difference b/w abstract class & Interface.

Note: from JDK 1.8 we can define static concrete method. or default concrete method.

```
Public interface Inf
{
public static void m1()
{
S.O.P("static Concrete method");
}
public default void m2()
{
S.O.P("default Concrete method");
}
}
```

abstract	Interface
• can have both concrete as well as abstract method	• can have only abstract method. (only default/concrete, static method can be defined)
• Abstract class can have instance variables	• Interface cannot have instance variables (data member public static and final)
• abstract class can have constructors	• Interface can't have constructors
• Multiple inheritance is not possible	• Multiple inheritance is possible
• abstract class can inherit another class	• Interface cannot inherit a class

• Constructor chaining is possible	• constructor chaining is not possible.
• methods can be public, private etc.	• Methods are automatically public
• not possible to indicate JVM about an activity	• It is possible to indicate JVM about any activity through marker interface.

★ Why multiple inheritance is not possible in class but possible in interface.

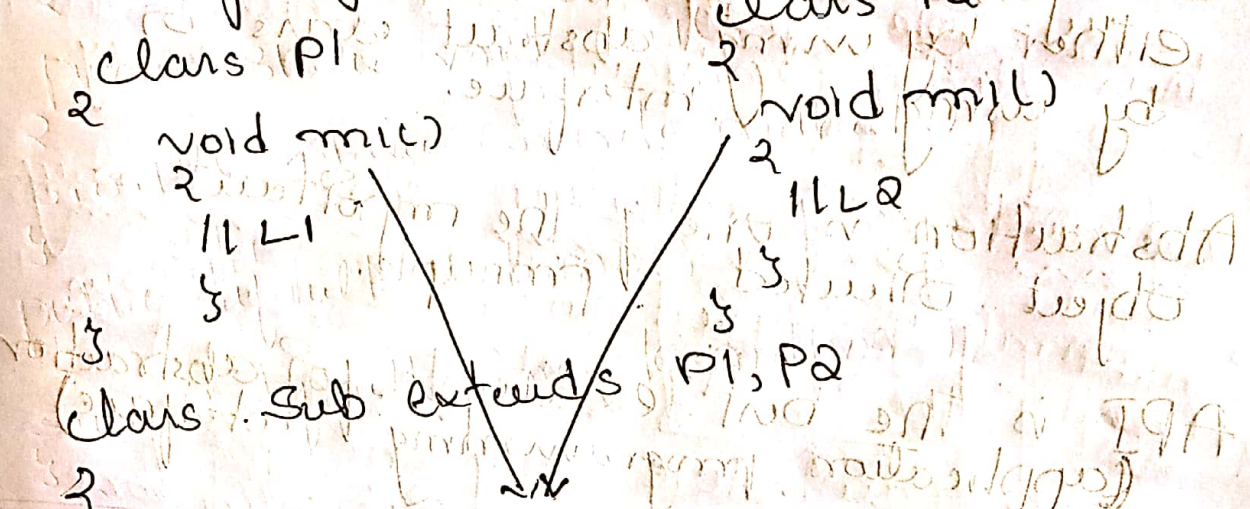
Reason 1: Ambiguity while Constructor chaining

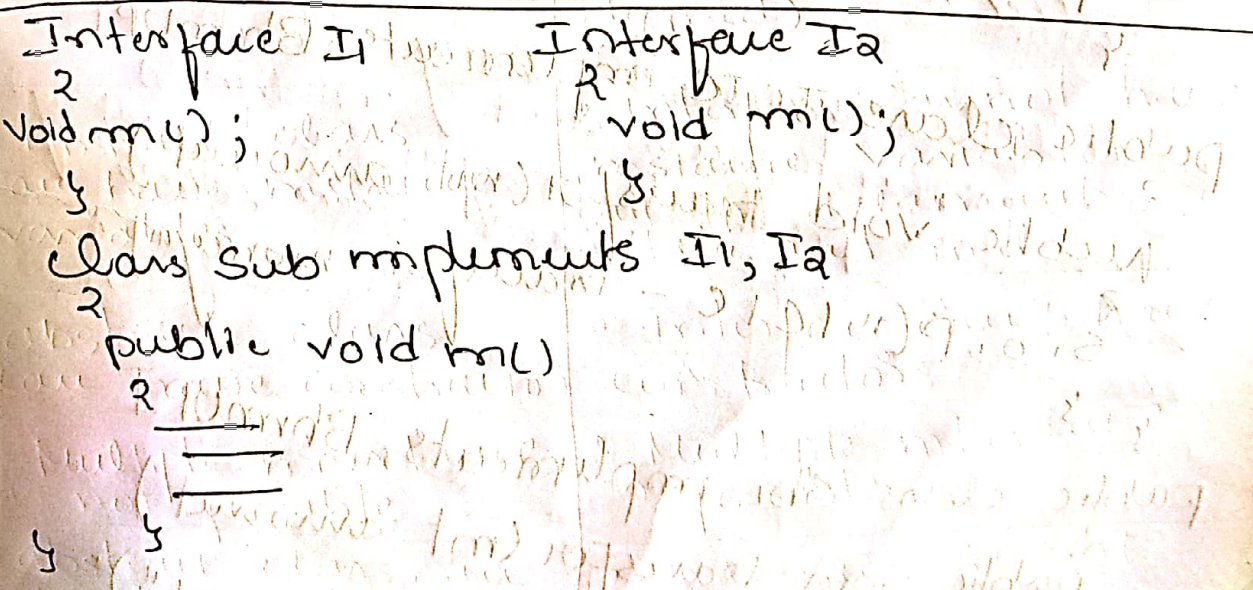
```
class P1                    class P2
{                           {
    P1()                        P2()
    {                           {
    }                           }
}                           }

class Sub extends P1, P2
{
    Sub()
    {
        super()
    }
}
```

?

Ambiguity while Constructor chaining is not possible in case of interface :: Interface cannot have a constructors

Reason 2:
Ambiguity while method execution





Abstraction

Abstraction is the process of mechanism of hiding the internal implementation details from the consumer by exposing only necessary functionalities.

In java we can achieve abstraction either by using abstract class or by using interface.

Abstraction is one of the important Object Oriented principle.

API is the best example for abstraction (Application programming interface)

API ex: JDBC API

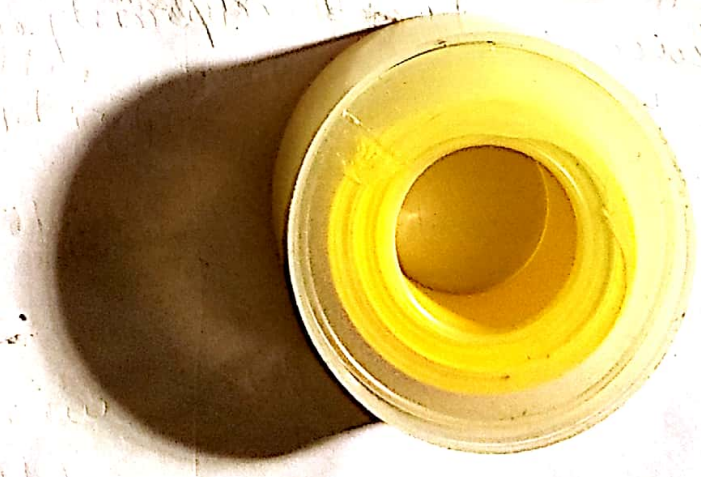
```
public interface BhimUPI
{
    public void transfer(int amount);
}

public class ICICI implements BhimUPI
{
    public void transfer(int amount)
    {
        S.o.P("ICICI - transfer money");
    }
}

public class SBI implements BhimUPI
{
    public void transfer(int amount)
    {
        S.o.P("ICICI - transfer money");
    }
}
```

```
public class PhonePe {
    public static void main(String args[])
    {
        BhimUPI upi = new ICICI();
        upi.transfer(5000);
    }
}
```

Purpose / advantages / abstraction

- To achieve loose coupling between Service & the consumer.

