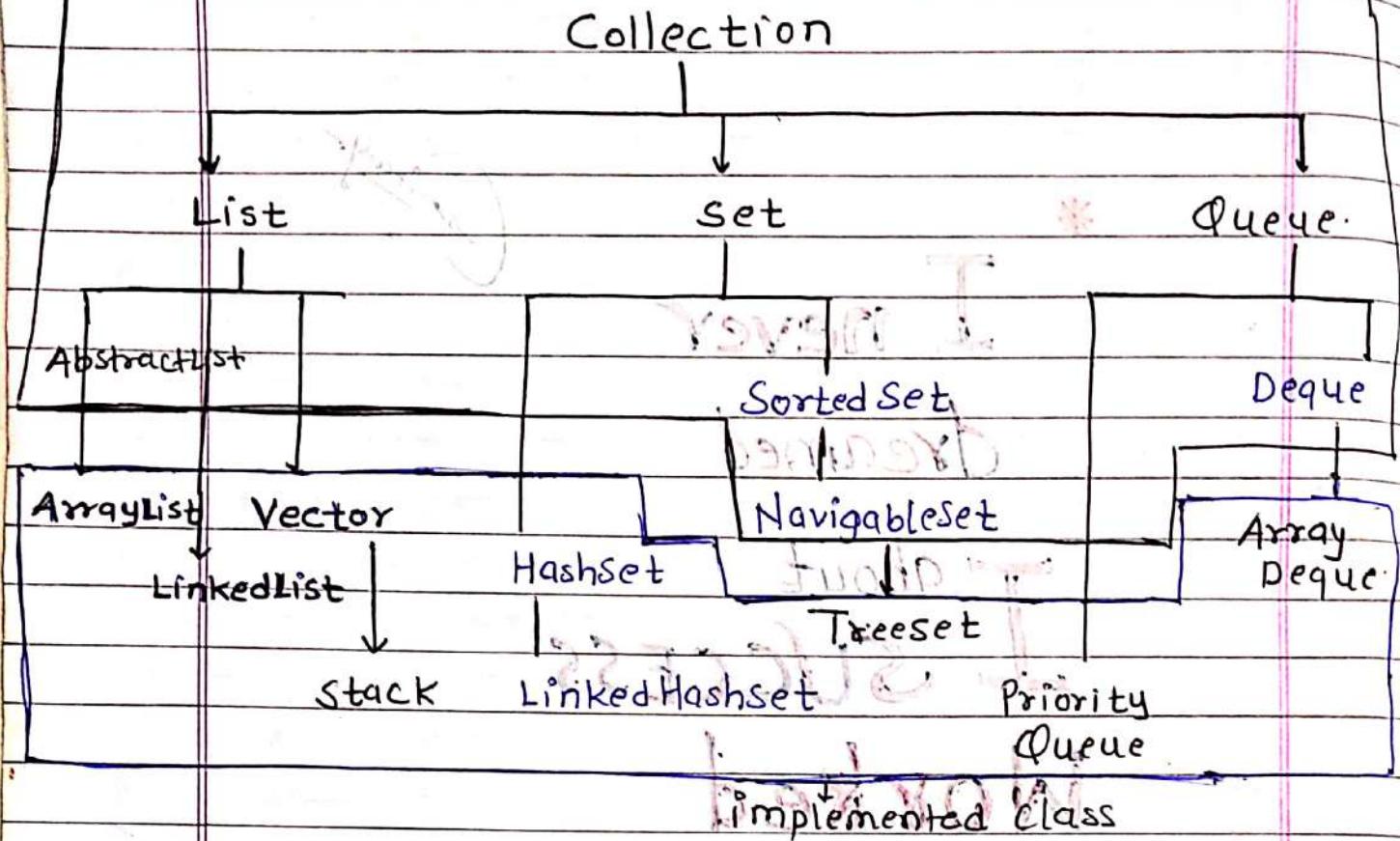


I never
dreamed
about
I SUCCESS
Worked
for it.

26/07/21

Briam

Interface



~~• If not method~~

~~of method is factory~~

~~↳~~ `List.of()`

~~↳~~ `Set.of()`

~~↳~~ `Map.of()`

~~↳~~ `List.of(v1, v2, v3)`

~~↳~~ `Set.of(x1, x2, x3)`

~~↳~~ `Map.of(k1, v1, k2, v2, k3, v3)`

~~↳~~ `Set.of(x1, x2, x3)`

~~↳~~ `Map.of(k1, v1, k2, v2, k3, v3)`

~~↳~~ `ImmutableSet.of(x1, x2, x3)`

~~↳~~ `Object.freeze(new Map())`

~~↳~~ `Create Keypair`

~~↳~~ `Object.create(null)`

Collection Framework

→ Types of Data structures: 1. Primitive Data structure.

e.g.: int, char, long, float, double, boolean, byte.

Sequence: boolean, char, byte, short, int, long, float, double.

2. Non-Primitive Data structure.

→ String, Files, Arrays, Collection

3. Java Collection Framework (Collection, map)

→ Application of Arrays:-

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

2. Arrays are used to develop some algorithms like Bubble sort, Insertion sort, Selection sort etc → ex-set^{rg}.

3. Arrays can be used to perform matrix operations.

4. Arrays can be used to perform matrix operations.

5. Arrays can be used for CPU scheduling.

6. Arrays are used to implement data structures for example stack, ArrayList, Queues etc

⇒ Points to remember for Array :-

Collection Framework :-

→ Array is Java language feature

inbuilt support provided by sun Microsystems.



Array basically Java ka

inbuilt feature hota hai jo Java

new allocation banaya hua hai ⇒ we

have to develop algorithms to sort or insert or delete etc

⇒ Collections are API features



collection jo hota hai API features

hoti hai API ka kya matlab hai

enho ne predefined kuch set, bandya

hui bhai kisi chij kei class and Interface

ke or un classes and Interfaces ke.

Andher classes and Interfaces bahut

sare method hain jinko directly use

nahi karne ~~padta~~ hota hai in them

Agar hum logo collection framework

use kai rukhai na to hum khudh

se koi logic path lagane ki need

nhi hui

⇒ It provides predefined classes

and interfaces and methods by which

we can easily iterate or delete or

sort the elements.

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

↑
array bhi hai Jo objects

Store Karwa skta hai

→ Collection Framework can store only non-primitive data type (objects)

~~Wrapper class~~ C Integer i = new Integer(2011no1);
object hai

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

→ Collection Framework can store heterogeneous data i.e. we can store different type of data.

Alphabets sent hua
class HomeLoan

Object inheritance
class CarLoan

hL1 CL1 hL2

BankApp

ye store kyu ki
data types
aggia hui

3

hL1 CL1

keyu ki
data types
store hogya
mujhe

HomeLoan hL1 = new HomeLoan();

CarLoan CL1 = new CarLoan();

ye store
kya hoga?

Collection with example
(ArrayList) containing more than 100 elements.

hL1 CL1 hL2

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

→ The size of collection can be increased or decreased according to our needs.

⑤ → Array are not good with respect to memory.

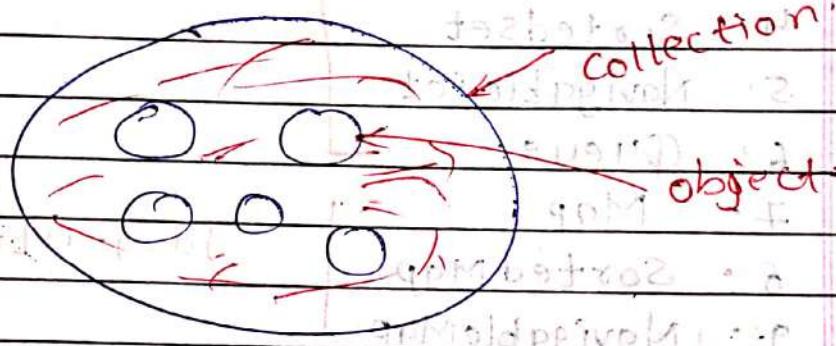
→ Collection framework are very good with respect to memory.

⑥ → Arrays are good by performance wise.

→ Collection are not good by performance wise.

* What is Collection Framework?

Ans: Collection Framework consists of 2 words
1-st Collection and Framework.



= Collection → collection woh ek single entity, ya phir single objects ko represent karta hai jiske under multiple data stored ho skta hain.

→ Definition: Collection is an single entity or an object which contains multiple data.

→ Framework → represents the library

→ collection framework is the set of classes and interfaces that implement commonly reusable collection data structure.

→ Collection framework contains 2 main parts:-

- ✓ 1. Java.util.collection
- ✓ 2. Java.util.map

m&

→ "9 key interfaces" of collection framework.

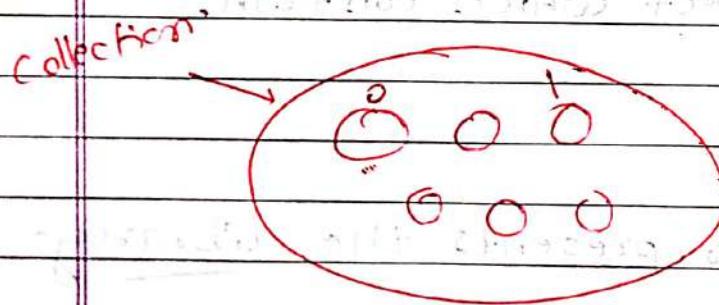
1. Collection
2. List
3. Set
4. SortedSet
5. NavigableSet
6. Queue
7. Map
8. SortedMap
9. NavigableMap

Java.util.mapcollection

Java.util.Map

* Hierarchy of Collection Framework:-

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



map

Key	Value
101	deepak
102	amit
103	rahul

(Root Interface
is collection.)

