferedWriter.

→ It internally requires a file Writer object.

~~import java.io.BufferedWriter;~~
~~import java.io.IOException;~~
~~import java.io.~~

① Create an Obj of BufferedWriter
② Write data to bufferedWriter Object using "write()" method.
③ Transfer data to physical file by calling "flush()" method.
④ Close BufferedWriter Object by calling "close()" method.

```
→ import java.io.BufferedWriter;
  import java.io.FileWriter;
  import java.io.IOException;
  public class Mainclass2 {
      P.S.V.M (S[] a) {
        BufferedWriter ref = null;
        try {
          ref = new BufferedWriter(new FileWriter("C:/FILEIO/
                                                  list.text"));
          ref.write("Banglore");
          ref.newLine();
          ref.write("Chennai");
          ref.newLine();
          ref.write("Delhi");
          ref.newLine();
          ref.write(97);
          ref.newLine();
          char[] ch = {'j', 'a', 'v', 'a'};
          ref.write(ch);
          ref.flush();
          sopln("Write Completed");
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            try {
              ref.close();
            }
            catch (IOException e) {
              e.printStackTrace();
            }
        }
  }
}
```

Drawback

BufferedWriter is not having ability to directly interact with file.

→ Here we can give ~~write~~ only String type (or) character type data

(iii) java.io.PrintWriter

Steps

① Create an object of PrintWriter.

② Write data to printWriter object using "println()" (or) "print()" method.

③ transfer data to physical file by calling "flush()" method.

④ Close PrintWriter object by calling "close()" method.

Program:-

```
import java.io.Writer;
import java.io.IOException;
public class Mainclass3 {
  P.S.V.M(S[] a) {
    PrintWriter ref=null;
  try {
    ref = new PrintWriter("C:/FILEIO/config.txt");
    ref.println("Karnataka");
    ref.println(100);
    ref.println(6.78);
    ref.println(true);
    ref.flush();
    Sop/n ("Write Completed");
  }
  catch (IOException e) {
      e.printStackTrace();
  }
  finally {
      ref.close();
  } } }
```

## ⇒ Reading Character Data from a file:

→ To read character data from a file the java library provides following two readymade programs in java.io.package

1) FileReader 2) BufferedReader

### (i) java.io.FileReader :-

Steps :-

① Create an object of file Reader

② Read data from file using "read()" method.

③ Close FileReader object by calling "close()" method.

drawback :-

→ For each character it was taking each iteration so time consumption is more

→ And here we need to do typecasting.

```
→ import java.io.FileReader;
  import java.io. IOException;
  public class Mainclass4 {
      P.S.V.M(S[] a) {
      FileReader ref=null;
      try {
          ref = new FileReader ("C:/FILEIO/Demo.txt");
          int val= ref.read();
          while (val != -1){
              sopln ((char)val);
              val= ref.read();
          }
      }
      catch (IOException e) {
          e.printStackTrace();
      }
```

```
finally {
    try {
        ref.close();
    }
    catch (IOException e){
        e.printStackTrace();
    }
}
}
}
```

O/P:-
ABC
abc

② BufferedReader:- (java.io.BufferedReader)

Steps:-
① Create an object of BufferedReader
② Read data from file using "readline()" method.
③ close BufferedReader object by calling "close()" method.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class MainClass5 {
    p.s.v.m.(S[] a) {
     BufferedReader ref=null;
    try {
    ref = new BufferedReader (new FileReader ("C:/FILEIO/readme.txt"));
    String str= ref.readLine();
     while (str! =null){
          Sopln (str);
         Str= ref.readLine();
     }
}
```