* what are the use of Interface.
- to achieve 100% abstraction.
- we can achieve multiple Inheritance
- It can be used as a coding contract or rules Repository.
Throug functional interface.
- it can be used to indicate JVM about certain activity through marker Interface.
-



Collection Frameworks

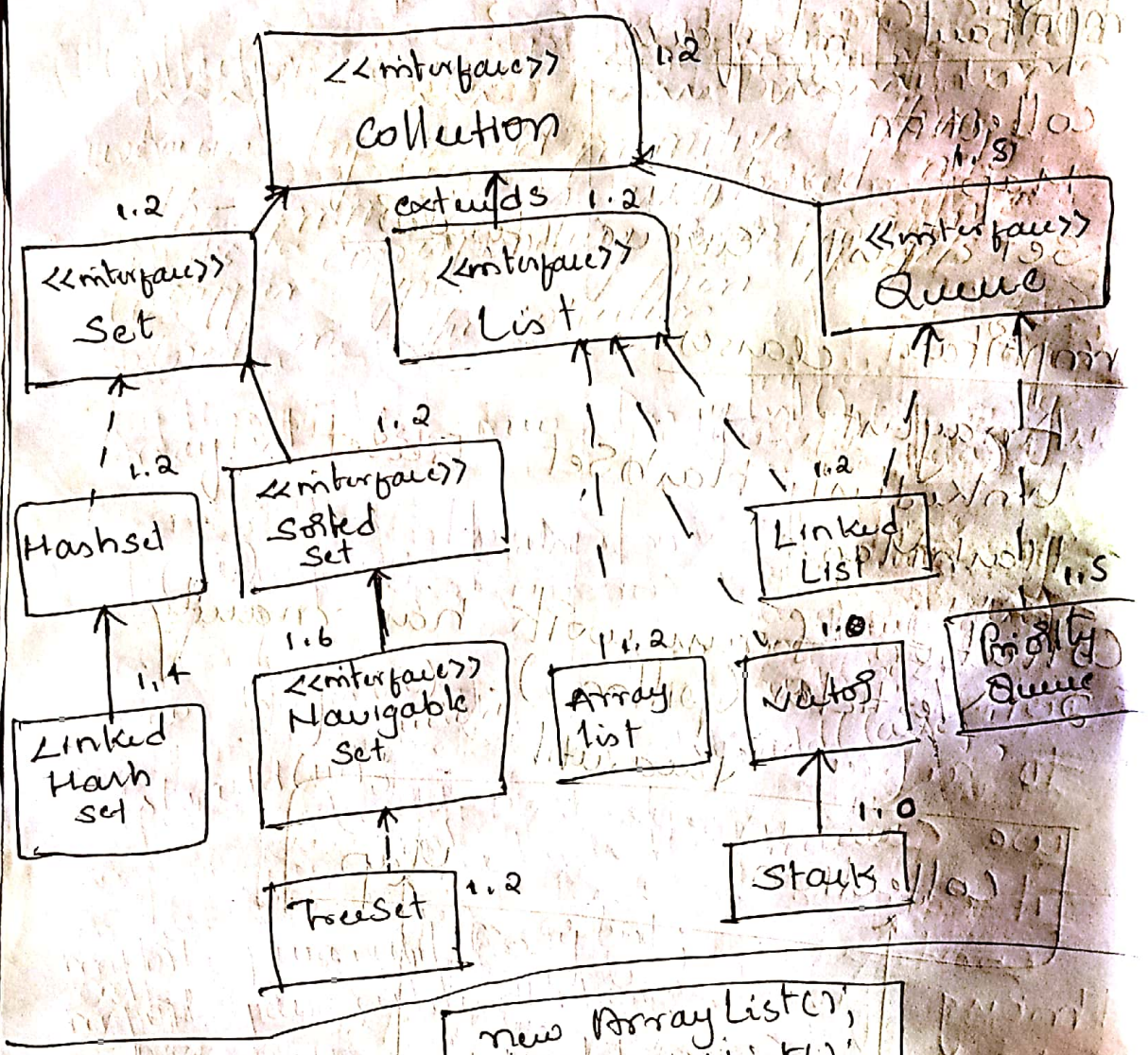
Collection framework is mainly used to store & deal with collection/group of data

Like an array, in collection also we can store group of data, but collection has lot of advantages over array.

Advantages of collection over an Array

- Array is fixed in size, whereas collection is dynamic.
- Array can store only one type of data. (Homogeneous). But collections can store different types of data.
- Array does not have methods to deal with data whereas, whereas collection has many utility methods.

Collection framework many inbuilt classes, interfaces and methods which are present in java.util package. Hence we need import them before the use.



Collection col = new ArrayList();
new LinkedList();
new Vector();
new stack();

interface reference

new Hashset();
new Treeset();
new LinkedHashset();

new priorityQueue();

// methods

all the methods are public & abstract

Methods of collection interface

boolean	add(E e)
boolean	addAll(Collection c)
void	clear()
boolean	contains(Object o)
boolean	containsAll(Collection c)
boolean	isEmpty()
Iterator<E>	iterator()
boolean	remove(Object o)
boolean	removeAll(Collection c)
int	size()
Object[]	toArray()

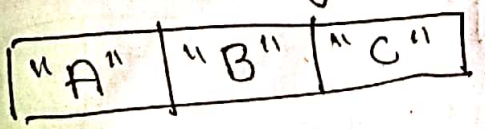
Generics

Generics is one of the feature of collection introduced from JDK 1.5

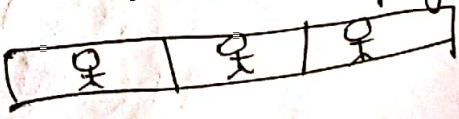
Generics defines the type of data of element type that can be stored in collection.

Commonly represented as <E>

Collection <String>



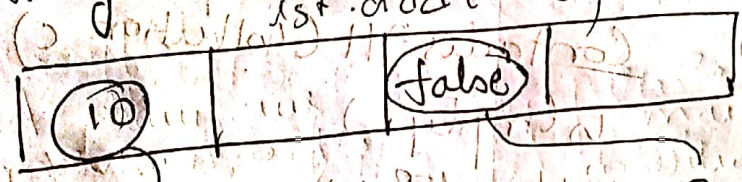
collection<Employee>



- Collection cannot store primitive data when we try to add primitive then the primitive is autoboxed into its corresponding non primitive wrapper type. Then it gets stored in the collection

ArrayList lst = new ArrayList();

lst.add(10)



lst.add(false)

lst.add("Abhi")

new Integer(10)
autobox

lst.add(10);

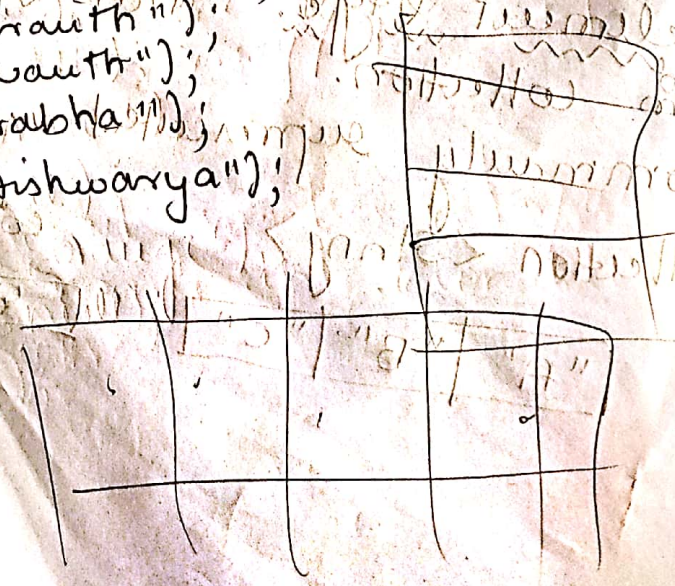
int

new Boolean(false)
auto box

lst.add(false);

boolean

```
import java.util.ArrayList;
import java.util.Collection;
public class ColMethodsDemo {
    P.S.V.M (                    )
{
Collection<String> csCol = new ArrayList<>();
csCol.add("Nagabhushak");
csCol.add("Karanth");
csCol.add("Ravanth");
csCol.add("Prabhat");
csCol.add("Aishwarya");
```



```
Collection<String> ecCol = new ArrayList<String>();
    eccol.add ("Reddy");
    eccol. add ("Sushil");
    eccol.add ("SS");
    ec col. add ("padma");

Collection<String> mechCol = new ArrayList
    <String>();
    mechcol.add ("Shrusha");

Collection<String> amazonCol = new ArrayList
<String>();
amazoncol.addAll(csCol);
amazoncol.addAll(ecCol);
amazoncol.addAll(mechCol);

S.o.p(amazonCol.size());
S.o.p(amazoncol.isEmpty());
amazonCol.clear();
amazonCol.remove("reddy");
 S.o.p(amazonCol);
amazoncol.removeAll(csCol);
S.o.p(amazonCol.contains("Nagabhushit"));
S.o.p(amazonCol.containsAll(ecCol));
Object[] arr = amazonCol.toArray();
```

Set Versus List

List	Set
• list is index based	• Set is not index based
• list can store duplicate data	• Set can't store duplicate data (can store only unique data)
• list maintains insertion order	• Set doesn't maintain any insertion order

```
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Set;

public class ListVsSet
{
    p.s.v.m.(String[] args)
    {
        List<String> lst = new LinkedList<String>();
        lst.add("red");
        lst.add("yellow");
        lst.add("red");
        lst.add("white");
        lst.add(null);
        lst.add("Orange");
        lst.add(null);
        S.o.p(lst);
```

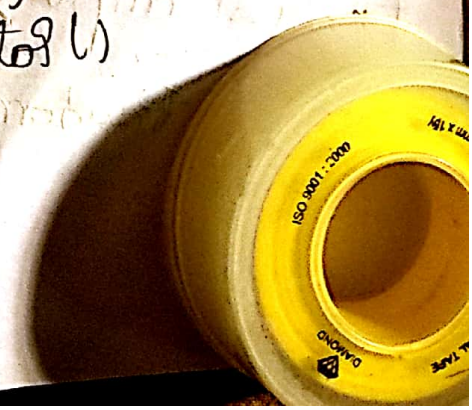
```
Set<String> set = new HashSet<String>();
    set.add("yellow");
    set.add("red");
    set.add("red");
    set.add("white");
    set.add(null);
    set.add("Orange");
    set.add(null);
     s.o.p(set);
  }
}
o/p:
[red, yellow, red, white, null, Orange, null]
[red, null, Orange, white, yellow]
```

LIST :

- List has index based methods
- List is the sub interface of collection present since JDK, since JDK 1.2
- List inherits all the methods from collection

Methods of List Interface

```
void add(int index, E element)
boolean addAll(int index, Collection c)
E get(int index)
int indexOf(Object o)
int lastIndexOf(Object o)
ListIterator<E> listIterator()
```



ListIterator<E> listIterator(int index)
E remove(int index)
E set(int index, E element)
List<E> subList(int fromIndex, int toIndex)
all the methods are public and abstract.