Collection

Date _____
Page _____

Collection: - interface (non-final, non-static)

→ collection is an interface which is present in Java

java.util package.

→ **syntax:** public interface Collection<E> extends
Iterable<E>

→ collection was introduced in JDK 1.2 version.

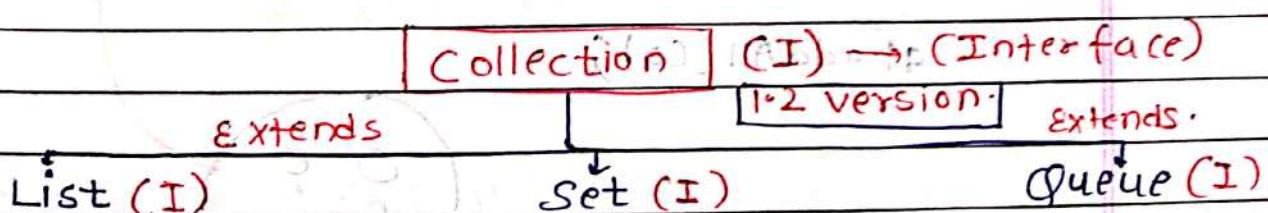
→ Collection is an object which is used to represent a group of individual objects as a single unit.

→ Collection interface is the root interface of collection Framework.

→ There is no concrete class which implements the collection interface directly but there are interfaces which inherit the collection interface.

↳ List, Set & Queue

→ Hierarchy of collection interface:



Collection Interface ka object

create nhi kar skata

update
Page

→ Collection Interface Jo hui banaya kyu hai?

DENE ki shayar ki diler mohor ki apni mitoosha (10)

Jitne bhi common hei methods hui (kis cheez ke liye) hai ek collection interface ya phir classes ke liye collection framework

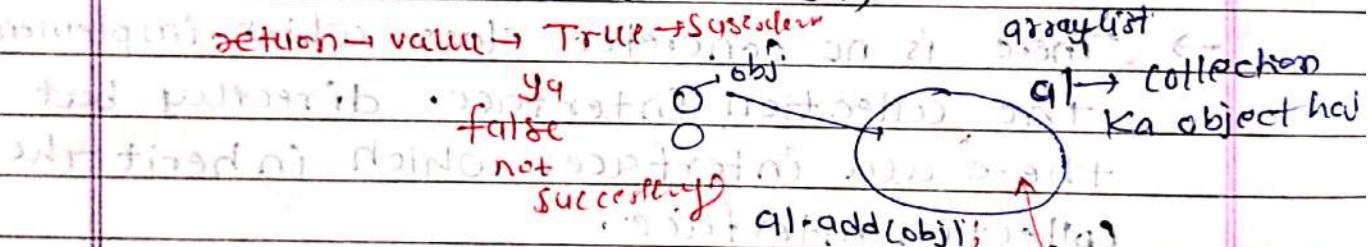
Interface and class ke liye wah sare ke sare common method hai collection ke

Adher banaye gye hai

→ Collection interface contains most common methods which are applicable for any collection object.

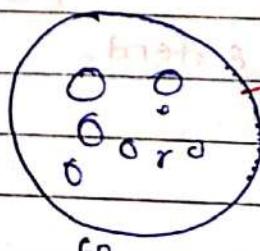
→ Methods of collection Interface :-

1. boolean add (Object obj);



2. boolean addAll (Collection c);

q1.addAll(c);



Agar App log ek collection ke object ko dusre collection me store karna chahte ho

to addAll(Collection)

method ko use kروں

✓ 3. boolean remove (Object obj)

al.remove(xyz);

collection object me send hoga
remove koojan ka kam
karta hai

collection
object

xyz

al

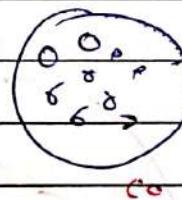
✓ 4. boolean removeAll (Collection c);

al.removeAll(c);

collection
object

c

((list of objects) and load)



✓ 5. default boolean removeIf (-) § - 3

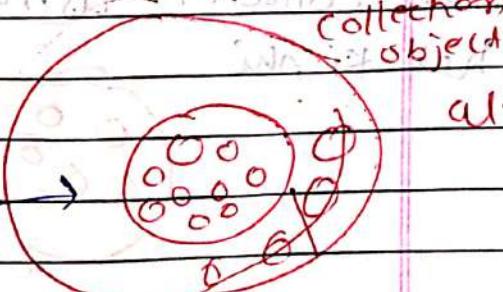
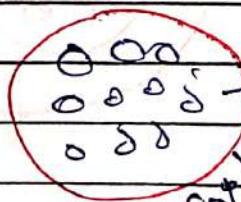
Agar ~~del~~ kisi bhi method ek Age default

laga de to uski body ~~benzene~~ hui

or uski Interface eke undhe create
kar skata hai

((a removeIf) interface) amlaat

✓ 6. boolean retainAll (Collection c);



co → ein object

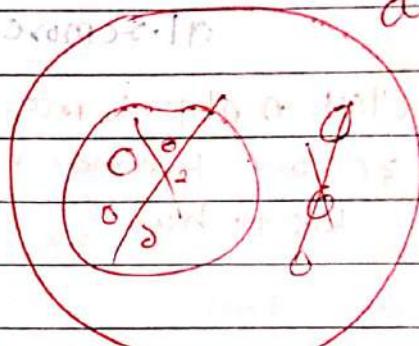
ko shod ke baki sabko remove kar dega

Collection is global
nature ka hota hai

Date _____
Page _____

✓ 7. void clear();

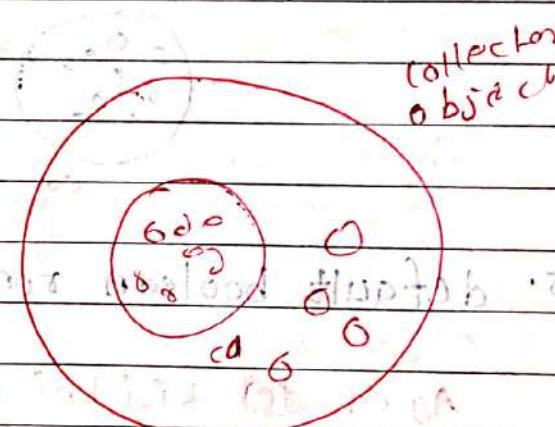
collection ke andher
Jitne bhi object hai
wah pura ke para
object ko empty
kar dega.



Saare ke saare objecte ko
empty kar dega.

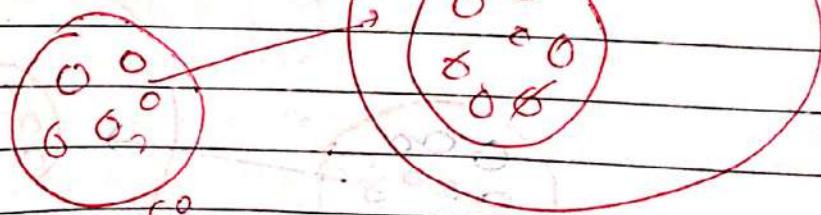
✓ 8. boolean contains(Object obj);

→ ye check karega
ke object jo hui
wah us wale
object me hain
ki nhi



✓ 9. boolean containsAll(Collection c);

ye check karega ki
sare collection usme
hai ki nhi



✓ 10) boolean isEmpty();

~~q1.isEmpty()~~: Empty check

Kawata Hui

: () about has to → return value

Individueller Antrag. Tzu kung

nhi' ba' i - ho → false

Kating Kargi.

~~11. int size()~~

collection object का size return करते हैं।

~~12. Iterator iterator();~~ ~~删除了~~

(collection set) element ko get karne ke liye to

Iterator method use kante hai

सन्तोष ग्रन्थालय | फॉर्म रेजिस्ट्रेशन

✓ 13. `ObjectToArray()`; `of` `base` `of` `Model`

coffee

collection object[] Array

me convert Karwai

~~to~~ to Array key

Use taught here

2001-09-16 00:00:00

~~1932. November 10. The following deposit~~

-chicken - - - - - chicken wings

② al.toArray();

14.) boolean equals (Object obj);
यह यहाँ पर्याप्त objects pass karwate hai
जो हमें दिया है।

15.) int hashCode();
इसका उपयोग object class ka method hai

=====

What is difference between Collection & Collections-

Collection

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

- ③ Collection interface contains abstract methods and static methods.

Collections

- ① Collections is a utility class.
- ② Collections defines several utility methods that are used to operate on collection objects like sorting, searching etc.

- ③ Collections class contains only static methods.

* What is Utility class in Java?

Ans utility class is also known as helper class which cannot be instantiated.



→ Jis class ka hum log object create nhi kar skte us class ko hum log utility class bolte hain.

→ Utility class contains only static methods.

→ Examples are, Arrays, Collections.

→ How we can create utility class :-

1. declare the class as public and final.