```
catch (IOException e) {
    e.printStackTrace();
  }
finally {
    try {
        ref.close();
    }
    catch(IOExceptiion e)
    {
    e.printStackTrace();
    }
  }
}
}
```

4/12/24

Serialisation & De-Serialisation:-

Serialisation:

→ It is the concept of writing the java object into a text file.

→ During serialization in java object is converted into a stream of bytes.

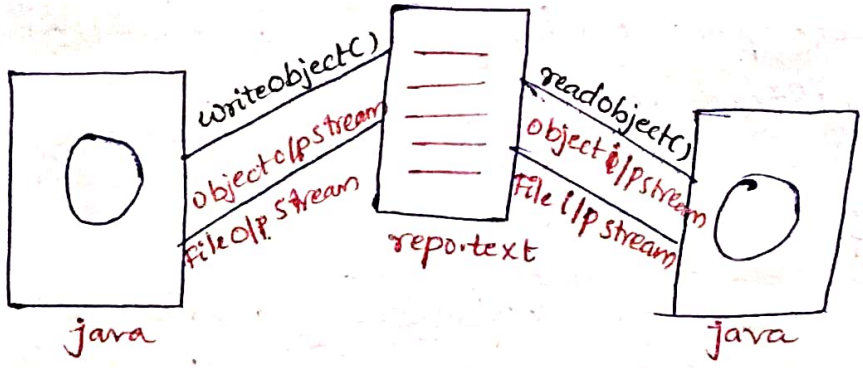
→ The concept of serialization is known as "MARSHALLING".

→ The process of serialization only works with serializable type objects.

De-Serialization :-

1. It is the concept of reading a java object stored in a text file.
2. During Deserialization the object is reconstructed from a stream of bytes.
3. The process of Deserialization is also known as "UNMARSHALLING".

4.




Serialization Code:-

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.Serializable;
class Employee implements Serializable {
 int eid;
 double ctc;
}
Public class Maindass 4 {
 P.S.V.M(S[] a) throws IOException {
   Employee emp = new Employee();
     emp.eid = 101;
     emp.ctc = 4.5;
FileOutputstream fos = new FileOutputstream("C:/FILEIO/reposi
                                -tory.txt");
 ObjectOutputStream oos = new ObjectOutputStream(fos);
   oos.writeobject(emp);
   oos.close();
   fos.close();
   sopln("serialization Completed");
 }
}
```

De-serialization:-

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
public class Mainclass5 {
 p.s.v.m(SC[] a) throws IOException, ClassNotFoundException {
FileInputStream fis = new FileInputStream("C:\FILEIO\repository.txt");
ObjectInputStream ois= new ObjectInputStream(fis);

Employee emp= (Employee) ois.readObject();
 Sopln (emp.eid);
 Sopln (emp.ctc);
 ois.close();
 fis.close();
 }
}
```

⇒ We can prevent serialization by using "transient" Keyword
It is only for Datamembers.

File Permissions:-

→ Every file that is created in a computer will have the following 3 permission attributes.

1. Read 2. Write 3. Execute.

→ It is possible for a java programmer to access and assign file permissions using file class.

→ We can access the file permissions using the below methods of fileclass.

CanRead, CanWrite, CanExecute.

→ We can assign permission to a file using the following methods.

Read → setReadable (false)
Write → setWritable (false)
Execute → setExecutable (false).

Ex:- import java.io.File;
import java.io.IOException;

```
public class Example {
    p.s.v.m (S[] a) {
File f = new File ("C:/FILEIO/Q.txt");
 if (f.exists() == false) {
  try {
    f.CreateNewFile();
    Sopln ("File Created");
   f.SetReadable (false);
   f.Set Writable (false);
   f.Set Executable (false);
    Sopln ("Permissions Assigned");
  }
  catch (IOException e) {
      e.printStackTrace();
  }
 }
}
}
```

Ex:- import java.io.File