Arraylist
It is one of the implementation class of List interface, Present Since JDK 1.2, It has all the characteristics of list like
- it is index based
- can store duplicate/null elements
- maintains insertion order

Arraylist internally stores the data in the form of Array. and the initial capacity of Array is 10 and The incremental capacity is

$$\frac{\text{current capacity} \times 3}{2} + 1$$

Arraylist implements the datastructure called growable array/Resizable array.

Arraylist implements marker interfaces like Serializable, Clonable and Random Access

There are 3 Overloaded Constructors are present in care of ArrayList

- public ArrayList()
- public ArrayList(int initCapacity)
- public ArrayList(Collection c)

ArrayList uses Continuous memory.

Situations to use ArrayList:

- ArrayList is good for data retrieval & Search operation. Because time taken to Search any data in the entire list is Same.

Situations not to use ArrayList:

If we try to add the data in between the list then all the existing data gets shifted to the next position So ∴ of this Shift operation The performance becomes Slow

ex: method Overloading

public boolean (E e)
public void add(int index, E e)

---

public boolean remove(Object o)
public boolean remove(int index)

Hash-based collection

In java collection framework we have many inbuilt hash based collection are present.

hash-based collection

- HashSet
  - LinkedHashSet
- HashMap
  - LinkedHashMap
- HashTable

All the hash based collection internally implements the data structure called hash-table

The hash-table data structure internally requires hash-function or hash-method called

hashCode()

Syntax: public int hashCode()

The hashCode() returns an integer hash-code

SET

In java Set is an interface which extends collection interface

Set maintains uniqueness by not storing the duplicate elements

Set is not index based. Hence we dont have get method. So we cannot use for loop with Set

ex:

```
import java.util.HashSet;
import java.util.Set;

Public class Demo2
public static void main (_____)
 {
Set<Double> set=new HashSet<Double>();
  Set.add (515.84);
  Set.add (null);
  Set.add (25.57)
  Set.add (3147.8)
  Set.add (null)
  Set.add (25.57);
   S.o.p(set);
 }
}
```

o/p:

[null, 3147.8, 515.84, 25.57]

toString()

public String toString()

toString() is a method which is defined in Supermost class called Object.

The return type of the method is String.

Automatically toString() method returns the classname @ object address in the form of String.

"ClassName@Obj Address".

If we wish we can Override toString() method.

When we Override toString() method then we must return the object content.

Note: toString() method is already Overridden in String class and can all the wrapper classes.

ex: public class Mobile

```
{
String model, clr;
public static void main (_____)
{
Mobile m = new Mobile();
m.model = "iphonex";
m.clr = "Silver";
String str = m.toString();
S.o.P(str);
}
public String toString()
{
return this.clr + " " + this.model;
}
}
```

o/p: Silver iphonex

```
public class City
{
String name;
int pinCode;
public City(String name, int pinCode)
{
    this.name = name;
    this.code = pinCode;
}
public String toString()
{
  return this.name + " " + this.pincode;
}
P.S.V.M(_____)
{
City c1 = new City("Bangaluru", 560050);
City c2 = new City("Vijaypete", 57212)
   S.o.P(c1);
   S.o.P(c2);
}
```

```
Mobile m=new Mobile();
s.o.p(m)
{
String val=m.toString();
// prints val
}
```

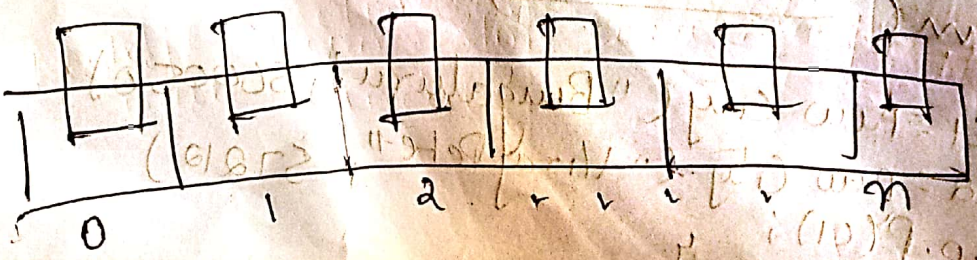
HashCode() method is defined in the Supermost class called Object

To work with any hash-based collection we need to override hashCode() method.

---

ex: ~~public~~ static

```
public static void main()
{
HashSet<String> set=new HashSet<String>();
set.add("A");
set.add("B");
set.add("C");
set.add("D");
set.add("E");
}
```



HashSet is one of the implementation class of set interface. present since JDK 1.2

HashSet implements a datastructure called HashTable

The initial capacity of HashTable is 16
And fill ration 0.75 (75%)

There are 4 overloaded constructors present in hashset

+ HashSet()
+ HashSet(int initCap)
+ HashSet(int initCap, float loadfactor)
+ HashSet(Collection col)

- HashSet does not allow duplicate data
- insertion order is not preserved.
- Objects are inserted based on Hashcode
- Heterogeneous obj allowed.
- accept single null element
- implements cloneable & Serializable
  NOT RandomAccess
- Search operation ~~becomes~~ works based ~~busy~~ on hashcode.

Linked hashset (LHS)

LHS is one of the implementation class of set interface present since JDK 1.4

LHS is just like hashset but it maintains insertion order.

Treeset

Treeset is one of the implementation class of set interface present since JDK 1.2.

Treeset is mainly used for uniqueness & sorting i.e., Treeset does not store duplicate data and also implement default natural sorting order.

default natural sorting order
- ascending order
- numb 0-9
- alphabet - a-z

Treeset is not a hashbased collection the data structure used is balanced tree.

Treeset is Homogeneous i.e., Tree Set can store only one type of data.

Treeset cannot store more Single null element

TreeSet ~~was~~ uses either Comparable or Comparator.

Note: Comparable is used for default natural sorting.

Comparator is used for custom sorting.

ex: import java.util.TreeSet;

Public class Test

{

public static void main(String[] a)

{

TreeSet<String> set = new TreeSet<String>();

Set.add("Babu");

Set.add("Suresh");

Set.add("Arun");

S.o.p(Set);

}

}

o/p: [Arun, Babu, Suresh]

VECTOR →

Vector is one of the implementation class of List interface. present since JKD 1.0 (legacy)

Vector is Single threaded i.e., methods are Synchronized.

Vector also stores the data in the form of array where initial capacity of the array is 10 and the incremental capacity is currentcapacity × 2.

Vector also has shift operation, if we try to add the data in between.

Vector maintains insertion of null and duplicate data are accepted.

Vector implements marker interfaces like randomaccess, Serializable, Clonable.

Vector has 4 Overloaded Constructors

public Vector()
public Vector(int initialCapacity)
public (int initialCapacity, int incremental Capacity)
public Vector(Collection col)

'X'

* What is the difference b/w arraylist & Vector

Arraylist	Vector
Arraylist is multithreaded	Vector is Single threaded
Arraylist methods are not Synchronized	Vector methods are Synchronized
Arraylist is present Since JKD 1.2	Vector is present Since JDK 1.0
incremental capacity is $\frac{cc \times 3}{2} + 1$	incremental capacity is cc x 2
Array list has 3 Overload constructors	Vector has 4 Over loaded Constructors.
Array list is performance wise faster	Performance wise Slower.

1 → LinkedList →

LinkedList is one of the implementation class of List interface present Since JDK 1.2

Linked List internally stores data in the form of node where in every node is connected next and its previous node.

The first node does not have previous node information and the last node does not have next node information

VECTOR →

Vector is one of the implementation class of List interface. present since JKD 1.0 (legacy)

Vector is Single threaded i.e., methods are Synchronized.

Vector also stores the data in the form of array where initial capacity of the array is 10 and the incremental capacity is current capacity × 2.

Vector also has shift operation, if we try to add the data in between.

Vector maintains insertion of null and duplicate data are accepted.

Vector implements marker interfaces like randomaccess, Serializable, Clonable.

Vector has 4 Overloaded Constructors

public Vector()

public Vector(int initialCapacity)

public (int initialCapacity, int incremental Capacity)

public Vector(Collection col)

'X'

* What is the difference b/w arraylist & Vector?

Arraylist	Vector
Arraylist is multithreaded	Vector is Single threaded
Arraylist methods are not Synchronized	Vector methods are Synchronized
Arraylist is present Since JKD 1.2	Vector is present Since JDK 1.0
incremental capacity is $\frac{cc \times 3}{2} + 1$	incremental capacity is cc x 2.
Array list has 3 Overload constructors	Vector has 4 Over loaded Constructors.
Array list is performance wise faster	performance wise Slower.

LinkedList

LinkedList is one of the implementation class of List interface present Since JDK 1.2

LinkedList internally stores data in the form of node where in every node is connected next and its previous node.