2. Final class Johai usko inherit nhi Karne dega.

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

```
public final class Test
```

```
private Test() { }
```

3

Class main

2 :

1/ Test t = new Test(); X nhii

3 Kaiska hui

Kay ki

private lag gaya hai

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

Code Snippet: public final class Test {

```
private Test () {}  
static void show()
```

class main

```
Test t = new Test(); // error.
```

4. every method should have deep documentation.

~~= = = = = = = =~~

What is Utility method?

→ Utility methods perform common, often reused methods.

Eg:

Utility methods are always static type.

⇒

Eg:- Examples are: Sorting, searching, methods, performing string manipulation,

methods connecting to database etc.

List (Interface) & ArrayList class

Date _____
Page _____

⇒ List Interface :-

→ List is a interface ~~which~~ which is present in Java.util package.

→ List is the child interface of collection interface.

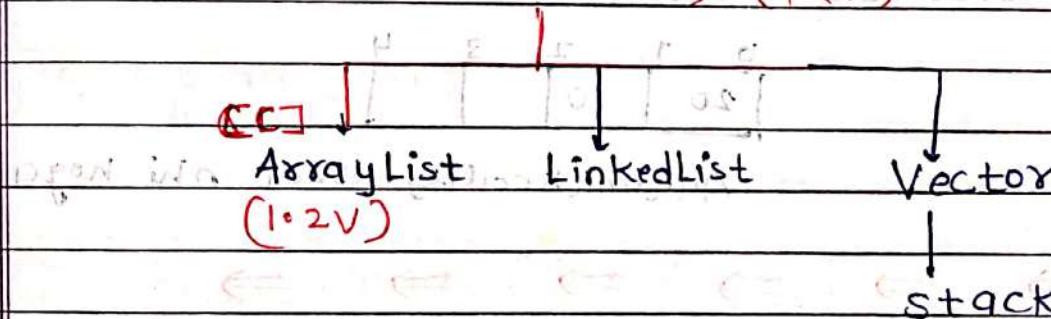
→ **Syntax :** public interface List extends collection { }

→ List was introduced in JDK 1.2 Version.

→

Hierarchy of List interface :-

List (I) (F2V)



→ Properties of List Interface :-

1. List is an index-based Data Structure. Which means that first element will be inserted at '0' index position.

0 1 2 3 4

2. List can store different data types or heterogeneous elements.

3. We can store duplicate elements in the List.

10 10 null null

4. We can store any number of null values

in list it finds the list position in the list
properly list due to

5. List follows the insertion order, which means the sequence in which we are inserting the elements, in the same sequence we can retrieve the elements.

Jai Jai App log List ke Another elements
provide knowledge. ~~use~~ use hi element ko
use sequence elements element ko retrieve
know skte hai.

6. List does not follow the sorting order.

0	1	2	3	4
20	10			50

Automatically sort nhi hogi.

⇒ ⇒ ⇒ ⇒ ⇒ ⇒

Methods of List Interface:-

1. List contains below the methods of

to implement interface contains method.

collection Interface ke same method.

List Interface me ~~me~~ bhi present hoga. kya ki List Interface ~~ko~~

so it inherits from Collection Interface

Ko "to" add method

2. void add(int index, object obj);

a1 →	10	20			
------	----	----	--	--	--

a1.add(10); a1.add(20);

directly element insert karwa dega.

a1.add(3, 5);

3.) boolean addAll (int index, collection c);

	0	1	2	3	4
a11	10	20	30	40	50

	0	1	2	3	4
a12	100	200	300		

q11.addAll(3, a12);

	0	1	2	3	4	5	6	7
a11.addAll(3, a12);	10	20	30	100	200	300	40	50

4.) object get (int index);

object obj = a11.get(5);

↓
300

remove
Koi dega

5.) object remove (int index)

object obj = a11.remove(5);

object
Ke Andhar
store karwao
daga.

6.) object set (int index, object newobj);
//set method is used to replace the object at given index position.

a11.set(4, 888);

	0	1	2	3	4	5	6	7
a11	10	20	30	100	200	300	40	50

a11.set(4, 888)

set
kar dega.

replace karwao
daga.

// It will return the index position of provided object and if object is not found then it will return -1

7.) int indexOf(object obj);

0	1	2	3	4	5	6	7
10	20	30	100	200	300	40	50

int i = arr.indexOf(50);

print(i);

eska index
no dega

int i = arr.indexOf(60);

Agaar Array me koi

value nahi milta hai to
wah (-1) return value

(-1) + 96 = Konsega 060

8.) int lastIndexOf(object obj);

0	1	2	3	4	5	6	7	8
10	20	30	200	200	300	40	50	100



if (i > 0) print(i = arr.lastIndexOf(100));

print(i); // 3

q1.lastIndexOf(100); // 8

for (int i = 0; i < arr.length(); i++)

if (arr[i] == 100)

return i;

else

→ ArrayList:

Date _____
Page _____

→ ArrayList is an implemented class of list

interface which is present in Java.util package.

→ ArrayList Jo hui List interface ko

inherit kerti hui. ye sare methods hui

Jinhi body nahi hoti hai. (-)

ArrayList ke method ke Andher body

banaya gya hui.

→ Syntax :- public class ArrayList extends

AbstractList implements List, Random
Access, Cloneable, Serializable

→ The underline Data-structure of ArrayList
is resizable array or growable array.

→ ArrayList was introduced in J.DK 1.2 version.

⇒ Properties of ArrayList :-

→ 1. ArrayList is an index based Data structure
which means that first element will be
inserted at '0' index position.

→ 2. ArrayList can store different data types
elements or heterogeneous elements.

→ 3. We can store duplicate elements in
the ArrayList.

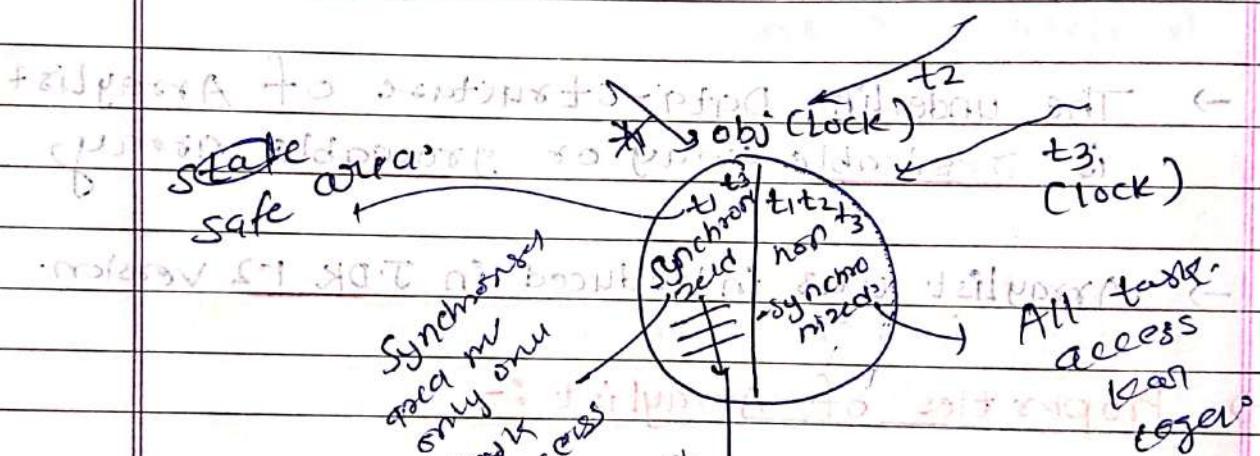
→ 4. We can store ~~not~~ any number of null
values in the ArrayList.

→ 5.) ArrayList follows the insertion order which means the sequence in which we are inserting the elements, in the same sequence we can retrieve the elements.

→ 6.) ArrayList does not follow the sorting order.

(above properties are same as List interface)

→ 7.) ArrayList is a non-synchronized collection because ArrayList does not contain any synchronized method.



→ 8.) ArrayList allows more than one thread

at one time.

→ 9.) ArrayList allows parallel execution.

All methods are Access
locked as they
are synchronized and used for parallel execution
and hence good for parallel execution

- 10) ArrayList reduces the execution time which in turn makes the application fast.

Application ki jo list hai,
wah fast ho Jayega.

- 11) ArrayList is not threadsafe.
→ 12) ArrayList does not guarantee for data inconsistency.

All methods are synchronized.
Same ke same thread ek sath us data ko access kar skte hain. Jiske karna wah Data inconsistency problem occur ho skti hain - ArrayList me.

* Working of an ArrayList :-

index pointer 1 2 3 4 5 6 7 8 9 10

91 →	10	20	30	40	50	60	70	80	90	100	110
------	----	----	----	----	----	----	----	----	----	-----	-----

q1.add(10), q1.add(20)

Jabhi humlog

new ArrayList

create karne hain. ArrayList a1 = new ArrayList();

Usi time yaha

par ek new

ArrayList

create ho Jayega

but uski capacity

kilni keegi. Humesha

10 deegi nahi

Kam hosi ya Jada

fix rheege.

① when we create default ArrayList, a new ArrayList with initial capacity 10 is created. (but size is 0)

(CurrentCapacity * 3 / 2) + 1

$$(10 * 3 / 2) + 1 = 16$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1	2	3	4	5	6	7	8	9	10					
10	20	30	40	50	60	70	80	90	100	110					

$$(16 * 3 / 2) + 1 = 25$$

- 2) When the ArrayList capacity is full, a new ArrayList will be created with new capacity.