```
public static void main (S[] a) {
 File f = new File ("C:/FILEIO/Q.txt");
  if (f.exists() == true)
   {
    Sopln ("Readable : " + f.canRead());
    Sopln ("Writable : " + f.canWrite());
    Sopln ("Executable : " + f.canExecute());
   }
}
}
```

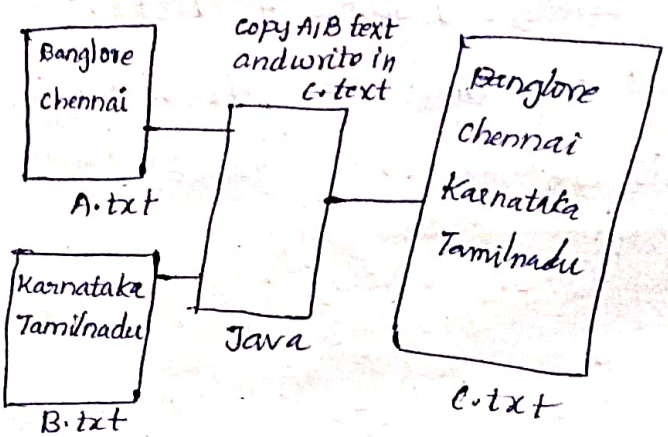
Activity-1:-

```
File f1 = new File ("C:/FILE IO");
  if (f1.exists() = = true)
   {
     String [ ] arr = f1.list();
  File f2 = new File (f1+"list.txt");
   if (f2.exist() = = false)
    {
     f2.CreateNewfile();
    }
  Print writer ref = new PrintWriter (f2);
  for (int i=0; i<arr.length; i++)
     {
     ref.Println(arr[i]);
     }
    ref.flush();
    ref.close();
    }
```

Activity-2:

```
public class Activity2 {
   p.s.v.m (S[] a) throws IoException{
  BufferedReader br = new BufferedReader (new FileReader("C:
                    /FILEIO/A.txt"));
 PrintWriter pw = new PrintWriter("C:/FILEIO/C.txt");
   String line = br.readline();
   while (line ! = null){
       Pw.println(line);
       line = br.readline();
    }
  br = new BufferedReader (new FileReader("C:/FILEIO/B.txt"));
      line = br.readline();
```

```
while (line! = null) {
    pw. println (line);
    line = br. readline();
  }
  br. close();
  pw. close();
}
}
```




***Activity 3:- (merging data line by line);

```
public class Activity 3 {
    P.S.V.M (S[] a) throws IOException {
Buffered Reader Br1 = new Buffered Reader (new FileReader ("C:/FILEIO
                                  /A.txt"));
Buffered Reader Br2 = new Buffered Reader (new FileReader ("C:/FILEIO
                                  B.txt"));
PrintWriter writer = new printWriter ("C:/FILEIO/C.txt");
    String Line1 = Br1. readline();
    String line2 = Br2. readline();
  while (line1! =null || line2! =null)
    {
      if (line1! = null)
        {
          writer. println (line1);
          line1 = Br1. readline();
        }
      if (line 2! =null)
        {writer. println (line2);
        line2 = Br2. read line();
        }
    }  writer. close();
       Br1. close();
       Br2. close(); } }
```

# Memory Management in Java

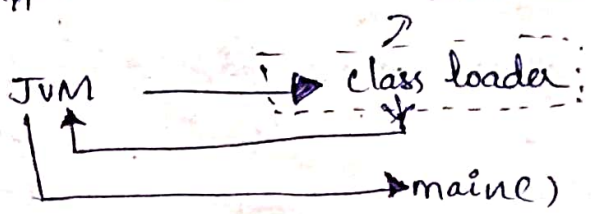
→ In java entire memory management is handled by the jvm at run-time.

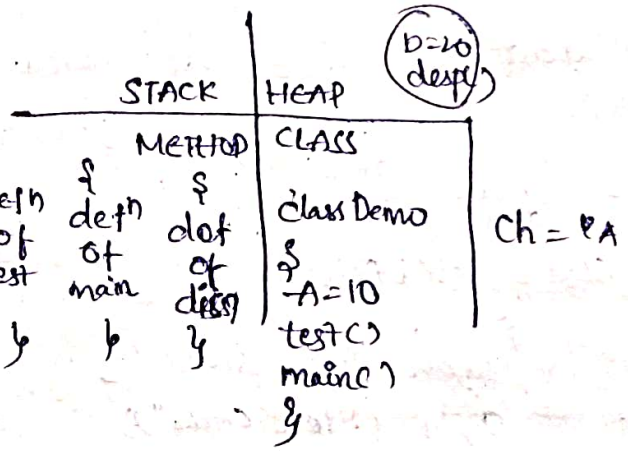
→ The JVM makes use of the following memory areas :-

(i) class Area
(ii) Heap Area
(iii) Method Area
(iv) Stack Area