The first node does not have previous node information and the last node does not have next node information

The internal data structure is doubly linked list

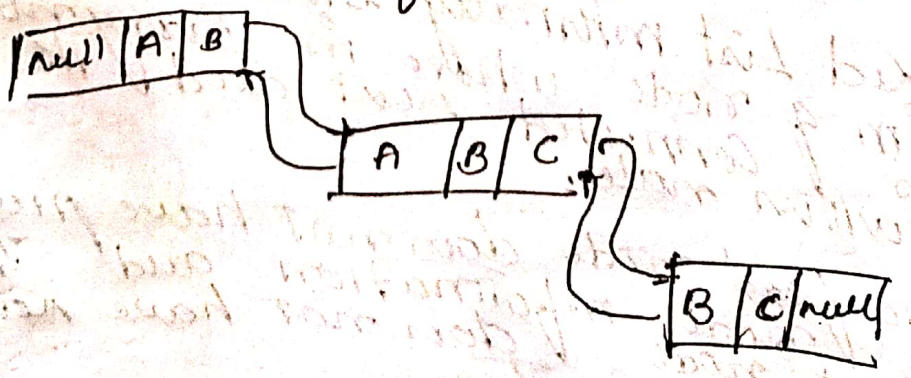
LinkedList has only 2 constructors

public LinkedList()
public LinkedList(Collection a)

LinkedList implements marker interface like Serializable & clonable but not randomAccess

- duplicates are allowed
- insertion order preserved
- heterogeneous elements allowed
- null insertion possible
- good for modification / addition / removal
- no initial capacity, no incremental capacity
- not good for search / retrieval

LinkedList list doesn't have any shift operators. Hence it is suitable for insertion & removal of data in b/w



LinkedList are not good for addition/removal because there control has to start/traverse through first node.

```
public class product {
    int price;
    double qty;
    String type;
    public product(int price, double qty,
        String name)
    {
        this.price = price;
        this.qty = qty;
        this.type = name;
    }
}
import java.util.LinkedList;
public class LLDemo {
    public static void main (_____)
    {
        product p1 = new product(1200, 1,
            "waterbottle");
        product p2 = new product(1400, 3,
            "shampoo");
        LinkedList<product> lst = new LinkedList
            <product>();
```

```
lst.add(p1);
lst.add(p2);

product pro = new product(5000, 1,
    "mobile");
lst.add(1, p10);

for (product p : lst)
    {
    s.o.p(p.qty + " " + p.type + " " +
    p.price);
    }
}
}
```

Arraylist	Linkedlist
Arraylist stores the data in the form of Array.	linkedlist stores data in the form of nodes.
data structure is resizeable/growable array	D.S is doubly linked list
Has 3 overloaded constructors	2 overloaded constructors.
initial capacity & incremental capacity is applicable	initial capacity & incremental capacity is not applicable
implements marker interfaces: Clonable, Serializable, Random Access	implements only Serializable & clonable

Arraylist has shift operations	no shift operations
good for data retrival & Search operations	LinkedList is good for insertion / removal of data in b/w list.
Memory is Continuous	memory may not be Continuous.

Iterating data from list

we can iterate data from any list using forloop

Iterating list in forward direction using for loop

```
import java.util.ArrayList
public class LD
{
p.s.v.m(--------)
{
ArrayList<String> list = new ArrayList<String>();
list.add("A");
list.add("B");
list.add("C");
list.add("D");
for(int i=0; i<=list.size(); i++) //forward
{
String chr = list.get(i);
S.o.p(chr);
}
```

```
// Reverse
for(int i = list.size()-1; i>=0; i--)
  {
String ctr = list.get(i);
 S.O.P(ctr);
 }

for (String ctr : list)
  {
  S.O.P(ctr);
} }
}
```

for each / enhanced for loop.

It is a feature introduced from JDK 1.5 it is mainly used to iterate either a collection or an array completely in forward direction.

```
LinkedList<product> plst = new
  LinkedList<product>();
plst.add(p1);
plst.add(p2);
plst.add(p3);
plst.add(p4);
```

→ references of type product

```
for(product p : plst)
  {
  S.O.P(p);
 }
```

for loop	for each loop
can Iterate forward as well as backward direction	only forward in direction..
partial Iteration is possible	partial Iteration is not possible
need to know the condition to use	can be used even without the condition
presence since begin of java	presence since JDK 1.5

Iterator <<interface>>

Iterator <<method>>

Iterator is an interface present in java.util package, it is used to Iterate any collection like Set, list and queue

Iterator is not index based.

Iterator can iterate only in forward direction

Iterator methods

public boolean hasNext()

public E next()

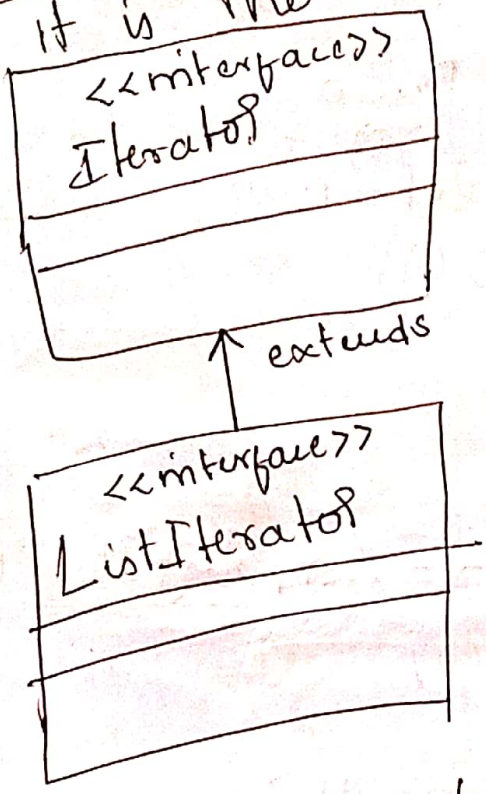
public void remove()

Iterator is also considered as a Cursor

```
import java.util.ArrayList;
import java.util.Iterator;

public class ItrDemo
{
p.s.v.m(                    )
{
ArrayList<String> lst = new ArrayList<>
        (10000);
  lst.add("Red");
  lst.add("Black");
  lst.add("yellow");
  lst.add("white");
  s.o.p(lst);

Iterator<String> itr = lst.Iterator();
  while(itr.hasNext())
  {
  String name = itr.next();
    if(name == "Black")
        itr.remove();
  }
  s.o.p(lst);
}

}
```

List Iterator

It is the Subinterface of Iterator



ListIterator can be used to iterate only list but not set & Queue.

ListIterator is indexbased hence it has indexbased methods

ListIterator can iterate both in forward as well as backward direction.

Methods of ListIterator

```
public boolean hasPrevious()
public E previous
public int nextIndex()
public int previousIndex()
```

ListIterator litr = lst.listIterator();

- AL
- LL
- Vector

Example:

```
import java.util.LinkedList;
Public class ListItrDemo
{
P.S.V.M (_____)
{
LinkedList<String> lst = new
LinkedList<String>();
lst.add("A");
lst.add("B");
lst.add("C");
ListIterator<String> litr = lst.listIterator();
while(litr.hasNext())
{
String fruit = litr.next();
Sop(fruit);
}
}
}
```

Difference b/w Iterator & ListIterator

Iterator	List Iterator
• It is not index based	• ListIterator is index based.
• It can iterate only in forward direction	• Can iterate both forward as well as backward direction
• Iterator is the Super Interface	• It is Sub interface
• Iterator can be used with any collection like list, Set & Queue	• Can be used only with List.

```
eg: public class city
    {
    int picode;
    String name;
public city (int picode, String name)
    {
    this.picode = picode;
    this.name = name;
    }
}
```

```
import java.util.ArrayList;
import java.util.ListIterator;

public class ListDemo
{
public static void main (String[] args)
{
city c1 = new city (561020, "Tumkur");
city c2 = new city (421410, "Vidoic");
city c3 = new city (560050, "Bangaloic");

ArrayList<city> lst = new ArrayList<city>();
lst.add(c1);
lst.add(c2);
lst.add(c3);

ListIterator<city> litr = lst.listIterator();
while(litr.hasNext())
{
city cty = litr.next();
S.o.println(city.name + "  " +
cty.pincode);
}
```

MAP →

Entry:
is a data in an asociated form
i.e., Key, value pair
Key must always be unique,
but value can be duplicated
& null.