The new capacity is calculated by this formula:-

$$(\text{CurrentCapacity} * 3 / 2) + 1$$

- 3) The all the element will be copied into the new ArrayList.

(And due to this reason performance of an ArrayList decreases)

- Then due to garbage collection, old ArrayList object will be deleted.

- When new ArrayList is created automatically, then reference variable will point to the new ArrayList.

So Then old ArrayList object will be not referenced by any reference and then garbage collection will delete that object.

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

(professionals) standards want to implement
a method to return the current size of a
list without changing the state of the
list. This method is called size.

designer proposed

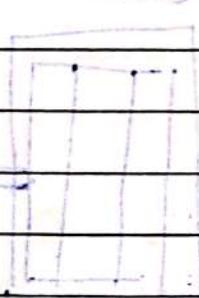
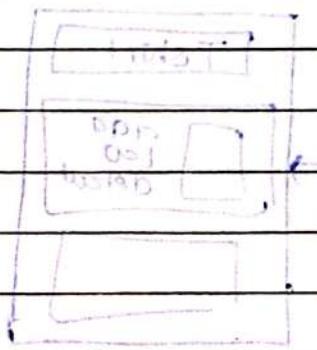
proposal for API

pg 3

new proposal must be

compared with

standard



old list & modern list

with standard list second follows

(creation phase)

Constructors of ArrayList :-

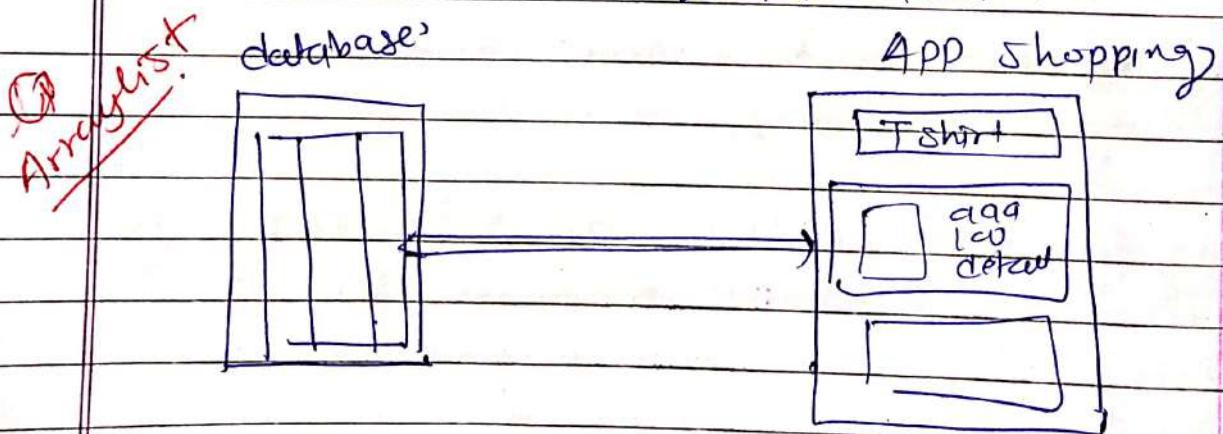
1. `ArrayList al = new ArrayList();`
* In this ArrayList, a collection object is created whose capacity is 10.
2. `ArrayList al = new ArrayList(int initialCapacity);`
* In this ArrayList, an ArrayList object is created with provided initialCapacity.
3. `ArrayList al = new ArrayList(Collection c);`
*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

* programs ArrayList.

- When we should use ArrayList?
- When we use retrieval operation mostly

Eg:-

value ko get karwan
ye kam kahta ha'



(retrieval operation is fast in case of ArrayList because it implements Random Access interface).

→ When we should not use ArrayList?

→ When we have insertion or deletion operation, then we should not use ArrayList.

Jaha Array Element ko se ~~get~~ shift
hoga karna hai waha ArrayList ko use

nhi karta hai → kyu ki performance slow ho Jayege

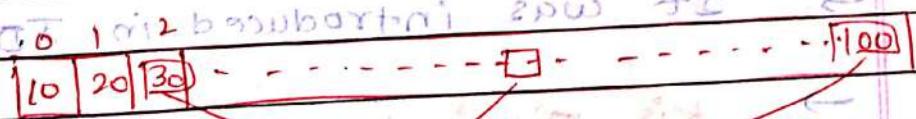
* RandomAccess Interface

RandomAccess Interface mein koi

→ RandomAccess interface is a marker interface that means it does not contain any methods or field (variables)

RandomAccess interface me koi contain ya method nahi hote hai.

→ ~~Marker~~



koi bhi element ko search karwao

uski speed fast hogi

same speed hoti hai

kyu ki RandomAccess

Interface ki wajah

→ The purpose of RandomAccess interface is to retrieve any random element in collection object at the same speed.

For eg:- we have collection object having 1 crore element, we have to search 3rd element or middle element or last element then it will search with the same speed.

→ There are only 2 classes which inherit the RandomAccess interface.

1. ArrayList class.

2. Vector class

* Cloneable Interface :-

→ Cloneable interface is also a marker interface.

→ It was introduced in JDK 1.0 version.

→ Kis object ke liye kam aata hai
Ye ek new object clone karwane ke liye kam aata hai
Ye kisi bhi object ko clone karwne ke kam aata hai

→ hum new operator use karne ki need nahi

→ directly clone karwa skta hai
object ko

→ If it is used to clone the object without
list - put using the new keyword, then
it will be known as shallow copy.

If it is used to clone the object with
deep copy then it is called deep copy.

If it is used to clone the object with
shallow copy then it is called shallow copy.

Now we will see how to implement shallow copy.

For this we have to define a copy constructor
which will take another object as argument
and will copy all the data members from
the source object to the destination object.

For example if we have a class named
Employee which has data members like
name, id, salary etc.

Then we can define a copy constructor
as follows:

Employee(Employee &obj) {
name = obj.name;
id = obj.id;
salary = obj.salary;

Now we can use this copy constructor to
copy one object to another.

For example if we have two objects
obj1 and obj2, then we can do
obj2 = obj1;

29/10/21

0153
0209

~~LinkList~~

Linked List

Date _____

Page _____

→ LinkedList is an implementation class of List interface which is present in java.util package.

→ Syntax :- public class LinkedList extends AbstractSequentialList implements List, Deque, Cloneable, Serializable { }

→ The underline data structure of LinkedList is Double Linked List or Circular Linked List.

→ LinkedList was introduced in JDK 1.2 version

* Properties of LinkedList :-

1.) LinkedList is an index based ~~data~~ data structure which means that first element will be inserted at '0' index position'

2.) LinkedList can store different data types or heterogeneous elements.

3.) We can store duplicate elements in the LinkedList.

4.) We can store any number of null values in the LinkedList.

5.) LinkedList follows the insertion order which means the sequence in which we are inserting the Element, in the same sequence we can retrieve the elements.

6. LinkedList does not follow the sorting order.

(i) because (Same points as list interface)
(ii) Head element always points to the first element)

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

8. LinkedList allows more than one thread at one time.

9. LinkedList allows parallel executions.

10. LinkedList reduces the execution time
which in turn makes the application fast.

11. LinkedList is not thread safe.

12. LinkedList does not guarantee for data consistency.

→ Working of LinkedList:-

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

a. Single Linked List

b. Double Linked List

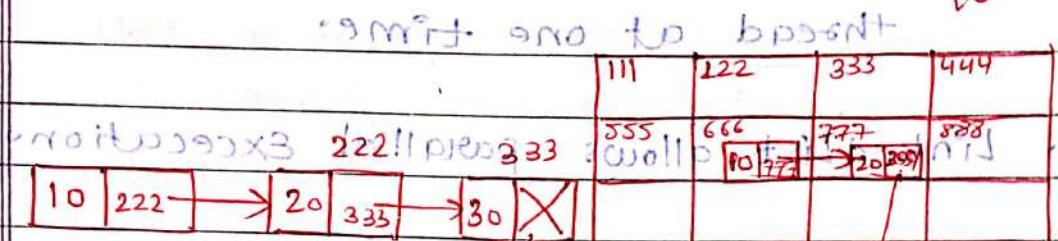
c. Circular Linked List.

[Java language.]

2.0.7 Linked List ~~is a one line with data structure~~
 in which elements are not stored in
 contiguous memory locations.