```
Class Demo {
Static int a=10; //char Ch='A'
  int b=20;
Static void test()
  {
    }
void disp()
{
}
p.s.v.m(s[] a)
{
Demo ref = new Demo();
  }
}
```

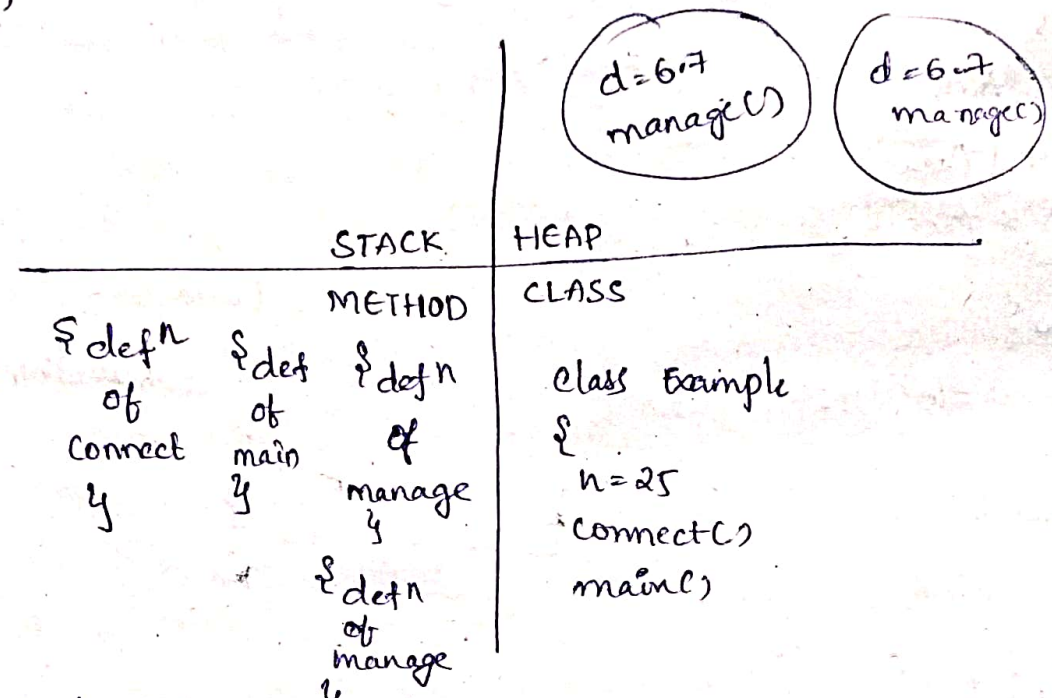
javac Demo.java
java Demo



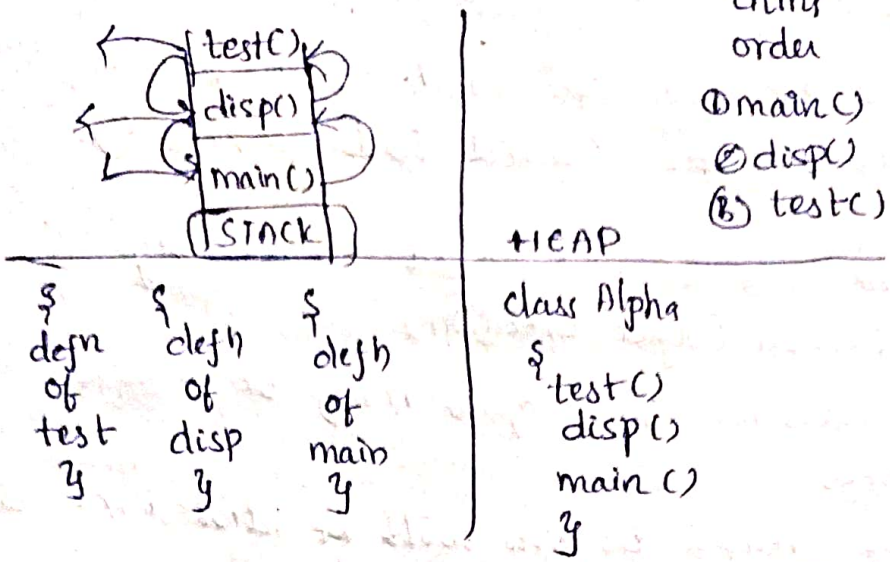