Map is an collection of Entries
i.e., map store multiple entries.

programatically map is an interface
present since JDK 1.2

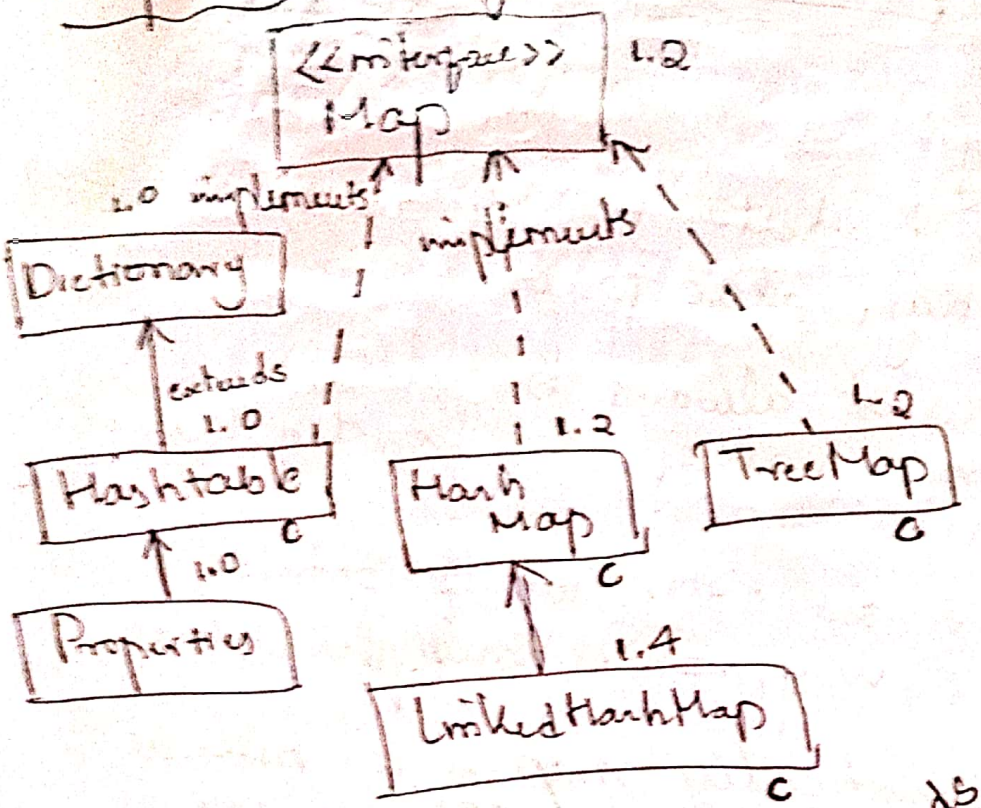
Map is the part of collection frame
work present in java.util package.

Map is a Seperate Vertical which
doesn't inherit collection interface.

Map<K, V>
Map<Integer, String>

8 = "Rohit"	9 = "Dohni"	10 = "Modi"

Map Hierarchy



Map Methods

9 methods

```
public void clear()
public boolean containsKey(Object key)
public boolean containsValue(Object key)
public V get(Object key)
public boolean isEmpty()
public V put(K key, V value)
public void putAll(Map m)
public V remove(Object key)
public int size()
```

```
public Set<K> keySet()
public Collection<V> values()
public Set<Map.Entry<K,V>> entrySet()
```

```
public interface Map
{
    public interface Entry
    {
        public K getKey();
        public V getValue();
    }
}
```

```
import java.util.HashMap;
import java.util.Map;
public class MapMethodsDemo
{
    P.S.V.M(_____)
    {
        Map<Integer, String> midMap = new
            HashMap<Integer, String>();
        midMap.put(17, "Dohni");
        midMap.put(8, "Virat");
        midMap.put(10, "Rohit");

        Map<Integer, String> ausMap = new HashMap
            <Integer, String>();
        ausMap.put(1, "Smith");
        ausMap.put(22, "Warner");
        ausMap(2, "Maxwell");
```

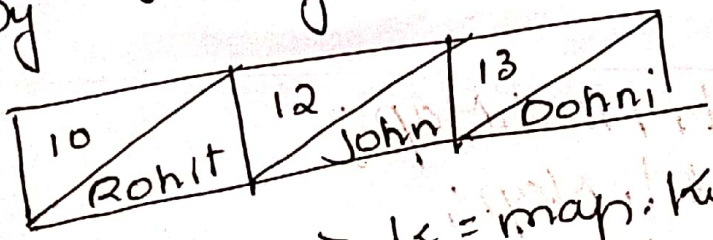
```
Map<Integer, String> iplMap = new
    HashMap<Integer, String>();
iplMap.putAll(midMap);
iplMap.putAll(ausMap);
S.o.p(iplMap.size()); //6
S.o.p(iplMap.isEmpty());
iplMap.remove(22);
S.o.p(iplMap.containsKey(45));
S.o.p(iplMap.containsValue("Rohit"));
S.o.p(iplMap.get(8));
S.o.p(iplMap.size()); //5
}
}
```

```
import java.util.HashMap;
import java.util.Map;
public class MapReplaceDemo
{
p.s.v.m(------)
{
  Map<Integer, String> midMap =
new HashMap<Integer, String>();
midMap.put(17, "Dohni");
midMap.put(8, "Virat");
S.O.P(midMap);
```

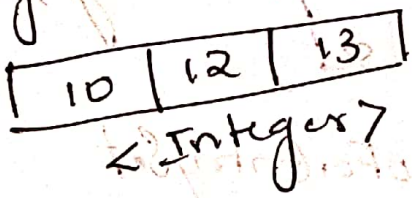
```
midMap.put(8,"Rohit"); // replace
    S.O.P(midMap);
  }
}
```

o/p 17=Dohni , 8=Virat
17=Dohni , 8=Rohit

Directly we cannot iterate a map by using iterator, for, foreach loop.




```
Set<Integer> ks = map.keySet();
```



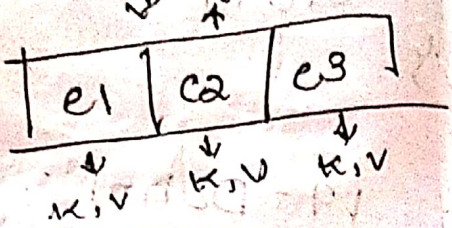

```
Collection<String> vals = map.values();
```

Datatype of Key & value can be anything.

Set<Entry> set = map.entrySet()

↓

Set<Entry<K, V>>



↓

Set<Entry<Integer, String>> set = map.entrySet();

↓

foreach iterator.

```
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry; //imp
import java.util.Set;
public class DemoForEntrySet
{
  p.s.v.m(c ————)
{
Map<Integer, String> mdMap =
    new HashMap<Integer, String>();
  mdMap.put(17, "Dhoni");
  mdMap.put(18, "Virat");
  mdMap.put(10, "Rohit");
```

```
Set<Entry<Integer, String>> set = rndMap.
    entrySet();
for(Entry<Integer, String> entry : set)
{
  Integer key = entry.getKey();
  String val = entry.getValue();
   S.O.P(key + " = " + val);
}
}
}
```

HASHMAP

- HashMap is a mapbased collection which is used to store entries. and present since JDK 1.2
- Hashmap hashbased, hence the internal data structure is hashtable
- Initial Capacity of hashmap is 16 & fill ration or 0.75(75%)
- hashmap can store single null key
- hashmap can store heterogeneous data & does not maintain any insertion order.

- It has 4 Overloaded Constructors.

```
public HashMap()
public HashMap(int initCapacity)
public HashMap(int initCapacity,
  float loadFactor)
public HashMap(Map anotherMap)
```

```
import java.util.*;
public class HashMapDemo
{
p.s.v.m(————————)
{
Map<Integer, String> hmap=
  new HashMap<Integer, String>();
// map<Integer, String> hmap=
  new HashMap<Integer, String>
  (1000);
// map<Integer, String> hmap=
  new HashMap<Integer, String>
    (1000, 0.95f);
    hmap.put(50, "Vijay");
  hmap.put(60, "Shree");
  hmap.put(70, "Dolmi");
  hmap.put(null, "Virat");
  S.O.P(hmap);
LinkedHashMap<Integer, String>
  hmap=new LinkedHashMap<Integer,
    String>(hmap);
```

```
        S.O.P(tmap);
    }
}
```

LinkedHashMap	
• LHM is one of the Subclass of map interface. present Since JDK 1.4 • LHM maintains insertion order • all the other features almost as Hashmap	

Hashtable

it is one of the implementation class of map interface. present Since JDK 1.0. it is also a hashbased map. and the data structure is Hash-table.

Hashtable is Single threaded.
i.e. methods are Synchronized.

Hashtable initial capacity is 11, fillration as 0.75 (75%)

Hashtable cannot even store a single null key.

Hashtable is slower than HashMap ∵ it is single threaded.

There are 4 overloaded constructors present in Hashtable just as like HashMap.

* HashMap	Hashtable
HashMap is multithreaded	it is single threaded
present since JDK 1.2	present since JDK 1.0
methods are not synchronized	methods are synchronized
can store single null key	cannot even store a single null key.
initial capacity is 16	initial capacity is 11
it is faster	it is slower.

→ TreeMap] it cannot we store a single null key.

TreeMap is one of the implementation class of Map interface present since JDK 1.2

TreeMap is mainly used for Sorting data based on Key.

TreeMap implements default natural Sorting order on the Key using Comparable

for Custom Sorting we use Comparator.

```
import java.util.*;
public class TreeMapDemo
{
p.s.v.m (_____)
{
TreeMap<Integer, String> tmap = new TreMap
<Integer, String>();
tmap.put(50, "Vijay");
tmap.put(20, "dilip");
tmap.put(25, "Gulabu");
s.o.p(tmap);
```

o/p ⇒ 20=dilip 25=Gulabu 50=Vijay.

- Treemap is homogeneous [Key]
- Null pointer Exception, when we try to add null

Comparable & Comparator

They are functional interfaces.