* Single Linked List: ~~basically~~

node ~~internal Address~~
data ~~internal Address~~
next ~~internal Address~~



Yaha kuchne must be done ~~in~~
 node bhi ~~Address~~,
 node ~~Address~~ ~~is in linked list~~

Ques ~~Linked List~~ ~~is a Node~~ ~~create node hai~~
 us me ~~data~~ ~~store~~ ~~node~~ ~~hai~~

[data, | Address]

hai ek node ke ander hamare pass
 two parts store node hai
 that is data, and Address.

a. Singly linked list

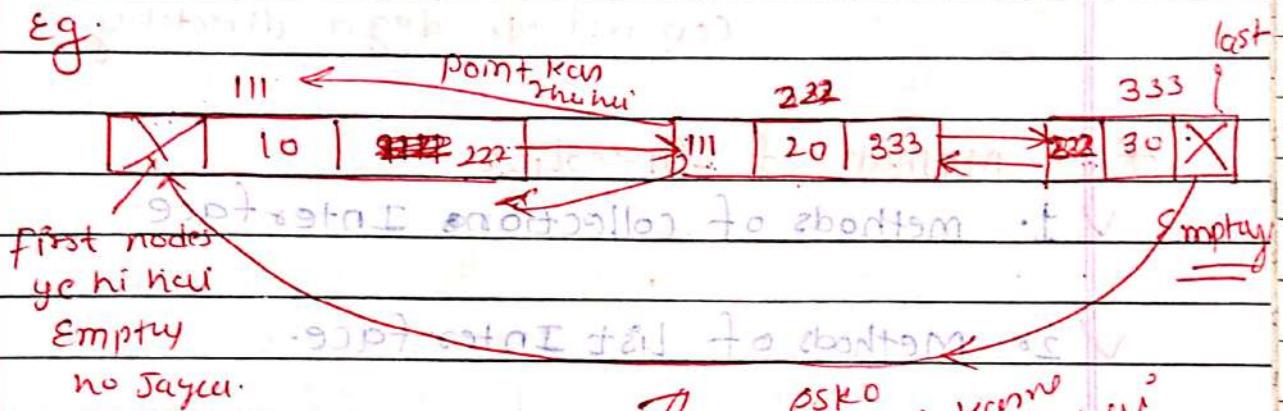
b. Doubly linked list

c. Circular linked list

(2) double Linked List

↑
priorus Address data. ↓ next node ka Address.

Eg.



(3)

Circular List:

Linked list me capacity nahi hoti hai

There is no capacity concept in linked list like ArrayList.



Constructors :-

1. public LinkedList ()

default constructor
on my one node created

2. public LinkedList (Collection c)

Kar dega

ArrayList ko Linked List me

convert kar dega. directly



Methods of LinkedList

✓ 1. methods of collection Interface

✓ 2. Methods of List Interface.

3. public void addFirst (object obj)

4. public void addLast (object obj)

5. public object getFirst ()

6. public object getLast ()

7. public object removeFirst ()

8. public object removeLast ()

only
linked
list

↑ linked list ke liye jaise list ke liye jaise

↑ linked list ke liye jaise list ke liye jaise

* When we should use LinkedList?

Ans: LinkedList is best when we have to use insertion or deletion operations.



Ji hum aye ArrayList me koi bhi element insert karwate the usme shift karna hoga, tha major LinkedList me major yea nhi hai ye do node ke bich connection me yeh node execute hoga jata hai to far to seba - jaisa to seba nahi

* When we should not use LinkedList?

Ans: LinkedList is worst in case of retrieval or searching operation (as LinkedList does not inherit ~~RandomAccess Interface~~)



Jab hum bar bar koi element retrieve karwana hai ya phir hum koi bhi element search karwana hai to hum log linked list ka use nahi karge.

(*) What is difference between ArrayList & LinkedList?

ArrayList

- ArrayList underline data structure is dynamic array or resizable array ~~linkedlist~~.

LinkedList

- LinkedList underline data structure is double linked list or Circular linked list.

- ArrayList stores the elements in contiguous memory locations.

- LinkedList does not store the elements in contiguous memory locations.

8. ArrayList acts as a List.

8. Linked List can act as a List or Deque.

4. ArrayList is good in case of retrieval operation.

4. Linked List is good in case of insertion and deletion operations.

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

5. Linked List is worst in case of retrieval operation.

* Vector (Legacy Class)

Date _____
Page _____

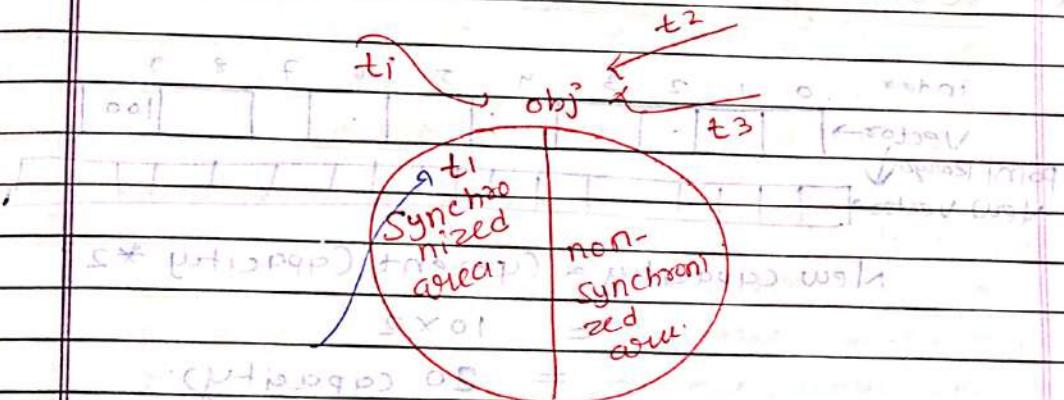
- Vector is an implementation class of List interfaces which is present in Java.util package.
- Syntax: public class Vector extends AbstractList implements List, RandomAccess, Comparable, Serializable
- The underline data structure of Vector is resizable array or growable array.
- Vector was introduced in JDK 1.0 version.
- Vector class is also known as legacy class.
(Legacy class kya hoti hai Jo ki pahle se hinde thi purane version ke Andher thi woh class jo purane version ke Andher thi abut Jab woh newe version ke Andher introduce hui to dubara hinde to woh restructured ya phir re-engineered ki gyi.)
- (Legacy class is the class which was formed in previous version and was restructured or re-engineered in new version)

5

* Properties of Vector

- Vector is an index based Data structure which means that first element will be inserted at 0 index position.
- Vector can store different data types elements or heterogeneous elements.
- We can store duplicate elements in the Vector.
- We can store any number of null values in the vector.
- Vector follows the insertion order which means the sequence in which we are inserting the elements, in the same sequence we can retrieve the elements.
- Vector does not follow the sorting order.
- Vector is synchronized collection because Vector contains many synchronized methods.
- Vector does not allow more than one thread at one time.
- Vector does not allow parallel execution or Vector allows sequential execution.

- Vector increases the execution time which in turn makes the application slow.
- A Vector is threadsafe.
- Vector guarantee for data consistency.



→ Agar harne pass method synchronized hote hain us case me shirf at a time ek hi thread yaha par aa payega.

kyu ki iske pass lock kota hai hoga, baki thread ke pass lock nahi hoga.

int wales wale area ko access nahi kar sakte

adjust par yehi hoga as shown above

multiple thread wah ek sath es wale area ko access nahi kar skte

Agar ek sath sare ke sare permission access nahi kar rhe hain to ese kya hoga, hoga jaga hi & execute speed jo hoga int of + wah 1% slow ho jayega. Application roti slow ho jaegi & uska use mark return do bhi jata hoga.

Synchronized area me multiple thread access nahi hone deti to thread sepe ho jayega.

→ Data consistency ki garantii data hai
 → ~~multiple~~ se kyu data yah yaha par
 ek hi time par. one thread
 access kar skte hai.



Working of Vector:-

index	0	1	2	3	4	5	6	7	8	9
Vector →										100
point karne wala ↓										
New Vector →										

$$\text{New capacity} = (\text{Current Capacity}) * 2;$$

$$= 10 * 2$$

$$= 20 \text{ capacity.}$$

- ① When we create a vector, a vector is present int of 'initial capacity' size create.



→ Pehle par 10 hogi or 10 se

→ Tab hum log vector created karne
 → tab woh new vector create
 → int ka karega 10 or uski capacity
 parle wale se jaidez hog Jayega.

- ② When the vector is full, then new vector will be created automatically
 with new capacity = previous capacity * 2;