```
class Example
{
Static int n=25;
  double el = 6.7;
void manage()
  {
  }
Static void connect()
  {
  }
```

```
P.S.V.M (_)
{
    Example ex = new Example();
    Example ex1 = new Example();
    }
}
```




class loader → load all static members.
→ execute all the static blocks.

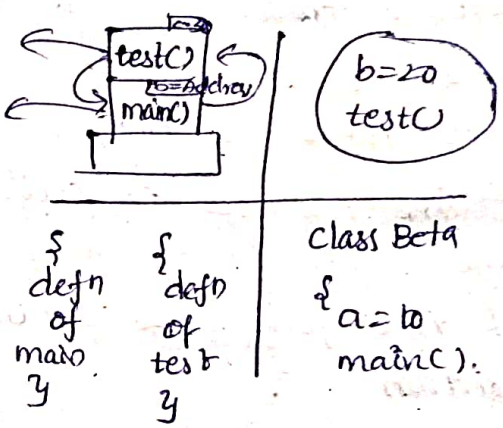
```
class Alpha
{
    static void test()
        {
        sopln ("test starts");
        sopln ("test ends");
        }
    static void disp()
        {
        sopln ("disp starts");
        test();
        sopln ("disp ends");
        }
    P.S.V.M (SC[] a) {
        sopln ("mm s");
        disp();
        sopln ("mm e");
        }
```




O/P
main starts
disp starts
test starts
test ends
disp end
main ends

LIFO → last in first Out
FIFO → first in first Out

```
class Beta
{
  static int a=10;
    int b= 20;
    void test()
      {
        int c=30;
        Sopln ("Executing test");
      }
  p.s.v. m(S[] a)
    {
      Beta b= new Beta();
      b.test();
    }
}
```




## Class Area:-

→ All static members are loaded to class area.

→ Static members are loaded to class area by the class loader.

→ Static member are loaded to memory by default during class loading time.

→ There exists only one copy of static members in memory.

→ Class Area is also known as static pool.

## Heap Area:-

→ The non-static members are loaded to heap area.

→ Non-static members are loaded to heap Area by JVM

→ non-static members are loaded to memory during instantiation (Object creation)

→ non static members multiple copies may be present in Heap Area.

→ All Objects are created in java will be alloted memory in heap Area

## Method Area:

→ All the method definitions are stored in the method Area.

## Stack Area:-

→ Stack Area is known as execution memory

→ All the method executions are done in Stack Area.

→ Stack Area follows a LIFO or FIFO order of execution

→ All local variables are alloted memory in stack area

## What is Class loader:-

Class loader is as subsystem of the JVM present inside the JRE.

→ It is responsible for.

- loading all the static members to memory
- executing all the static initialisation block.

→ The time interval in which class loader works is known as class loading time.