Comparable has an abstract method

- public int compareTo(T ele) ← Type of data
- public int compare(T ele1, ele2)

Comparable is present java.lang package

Comparator is present in java.util package.

both Comparable & Comparator is used to compare the data

If 2 data are same the methods must Return zero (0)

If first data is greater the Second data the methods must return +1

If first data is Smaller than Second data, then methods must return -1

String class and other wrapper classes have implemented Comparable and Overridden compareTo() method

ex:

```
public class Demo
{
P.S.V.M(------->)
{
 int i = 28; } primitive type
 int j = 35;
 Integer wi = i;
 Integer wj = j;
S.O.P(wi.compareTo(wj)); // -1

 Double d1 = 545.8;
 Double d2 = 5718;
S.O.P(d1.compareTo(d2)); // +1

 Long l1 = 98867321604;
 Long l2 = 98867321604;
S.O.P(l1.compareTo(l2)); // 0
 }
}
```

o/p
-1
+1
0

* IX

where Comparable & Comparator is used / useful ?

Both Comparable & Comparator is useful while sorting data in case of collection.

List can be sorted using Collections.sort(List)

Set can be sorted using TreeSet

Map can be sorted using TreeMap

Sorting a List ⟷

A List (ArrayList, LinkedList, Vector) can be sorted by using an utility method called sort(), present in java.util.Collections class.

Collections is an utility class which has some utility methods. (automatically use Comp

public static void sort(List lst)

public static void sort(List lst, Comparator ctr)

example:

```
import java.util.ArrayList;
import java.util.Collections;

public class AlSortDemo
{
P.S.V.M( ------------ )
{
ArrayList<String> lst = new ArrayList<String>
();
lst.add("A");
lst.add("x");
lst.add("D");
lst.add("4");
S.O.P(lst);
Collections.sort(lst);
S.O.P(lst);
Collections.sort(lst, Collections.reverseOrder());
S.O.P(lst);
```

o/p:

[A, x, D, 4]

[A, D, x, 4]

[4, x, D, A]

*) what is the difference b/w Collection & Collections.

Collection is an Interface

Collections is a class

Collection	Collections
it is the Super interface Collection Hierarchy	Collections is an utility class which has utility methods

Collection.Sort(["B" | "A" | "Z" | ... |])

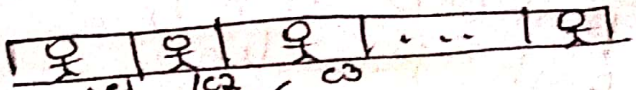
String Element's class

- compare the data → Comparable compareTo
- sort / Arrange data

}

Sort method internally calls CompareTo methods

Note: for sorting a list we have to make sure the compareTo() method is present in element (element's class)

Collections.sort(

);

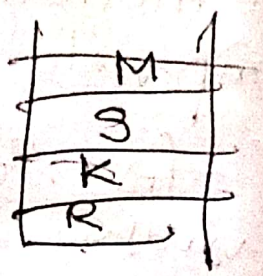
Employee
↳ it should implement Comparable & override compareTo()

(int) e1.compareTo(e2);
Sort based on comparision result.

}

1 -

Stack : FILO




```
import java.util.*;
public class StackDemo
{
P.S.V.M(_____)
{
StackDemo * stk = new StackDemo();
 stk.push("Ramesh");
 stk.push("kamalsh");
 stk.push("Suresh");
stk.push("Mahesh");
  S.o.p(stk);
  S.o.p(stk.size()); //4
  S.o.p(stk.peek()); // mahesh
  S.o.p(stk.peek()); // mahesh
  S.o.p(stk.size()); //4
  S.o.p(stk.pop()); // mahesh
  S.o.p(stk.pop()); // Suresh
  S.o.p(stk.size()); //2
  S.o.p(stk); [Ramesh, ~~Suresh~~ kamalsh]
  S.o.p(stk.Search("Ramesh"); //4
  S.o.p(stk.Search("Suresh"); //2

//Search method tells the position from top
 } S.o.p(stk.Search("beat"); // -1

}
```

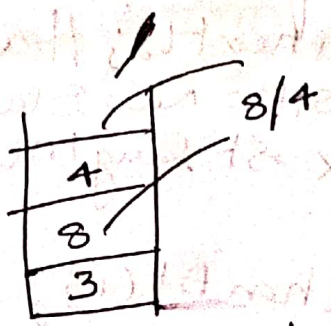
Application

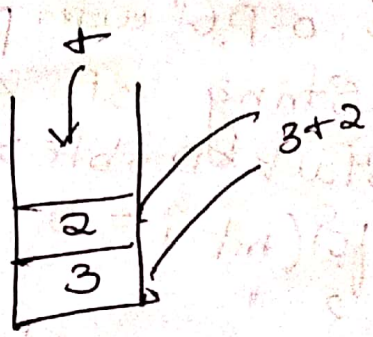
- use in methods of functions invocation (Recursively called method will get finished first)
- Arithmatic Expression Evaluation
  - infix
  - prefix
  - postfix

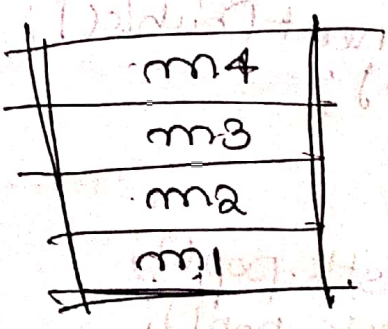
  } expression conversion.

3+8/4

384/+







Exception handling

Exception is a runtime interruption which stops the program execution

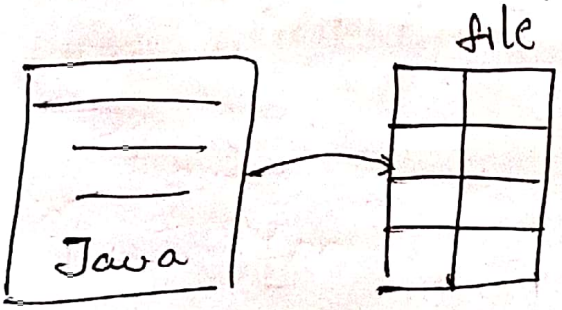
Exception can be suppressed or handled so that the program can continue its execution.

There are many inbuild exception available in Java. Apart from inbuilt exception it is also possible to create our own exceptions, which are called as custom or user defined exceptions.

Exception is broadly classified into 2 types.

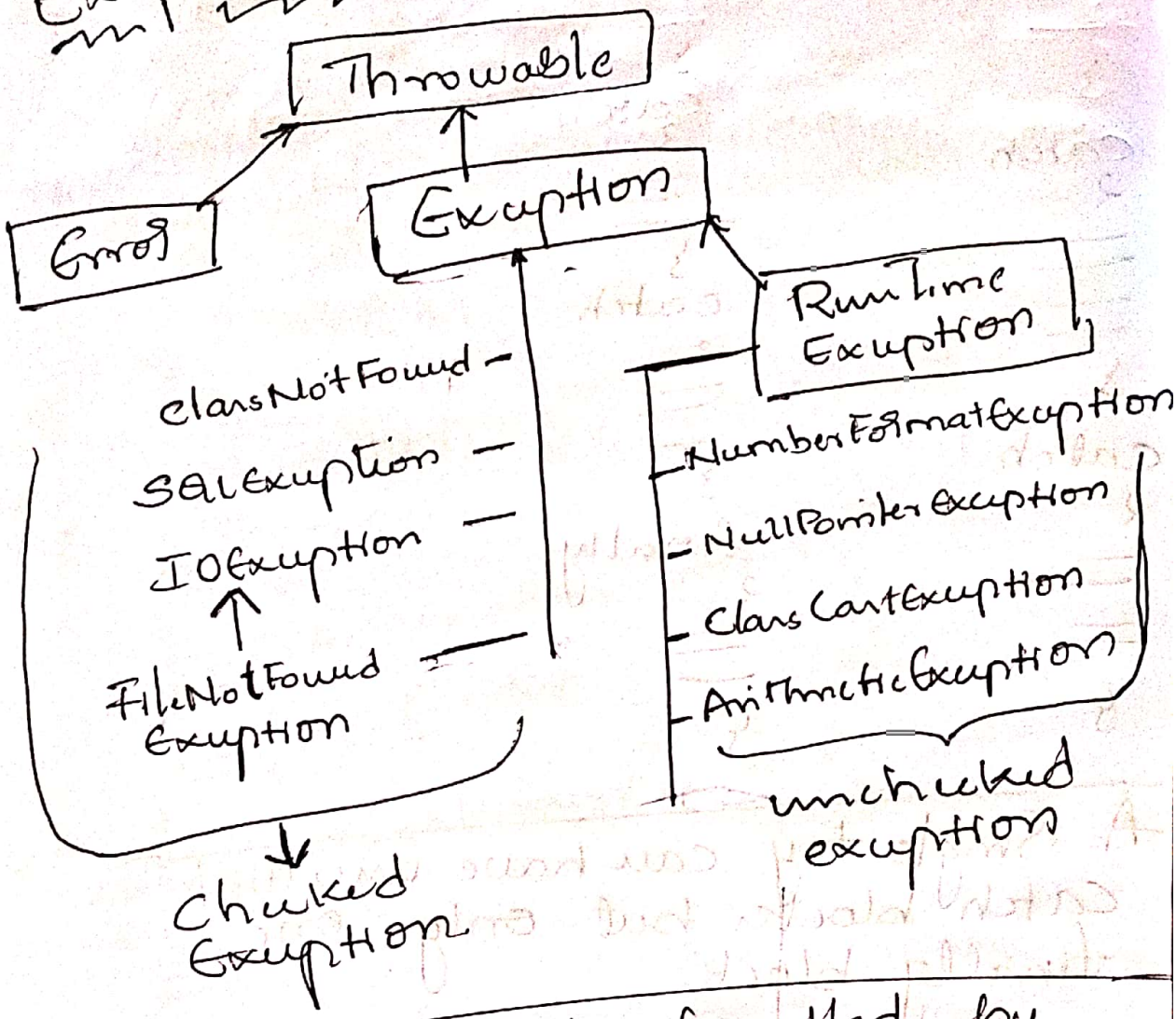
→ checked exception

→ unchecked exception



FileNotFound exception

Exception Hierarchy



An exception can be handled by using try, catch and finally blocks

```
try
{
==
}
catch(Exception e)
{
==
==
==
}
```

```
try
{
}
catch
{
}
catch
{
}
```

```
try
{
}
catch
{
}
catch
{
}
finally
{
}
```

- A Single try can have multiple catch blocks but only one finally block.
- try, catch & finally block must always be associated together. There must not be any executable code b/w them

* In what situation we have to write multiple catch blocks?