→ 10 se 20 se 40 se 80 se 160 se 320 se 640 se 1280 se 2560 se 5120 se 10240 se 20480 se 40960 se 81920 se 163840 se 327680 se 655360 se 131072 se 262144 se 524288 se 1048576 se 2097152 se 4194304 se 8388608 se 16777216 se 33554432 se 67108864 se 134217728 se 268435456 se 536870912 se 1073741824 se 2147483648 se 4294967296 se 8589934592 se 17179869184 se 34359738368 se 68719476736 se 137438953472 se 274877906944 se 549755813888 se 1099511627776 se 2199023255552 se 4398046511104 se 8796093022208 se 17592186044416 se 35184372088832 se 70368744177664 se 140737488355328 se 281474976710656 se 562949953421312 se 1125899856842624 se 2251799713685248 se 4503599427370496 se 9007198854740992 se 18014397709481984 se 36028795418963968 se 72057590837927936 se 144115181675855872 se 288230363351711744 se 576460726703423488 se 1152921453406846976 se 2305842906813693952 se 4611685813627387904 se 9223371627254775808 se 18446743254509551616 se 36893486509019103232 se 73786973018038206464 se 147573946036076412928 se 295147892072152825856 se 590295784144305651712 se 1180591568288611303424 se 2361183136577222606848 se 4722366273154445213696 se 9444732546308890427392 se 18889465092617780854784 se 37778930185235561659568 se 75557860370471123219136 se 151115720740942246438272 se 302231441481884492876544 se 604462882963768985753088 se 1208925765927537971506176 se 2417851531855075943012352 se 4835703063710151886024704 se 9671406127420303772049408 se 19342812254840607544098816 se 38685624509681215088197632 se 77371249019362430176395264 se 154742488038724660352790528 se 309484976077449320705581056 se 618969952154898641411162112 se 1237939904309797282822324224 se 2475879808619594565644648448 se 4951759617239189131289296896 se 9903519234478378262578593792 se 19807038468956756525157187984 se 39614076937913513050314375968 se 79228153875827026100628751936 se 158456307751654052201257503872 se 316912615503308104402515007744 se 633825231006616208805030015488 se 1267650462013232417610060030976 se 2535300924026464835220120061952 se 5070601848052929670440240012384 se 10141203696105859340880480024768 se 20282407392211718681760960049536 se 40564814784423437363521920099072 se 81129629568846874727043840198144 se 162259259137693749454087680396288 se 324518518275387498908175360792576 se 649037036550774997816350721585152 se 1298074073101549995632701443170304 se 2596148146203099991265402886340608 se 5192296292406199982530805772681216 se 1038459258481239996506161154536232 se 2076918516962479993012322309072464 se 4153837033924959986024644618144928 se 8307674067849919972049289236289856 se 16615348135699839944095789472579712 se 33230696271399679988191578945159424 se 66461392542799359976383157890318848 se 132922785085598799532766357880637696 se 265845570171197599065532715771275392 se 531691140342395198130665431542550784 se 1063382280684790396261310863085101568 se 2126764561369580792522621726170203136 se 4253529122739160795045243452340406272 se 8507058245478320790090486904680812544 se 17014116490956407901808138083361625888 se 34028232981912807903616276166723251776 se 68056465963825607907232552333446503552 se 13611293192765120791446505466683101104 se 27222586385530240792893010933366202208 se 54445172770660480795786021866732404416 se 10889034554132096079157204373346480832 se 21778069108264192079314408746692961664 se 43556138216528384079628817493385923328 se 8711227643305676807925763498677186656 se 17422455266611333607911526981354373312 se 34844910533222667207923053962708746624 se 69689821066445334407946107925417493248 se 139379642132890668807982058850834969696 se 27875928426578133760796411770166939392 se 55751856853156267520792823540333878784 se 11150371370631253504079564708066755768 se 22300742741262507008079129416133511536 se 44601485482525014016079258832267023072 se 89202970965050028032079517664534046144 se 178405941930100056064079035329068092288 se 356811883860200112128078570658136184576 se 713623767720400224256077141316272369152 se 1427247535440800448512074282632446738304 se 2854495070881600897024078555264893476608 se 5708990141763201794048077110529786953216 se 1141798028352640358809615422055957386432 se 2283596056705280717619230844111914772656 se 4567192113410561435238461688223829545312 se 913438422682112287047692337644765908624 se 182687684536424457409538467528953817248 se 365375369072848914819076935057857634496 se 73075073814569782963815387011571532992 se 14615014728538565883630674022342665984 se 29230029457077131767261348044685331968 se 58460058914154263534522696089370663936 se 11692011728238531766854538017874133872 se 23384023456477063533709076035742666744 se 46768046912954127067418152071485333488 se 93536093825908254134836304142970666976 se 187072187658176508268732608285941333952 se 374144375316353016537465216571882667904 se 748288750632706033074930433143765335808 se 1496577501265412066149658866287533351616 se 2993155002530824132299317732575066673232 se 598631000506164826459863546555013334664 se 1197262001012329652919727093110026669328 se 2394524002024659305839454186220053338656 se 4789048004049318611678908372440106677312 se 9578096008098637223357816744880201354624 se 19156192016197274446715633489760402709248 se 38312384032394548893431266979520805418496 se 76624768064789097786862533959041610836992 se 153249536129578195573725667918083221673984 se 306499072259156391147451335836166443347968 se 612998144518312782294875671672332886695936 se 1225996289036625564597513423344665773391872 se 2451992578073251129195026846689331546783744 se 4903985156146502258390053693378663093567488 se 9807970312293004516780007386757326187134976 se 1961594062458600903350000777351465236269952 se 3923188124917201806700001554702930472539904 se 7846376249834403613400003109405860945179808 se 1569275249666880722680006218881173189035816 se 3138550499333761445360001237763463788071632 se 6277100998667522890720024755526927576143264 se 12554201997335045781440049511538555153286288 se 25108403994670091562880098777077110306572576 se 50216807989340183125760197554154220613145152 se 10043361597668036625520395110830844122628504 se 20086723195336073250560790022166168825257008 se 40173446390672146501120180444323377655114016 se 80346892781344293002240360888646755310228032 se 16069378556268458600448072177729511020456064 se 32138757112536917200896144355459022040912128 se 64277514225073834400172288710918044081824256 se 12855502845014766880344457742183608816364912 se 2571100569002953376068891548436721763273824 se 5142201138005906752137783096873443526557648 se 10284402276011813504275566193746887053115296 se 20568804552023627008551132387493774106230592 se 41137609104047254017002664774987548212461184 se 82275218208094508034005339549775096424922368 se 16455043641618901606801067909555098849844736 se 32910087283237803213602135819111097789689472 se 65820174566475606427204271638222195578378944 se 13164034913295121285440843317644439115675788 se 26328069826585242570881686635288878231351576 se 52656139653170485141763373270577756462703152 se 10531227930634097028352674654115551292446304 se 21062455861268194056705359308231102584892608 se 42124911722536388113410678616462205169785208 se 84249823445072776226821357232924410339570416 se 168499646891455532456426754465848820679140832 se 336999293782911064912853508931697641358281664 se 673998587565822129825707017863395282765563328 se 1347997171131644257654140357726790565531126656 se 2695994342263288515308280715453581131062533212 se 539198868452657703061656143090716226212566424 se 1078397736905315406323312286181432452425332488 se 2156795473810630812646624572362864904850669776 se 4313590947621261625293249144725729809101339552 se 8627181895242523250586498289451459618202679104 se 17254363790485466501173976578902919236405358088 se 34508727580970933002347953157805838472810716176 se 69017455161941866004695906315611676945621432352 se 13803491032388372009349181263122333389122864704 se 27606982064776744018698362526244666778245729408 se 55213964129553488037396725052489333564911458816 se 11042792825910697607479345010497866712882877632 se 22085585651821395214958687020995733425765755264 se 44171171303642790429917374041991466855315510528 se 88342342607285580859834748083982933710631021056 se 17668468521457116171968596016796586742126042112 se 35336937042914232343937992033593773484252084224 se 70673874085828464687875984067187554888504168448 se 14134774817165692935571968813437510977008333696 se 28269549634331385871143937626875201954016667392 se 56539099268662771742287875253550403908033341584 se 113078198537325543484557550507100807816066831168 se 226156397074651086969115101014201615632133662336 se 452312794149302173938230202028403231264267324672 se 90462558829860434787646040405680646252533464932 se 180925117657720869573292080811361292510533929664 se 361850235315441739146480161622723584850678593328 se 723700470630883478293210323245446716701357886656 se 1447400941261766956584220646490893434027155773216 se 2894801882523533913168441292981786868054315546432 se 578960376504706782633688258596357734805863109288 se 1157920753009413565267376517192755468011726521676 se 2315841506018827130534753034385115136023452543352 se 463168301203765426106950606877023027204685108704 se 926336602407530852213901213754046054409340217088 se 1852673204815061704427802427508092108818604234176 se 3705346409630123408855604855016184021637208448352 se 741069281926024681771120971003236804434161691712 se 148213856385204936354480961000647360886832333424 se 296427712770409872708961922001294721773666666848 se 592855425540819745417923844002589443547333333696 se 118571085108163949083584768800517888705466667392 se 237142170216327898167169537600103777410933337584 se 474284340432655796334339075200207554821866675184 se 948568680865311592668678150400415109643733355168 se 189713736173062318533756300080830309287466710336 se 379427472346124637067512600161660609568533305132 se 758854944692249274135025200323312191371467406664 se 151770988984489554827050500646661238274133410132 se 303541977968979109654101001293322476548267801324 se 607083955937958219308202002586644473193533820648 se 121416791187591643861640400517328994638707641296 se 242833582375183287723280800103465798937415368192 se 485667164750366575446561600206931597874291436384 se 971334329500733150893123200413863119545828732768 se 194266865900146630176246400822732623909175465536 se 388533731800293260352492800164546524781830911528 se 77706746360058652070498560032909314956366182256 se 155413492720117304140981200658186549127332364512 se 310826985440234608281962400131637309825466329024 se 621653970880469216563924800263274619651332658048 se 124330794176093843312848960

hum log vector create kiyah hai uski capacity
 to hui hui algan wahi full ho jati hai to
 and Agar usme humlog ~~new~~ element
 insert or kuchna hui tab wahi ek
 new vector create hogi jo and pahle wale
 vector ko pointed nhi karega but
 (new vector) ko pointed karega.
 and pahle wale vector ko garbage
 collector delete kar degi.

→ when new vector is created then all the
 elements from old vector will copied to
 new vector and then the reference
 variable will point to the new vector and
 garbage collector will delete the previous
 vector from the memory.

Note:- In vector we can find the
 capacity =

vector ke ander capacity

findout kar sake hai

using method \rightarrow banaya gaya

hai already

(constructor) \rightarrow size

Constructors of Vector:-

1. public vector()

⇒ ye ek vector create karega or eski capacity 10 hai agar vector full ho jata hai to new vector ki capacity 20 ho jaoge

2. public vector (int capacity)

Yaha kyun se capacity ~~de~~ do skte hui magar app jitna capacity dega

int mlf his create hoga woh uska vector ~~ki~~ create nahi hogaya usko

bita raha public vector (int capacity, int) ~~create~~ ~~int~~ ~~incremental ratio~~ add ho

for eg: hum logo ne 100 ~~provide~~ provide kar diya uske bad new capacity kitni hogi

200 hogi agar 200 full hogya tab 400 capacity ho jaoge

(100, 5)

Yeh to apki bar ratio me change hogi

Jis ki 10% complex hoga new capacity 100, 115, 120

4. public vector (collection C).