If handling Scenarios are different, Then write different catch block for different exception

ex: Void method()
{
try
{
≡ ArithmaticException
≡ SQLException
}
catch(ArithmaticException ae)
{
≡
}
catch (SQLException sql)
{
≡
}
}

* In what situation we write only one catch block?

A) If Scenario is same for multiple exception then we write only one catch block

```
try
{
    ArithmaticException
    SQLException
}
catch (Exception e)
{
    // Show error msg
}
```

```
try
{
}
catch (ArithmaticException | SQLException e)
{
    // Show error msg
}
```

Catch block gets executed only if there are any exception in the try.

At any given point of time only one exception can occur in the try block (multiple exception cannot occur at a same time)

Once an exception occur in the try block then the control immediately comes out of try block without executing the remaining lines of code within try block.

When control comes out of try block then if the matching catch block is found it gets executed.

If the matching catch block is not found then program gets terminated

At any given point of time for a single exception only one catch block gets executed

Once control comes out of try block then it will not go back to try block again.

Catch block gets executed only if there are some error in try block

Seaunce of catch blocks

The Seaunce of multiple catch block must always be from from Subclass to Superclass, other wise compilation error.

when we write multiple catch block, and if there is an IS-A Relationship b/w multiple catch block then the Seaunce must be Subclass to Superclass

Sub
↓
Super

ex:
```
try
{
    ≡ Arithmatic
}                Exception
catch(Exception e)
{
}
catch(ArithmaticException ae)
{
}
```

↳ unreachable catch block

```
try
{
  ========
  ========
}
catch(IOException ioe)
{
  ========
}
catch(FileNotFoundException fe)
{
  ========
}
```

↓ unreachable catch block

↓ compilation error.

throws :

Throws is a keyword which is used with method declaration it is used to indicate the possibility of exception from a method.

Throws keyword don-not throw an exception rather it only indicates the possibility of exception.

- using throws we can indicate multiple exceptions.
- when we call a method which an throws declaration, then we have to handle the code by using try & catch block

ex: public static int parseInt(String input)
throws NumberFormatException

{

}

```
public class Demo
{
p.s.v.m(_____)
{
    try
    {
    int i = Integer.parseInt("hello");
    s.o.p(i);
    }
    catch(NumberFormatException nfe)
    {
    }
}
}
```

Throw:

Throw is a keyword which is used to throw an exception object.

using Throw keyword we can throw only one exception object

Throw can be used within a method or constructor.

Difference b/w Throw & Throws

Throws	Throw
It is used to indicate the possibility of exception	Throw is used to actually throw an exception object
~~Throw is used to actually~~	
Throws deals with method of associated with methods	Throw deals with of associated with exception object
Throws is used with method declaration	Throw is used inside the method

Note:
Throws can also be used with constructor declaration.

Checked Exception

Checked Exceptions are those exceptions for which the Compiler checks whether the exception is handled of not.

If the exception is handled by using try, catch block or by using throws then compiler will not give any error. if not, compiler give error. and force to handle the exception.

Once the checking happens by compiler it is called as checked exceptions.

ex: import java.io.FileReader;
public class CheckedDemo
{
public static void main (——)
{
FileReader fr = new FileReader("data.txt");
}
}

// compilation error
// possible exception is not handled.

ex: DriverManger.getConnection(url)
error;
// checked-exception is not handled

---

try
{
DriverManger.getConnection(url);
}
catch (SQLException e)
{
// no error
}

Note:
programatically class which belongs to checked exception does not inherit runtime exception.

Unchecked Exception:
are those exception for which the compiler does not check whether the exception is really handled or not programatically ~~runtime exception extends~~
Unchecked Exception class extends RuntimeException.

Custom or user defined Exception
There are 2 important methods which are useful for debugging the exception

public void printStackTrace()
public String getMessage()

both the methods are defined in Superclass class called Throwable

Hence it is inherited to all the classes which inherit or extends Throwable.

---

when an inbuilt exceptions are not enough then its possible to write & define our own custom & user defined exceptions.

writing a custom exception is a 2 step process.

1. write a class which extends either Throwable or exception or RuntimeException

2. Override toString() method and getMessage() method

ex: public class InvalidTransaction extends Throwable) checked
Exception)
RuntimeException (unchecked)
{
private string msg = "invalid transaction";
public string toString()
{
return msg;
}
public string getMessage()
{
return msg;
}
}

* what is Exception, and how do we handle it?
* diff b/w exception & error?
* Super class of Exception?
  A: Throwable.
* diff b/w throw & throws
* Explain checked & unchecked Exception with some example.
* what is finally block.

finally :

It is a block which is used in exception handling

finally block always gets executed irrespective of whether exception occurs or not

a single try block can have maximum of one finally block.

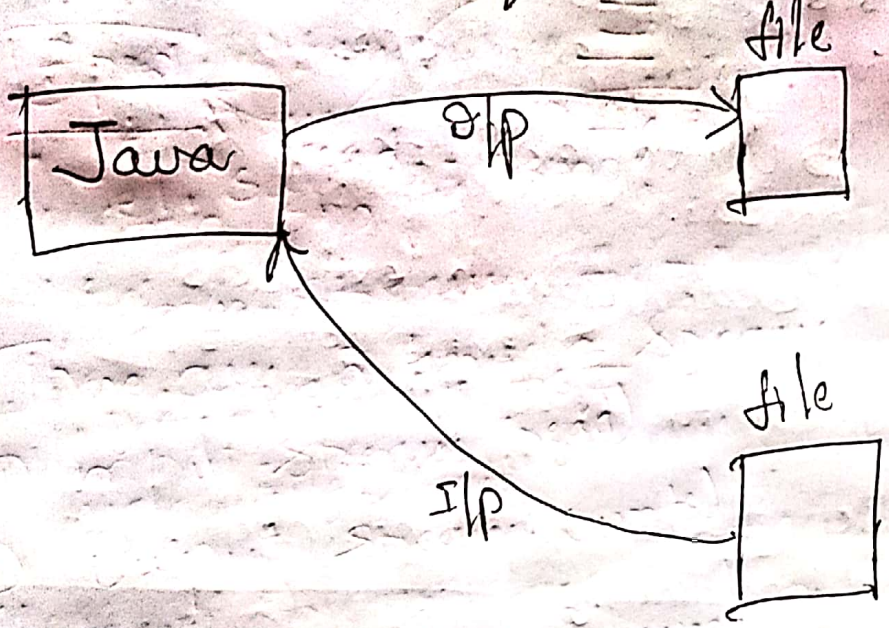
usually costly resources are closed in finally block. (DB connection, IO streams)

Note:

We can write try, catch & finally block within another try block or another catch block or even within finally block which is called as nested try, catch block.

IO Serialization Filehandling

In Java we have an inbuilt package java.io, which has many classes and interfaces which is useful for IO operation.



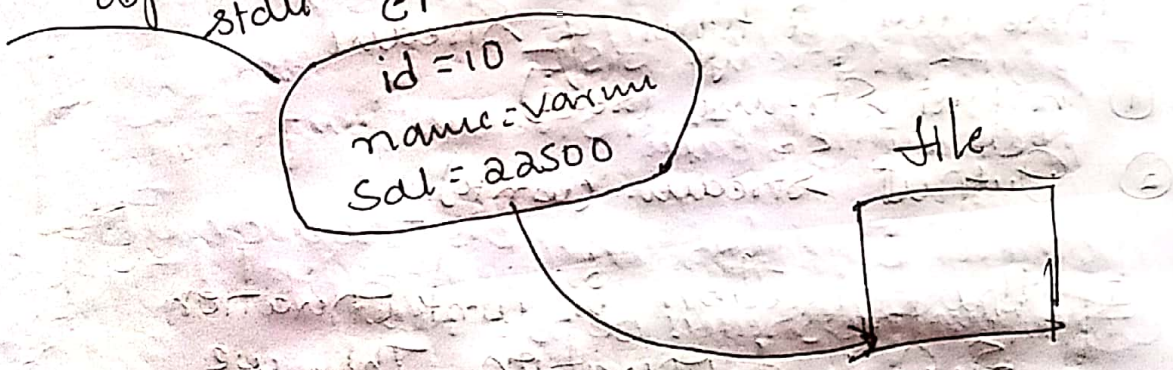
Serialization

It is the process of converting Object's state along with class information into byte stream

To Serialize an Object the class must implement a marker interface named Serializable.

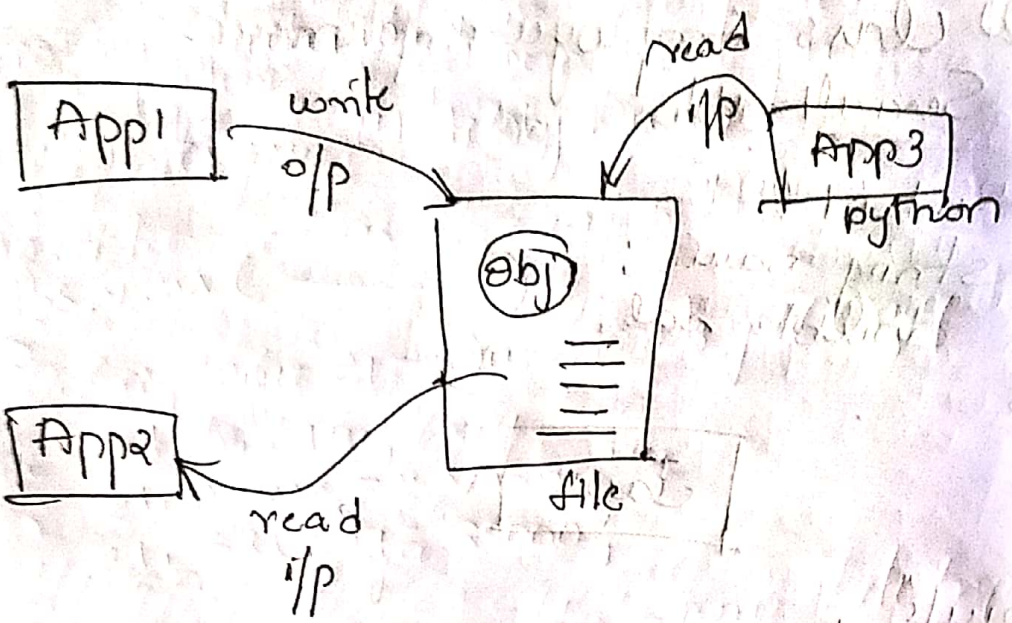
class Employee implements Serializable
{
int id;
String name;
double sal;
}

Java



The reverse mechanism of Serialization is called Deserialization. i.e., Converting byte stream into Object state

We can Save the data in a file either in an Object format or in a Simple text format



There are 2 types of Stream

① input Stream (read)

② Output stream (write)

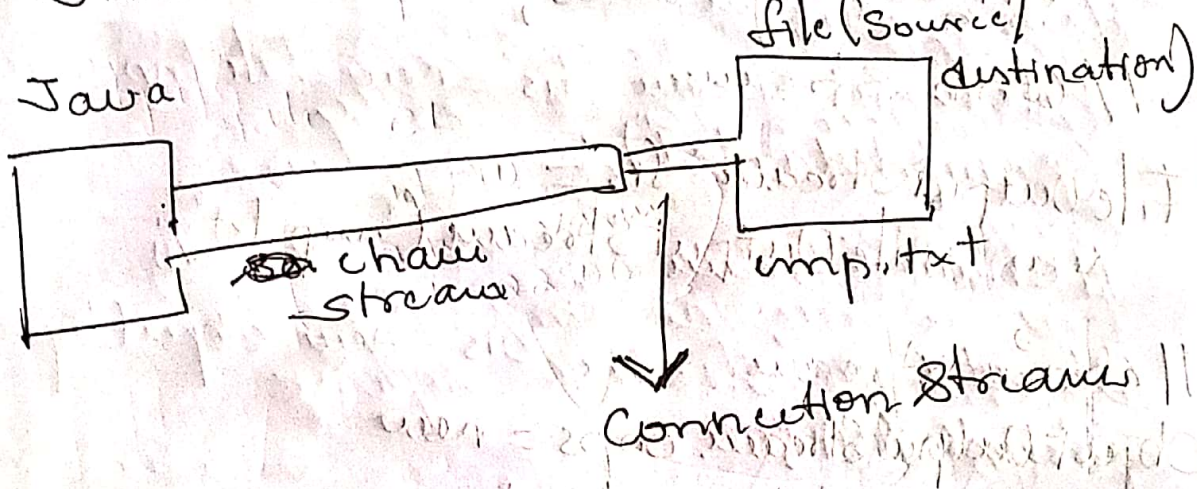
If the operation is write operation or Output operation Then we use output stream

If the operation is read operation or input operation Then we use input stream

both input stream & Output Stream can be further divided into Chain Stream & Connection stream.

A Connection Stream is a Stream which is directly connected to the source or the destination.

A chain Stream is not directly connected to source/destination rather it connects to connection Stream.



ex:

```
import java.io.Serializable
public class Employee implements
    Serializable
{
    public int id;
    public String name;
    public double salary;
}
```

```
import java.io.*;
public class WriteObjDemo
{
P.S.V.M(------------)
{ Employee e1=new Employee();
  e1.id=10;
  e1.name="Varun";
  e1.salary=25000;
 // Conn Stream
FileOutputStream fis=
  new FileOutputStream("emp.txt");
// chain stream
ObjectOutputStream oops=new
  ObjectOutputStream(fis);
oops.writeObject(e1);
oops.close(); // fis.close();
S.O.P("write proms Success");
 }
}
```

transient:
transient variable cannot be Serialized

Note: Static Variable also cannot be Serialized.

Note: If class is not implements Serializable then at runtime JVM throws NotSerializableException,

---

Example for Deserialization

```
import java.io.*;
```

Deserialization is the process of converting the byte stream into Java object.

i.e., Deserialization is the reverse process of Serialization.

At the time of deserialization there is a possibility of IOException, FileNotFoundException, ClassNotFoundException

Deserialization is one of the ways of object creation in java.

```
import java.io.*;
public class ReadObjDemo
{
P.S.V.M(——) throws IOException,
ClassNotFoundException
{
// Conn Stream
FileInputStream fis = new FileInput
("empdata.txt");

// Chain stream
ObjectInputStream ois = new ObjectIn
Stream(fis);

Object o = ois.readObject();
Employee emp = (Employee) o;
Sys.o.p(emp.id + " " + emp.name +
" " + emp.~~sal~~ salary);
}
}
```

```
import java.io.*;
public class ReadObjDemo
}
p.s.v.m(------)
{
s.o.p("main starts");
FileInputStream fis=null;
FileInputStream
ObjectInputStream ois=null;
try
{
fis=new FileInputStream("empdata.txt");
ois=new ObjectInputStream(fis);
Object o=ois.readObject();
Employee emp=(Employee)o;
s.o.p(emp.id+" "+emp.name+"\n"+
  emp.Salary);
}
catch(ClassNotFoundException e)
{
  e.printStackTrace();
}
catch(IOException e)
{
e.printStackTrace();
}
finally {
  if(fis!=null && ois!=null)
    try {
      fis.close();
      ois.close();
      }
```

```
catch(IOException e)
{
  e.printStackTrace();
}
}
}
s.o.p("main end");
}
```

Multiple IO operations are considered to be costly operation hence we use buffers.

Buffers are the temporary stream which are used to improve performance.

for output operation we use BufferedWriter

for input operation we use BufferedReader.

* difference b/w flush() & write() in case of Buffers:

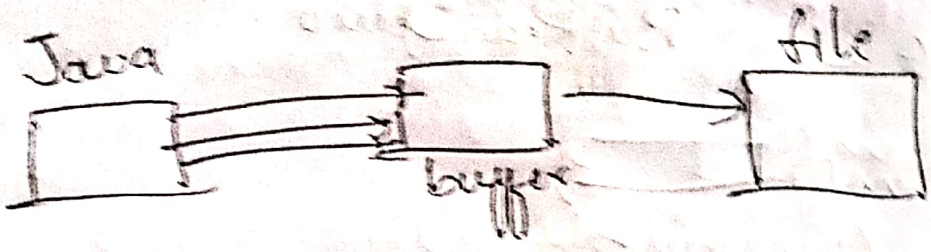
write() method writes the contents of a buffer into a file only when the buffer is full.

flush() method writes the contents of a buffer to a file even if the buffer is not full

```
import java.io.*;
public class BufReadDemo
{
P.s.v.m( ——— )
{
 try
  {
  FileReader fr=new FileReader("bahubali.txt");
BufferReader br=new BufferReader(fr);
   String content=null;
while((content=br.readLine())!=null)
 {
  s.o.p(content);
  }
}
catch(IOExcuption e)
  {
   e.printStackTrace();
  }
}
}
```

bahubali.txt

———
———
———
———
———



Serial Version UID

It's a long value & unique for every class

Serial Version unique id is useful in identification of class at the time deserialization

jdk\bin\serialver.exe is a developmental tool, which is used to create Serial Version unique ID