② vector contains all the methods of list interface

① vector contains all the methods of collection interface

1. public synchronized int capacity().

2. public synchronized void addElement(Object obj)

3. public synchronized Object firstElement()

4. public synchronized Object lastElement()

5. public synchronized boolean removeElement(Object obj)

6. public synchronized void removeElementAt(int index)

7. public synchronized void removeAllElements()

* When we should use Vector

Ans: We should use Vector in case of fast eval. searching operations. (vector inherits the RandomAccess interface)

→ When we should not use Vector

Ans: We should not use Vector in case of insertion or deletion of elements.

What is

Difference between ArrayList & Vector?

Date _____
Page _____

ArrayList vs Vector

- | | |
|--|---|
| ① ArrayList was first introduced in JDK 1.2 version. | ① Vector was introduced in JDK 1.0 version. |
|--|---|

- | | |
|------------------------------------|-----------------------------|
| ② ArrayList is not a legacy class. | ② Vector is a Legacy class. |
|------------------------------------|-----------------------------|

- | | |
|---|--------------------------------------|
| ③ ArrayList is non-synchronized collection. | ③ Vector is synchronized collection. |
|---|--------------------------------------|

- | | |
|--|---|
| ④ ArrayList allows more than one thread at one time. | ④ Vector does not allow more than one thread at one time. |
|--|---|

- | | |
|--|---|
| ⑤ ArrayList allows the parallel execution. | ⑤ Vector does not allow parallel execution. |
|--|---|

- | | |
|--|---|
| ⑥ ArrayList decreases the execution time which in turn makes the application fast. | ⑥ Vector increases the execution time which in turn makes the application slow. |
|--|---|

- | | |
|---------------------------------|------------------------------|
| ⑦ ArrayList is not thread safe. | ⑦ Vector is not thread safe. |
|---------------------------------|------------------------------|

- | | |
|--|---|
| ⑧ ArrayList does not guarantee for the data consistency. | ⑧ Vector guarantees for data consistency. |
|--|---|

- | | |
|--------------------------------------|--|
| ⑨ In case of ArrayList newCapacity = | ⑨ In case of Vector
newCapacity = presentCapacity * 2 |
|--------------------------------------|--|

$$\text{presentCapacity} * 3/2 + 1$$

- | | |
|---|---------------------------------------|
| ⑩ In ArrayList we cannot find the capacity. | ⑩ In vector we can find the capacity. |
|---|---------------------------------------|

Stack Legacy class

Date _____
Page _____

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

Properties of stack:-

1. Stack class was specially designed for Last In First Out (LIFO)
 - ↙ We can change this algorithm according to our requirement.
 - ↙ but ye wala algorithm Aap Apni requirement ke according change kar skte hain
2. Stack can be implemented using Array, ArrayList, LinkedList or Vector
3. Stack is also index based data structure

Working for stacks.

peek ke case mein top ind ait ki stack se

wala element dekhega kyun se
hai or wahi wala element

return karwao dega

magar Top wala element
waha par present
degaa.

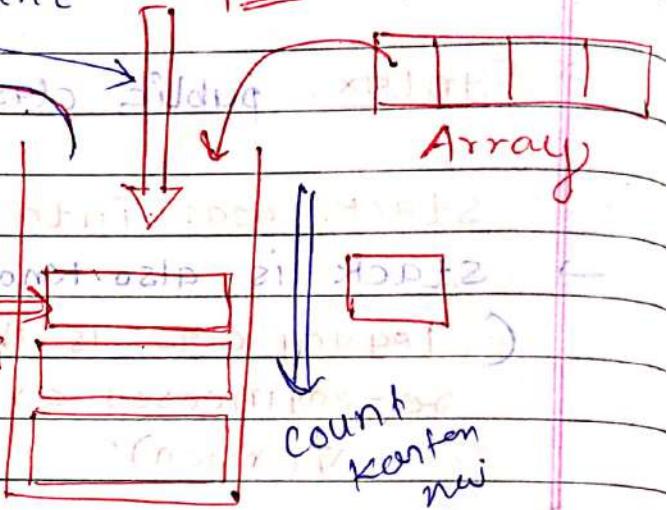
pop ke case me
TOP wala element

To set
Kardega or Top wala
Element ko
remove bhi
kardega

pop

TOP

bolt hai



Stack.

→ Jab hum log Element ko stack ke

Andhera put karwate hain to us wale
part (or -) push bolte hui

→ Jab hum log stack se elements ko
setare karte hain ya phir get

karwate hui to us wale part ko hum
log kya bol dete hui pop

hum log stack se upper side tak she hain

us punto ko peek bolte hui

hum log upper se element ko
peek karte hui

Last In FIRST OUT ka matlab:

Jo bhi element bad me aya hai hum
log inserter karwaoge wahi wala
element Sabse pahle setare karwa skte hain

Last In FIRST OUT

* Constructors:

1. public stack () { }

* Method:

1. Method of vector, list and collection,

2. public object push (object obj)

↓ Element insert karwa rhe hai

3. public synchronized Object pop()

yeho hai value ko return

Karwa deta hai

or value ko likal
dega.

4. public synchronized object peek()

5. public synchronized int search (object obj)

↓ kon sa method search
Karna hai

6. public boolean empty()

ye wala method check karega

ki stack empty hai ya

nhi

Cursors in Collection:-

Date _____
Page _____

→ Cursors :-

In Java, whenever we print then object reference, internally JVM will call `toString()` method of object class. In case of simple object it will print `className @ reference` but in case of printing collection object it will print the element present in collection object.



ArrayList mein aatamie usse ke sath element

print karwa sha that but mujhe kya kam karva
hai mujhe ek sath element ko print nhi
karani mujhe individually print karvaani
hui tab hum log use cursors ka

Agar hum log simple object ko print karva

hai to output provide `classname @`
reference.

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

* Types of Cursors:-

1. Enumeration

(most primitive and old fashioned)

2. Iterator

(ListIterator) (old fashioned)

3. (ListIterator) (old fashioned)

(new programming in java 8 & onwards)

→ 1. Enumeration:-

- ① Enumeration is the cursor which is used to get the elements one by one from the collection object.
- ② Enumeration ek cursor hota hai Jo collection object se values ko print karwata hai 'get'

→ ③ Enumeration cursor is used only for legacy class.

Enumeration cursor ko skip shrif legacy class ke liye hi use kya gaha hai

② Enumeration was introduced in JDK 1.0 version.

* Steps "how to use" Enumeration Cursor:-

1) Create Enumeration cursor object.

→ For this we have to use this public Enumeration elements() methods which is present in vector & stack Legacy class.

2) Read one by one all the elements from Enumeration cursor.

→ public boolean hasMoreElements()

→ public object nextElement()

(These methods are present in Enumeration interface)

→ Public class Test 2

- public static void main(String[] args)

2. 6

Vector v = new Vector();

v.addElement("aaa");

v.addElement("bbb");

for (v.addElement("ccc");

v.addElement("ddd");

Step ①

Enumeration e = v.elements(); Step ②

while (e.hasMoreElements())

int i = 0; System.out.println(e.nextElement()); Step ③

Output: aaaa bbbb cccc dddd

3

→ aaaa bbbb cccc dddd

aaa
bbb
ccc
ddd

①

copy kyu rahi hai

e enumeration ka object

yehi par
ek cursor
aa jaega

e.nextElement()

aaa
bbb
ccc
ddd

②

pski valitai ho
kar dega

yehi par
check
kroga element
hai ya nahi

aaa

point
krene
dege

bbb

ccc

ddd

limit

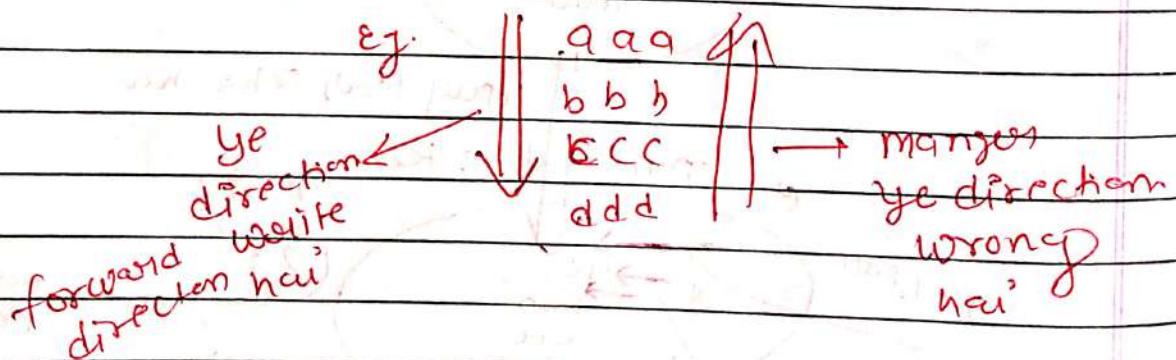


Limitations of Enumeration

1. It can be used only with Legacy class
(and thus it is not universal cursor.)
2. By using enumeration cursor we can only perform ~~read~~ operation but not update or remove operation.

3. It can be used to traverse the elements only in forward direction.

ye ~~is~~ shift forward direction. ~~merhi~~ ke andher hi element ko determine or traverse direction karwane ke liye kam aita hai



Iterator:-

- * 1) Iterator is a cursor which is used to get the elements one by one from the collection object.

Values Ko get Karwane Keliye Kam qta hai
- 2) It is universal cursor which means that we can use it with all collection objects.
- 3) It can be used for read and remove operation.
- 4) It was introduced in JDK 1.2 version.

* 4 steps of how to use "Iterator cursor":-

from 1996 for for full note see page 20

1. Create Iterator cursor object.
 2. Read one by one all the elements from iterator cursor, and modify them.
- ~~method~~ → public Iterator iterator()
(this method is present in collection objects)
- 1. public boolean hasNext()
 2. public Object next()
 3. public void remove()

(these methods are present in Iterator cursor interface)
borrowed to borrow ki ek samaaj se
method

Date _____
Page _____

remove karwana hai to

int i=0; of loop ki nahi hote pki yaar pki (1)

while (itr.hasNext()) no problem - ek mali

5

String element=(String) itr.next();

if (element.equals("bbb")) { (1)

itr.remove();

System.out.println ("Element removed successfully");



Limitations of Iterator & Cursor :-

1. It can be used only for read and remove operation but not for replacement or addition operation.
2. It can be used to iterate the elements only in forward direction.



ListIterator :-

1. ListIterator is a cursor which is used to get the elements one by one from a collection object.

2. ListIterator is a bi-direction cursor which means it can be used to traverse the elements in forward or backward direction.

3. It can be used to add, remove, insert and
replace operations

4. It was introduced in JDK 1.2 version.

Steps "how to use" ListIterator cursor:-

1. Create ListIterator cursor object

→ public ListIterator ListIterator();
(which is present in only List implementation classes)

2. Read one by one highlighted elements from

cursor of ListIterator cursor

→ public boolean hasNext() { Elements ko
forwards direction}

→ public Object next() { Ke Andher traversal}

→ public int nextIndex() { backward ke liye}

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

→ public boolean hasPrevious() { Elements ko
backward direction}

→ public Object previous() { Ke Andher traversal}

→ public int previousIndex() { backward ke liye}

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

→ public void remove() { Ke Andher traversal}

→ public void add (Object obj) { ke liye}

→ public void set (Object obj) { ke liye}

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

QUESTION

* Limitations of ListIterator cursor :-

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

QUESTION

* What is difference between Enumeration, Iterator & ListIterator?

Enumeration	Iterator	ListIterator
1. Enumeration can be used only with Legacy classes.	1. Iterator can be used for any collection object.	1. ListIterator can be used only for List implemented classes.
2. Enumeration can be used to traverse the elements only in forward direction.	2. Iterator can be used to traverse the elements only in forward direction.	3. ListIterator can be used to traverse the elements in forward and backward direction.
(3) Enumeration is used only for read operation.	(3) Iterator can be used for read and remove operations.	(3) ListIterator can be used for read, remove, add and replace operations.
(4)		

Set (Interface) & HashSet

→ Set (Interface)

1. Set is an interface which is present in Java.util package.
2. Set is the child interface of Collection Interface.

* 3. Syntax: public interface set extends collection { }

→ HashSet was introduced in JDK 1.2 version.

* → Hierarchy of set Interface.

* Properties of Set Interface:

1. Set is not an index based data structure, so when it stores the elements as per elements' hashcode values.
2. Unique Integer value hota hai jo use hi object create hota hai ushi time Heap manager create karata hai.

2. Set does not follow the insertion order.

(except LinkedHashSet) → LinkedHashSet

3. Set does not follow the sorting order.

(except SortedSet, NavigableSet, TreeSet)

ye thino sorting order ko follow karate hain?

SortedSet

new ArrayList

of nodes for sorted

hashmap

4. Set can store different data types or

(ii) heterogeneous elements (Except HashSet, NavigableSet, TreeSet) ^{Set of objects}
different data types ko store nahi karwa

abhi set ke hain → majority same data types ko
store karwe skte hain

5. We cannot store duplicate elements in set.

6. We can store only one null value in set.

* Difference between List & Set :-

"List"

"Set"

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

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

2. List allows duplicate elements.

2. Set does not allow duplicate elements.

3. List can store any number of null values.

3. Set can store only one null value.

4. List follows the insertion order.

4. Set does not follow the insertion order.

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

5. In case of set we can use only Iterator cursor.

6. List is used in case of retrieving the elements.

6. Set is used when we does not want to allow duplicacy.

* Method of feasible interface :-

- 1. Same methods as that of collection interface
- Approximate woh toh jon zah terleph ->

* HashSet :-

1. HashSet is an implemented class of set interface which is present in `java.util` package.

↳ AbstractSet implements Set, cloneable, Serializable & -> mit sno + p

3. The underline data structure of HashSet is Hashtable (HashSet is backed up by Map)

↳ HashSet holdai map ke according, ya phir Hashtable ke according kam kahta hai, donki amit yehi ha's underline data structure.

4. HashSet was introduced in JDK 1.2 version.



* Properties of HashSet :-

1. HashSet is not an index based data structure, it stores the elements according to elements hashcode values.
2. HashSet can store different data types of heterogeneous elements.
3. HashSet cannot store the duplicate elements.
4. HashSet can store maximum only one null value.

5. HashSet does not follows the insertion order

6. HashSet does not follow the sorting order.

(Same properties as Set interface)

7. HashSet is a non-synchronized collection

because HashSet does not contain any synchronized method.

8. HashSet allows more than one thread at one time.

HashSet Jo hai woh Thread ko ek sathe allows kar deya hai execution ke liye.

9. HashSet allows the parallel execution.

10. HashSet reduces the execution time which in turn makes our Application fast.

11. HashSet is not thread safe.

12. HashSet does not guarantee for data consistency.

Working of HashSet:-

Date _____
Page _____

- Hashmap ke another values koi kise type wise store nahi kar sakte hain. jaisa paaos se fit
- Hashmap ke another hamisha steady values paare ke form me data store kar sakte hain.

Map concept

key	values
1	aab
2	bbb
3	ccc
4	ddd
5	eee

- ① HashSet internally works on the basis of Hashtable (it internally backed up by map)

HashSet

key	value	present
1	aab	present
2	bbb	↓
3	ccc	object
4	ddd	class
5	eee	ka
6	fff	reference

- ② Koi bhi element Hashset me store karne hain wah Jmap me Jakan store hota hain jai ki waha Hashset me store hua. or map me key ~~me~~ ke another store hakan value me present hota.

Jiski present is object class ke reference hain

Another

2. When we insert any element, in HashSet it stores as a key inside the map and at value position PRESENT reference variable is stored which is the reference variable of object class.

3. Initial capacity of HashSet is 16 Element. (map ke bhi initial capacity 16 hoti hai)
16 maximum element store in HashSet

* HashSet ke 16 capacity kyu hoti hai

map ke initial capacity 16 hoti hai

6 entry by default hashset ke bhi capacity 16

4.) Its load factor or fill ratio is 75%.

16 ka 75% 12 hoga. / yeh phir 12 element ke Andar fill hogega wahan

Jaise hi ~~wah~~ map 75% fill ho

Jayega ushi time ek or map create ho Jayega. Tiski capacity

kitni hoga haygi eski double ho Jayegi

5.) After load is filled then a new map is created with its double capacity.

* Constructors of HashSet :-

Date _____
Page _____

1. public HashSet() { int i; } →

2. public HashSet(Collection<? extends E> c)
(object obj)

3. public HashSet(int initialCapacity, float
(10, 0.75f)
loadFactor) →

2 - 7 → 10 → 1000 → 75 full
no check for
initialCapacity → 1 → 1000.

4. public HashSet(int initialCapacity).

→
It is supporting initialCapacity, provide
no array koi state hai.

* Methods of HashSet :-

= approx same methods as that of Collection

but it is Set interface.

* Programs

* When We should use HashSet ?

Ans HashSet jo hai? search operators ke liye
bahut achha & hota hai silna.

Jobhi Data, Hashcode & value ke pair kam karti hui na
structures
(long) usi case me searching operation bahut
jaldi achha & hota hai?

→ HashSet is good for searching or
retrieving operations.

* LinkedHashSet (class)

- LinkedHashSet is the child class of HashSet which is present in Java.util package.
- Code Syntax :-

```
public class LinkedHashSet extends HashSet implements Set,
    Cloneable, Serializable { }
```
- The underline data structure of LinkedHashSet is "Hashtable + LinkedList".
- LinkedHashSet was introduced in JDK 1.4 version. J2SE 1.4

* Properties of LinkedHashSet :-

- 1. All properties of LinkedHashSet is same as HashSet except LinkedHashSet follows the insertion order.

(Isme insertion order follow karta hai?)
 (Or Sab same properties hai?)

→ Constructors of LinkedHashSet :-

1. public LinkedHashSet ()
2. public ^{linked} HashSet (Object obj)
3. public ^{linked} HashSet (int initialCapacity)
4. public ^{linked} HashSet (int initialCapacity, float loadFactor)

(Same Constructors as HashSet)

→ Methods of LinkedHashSet :-

⇒ Same methods as HashSet.

6

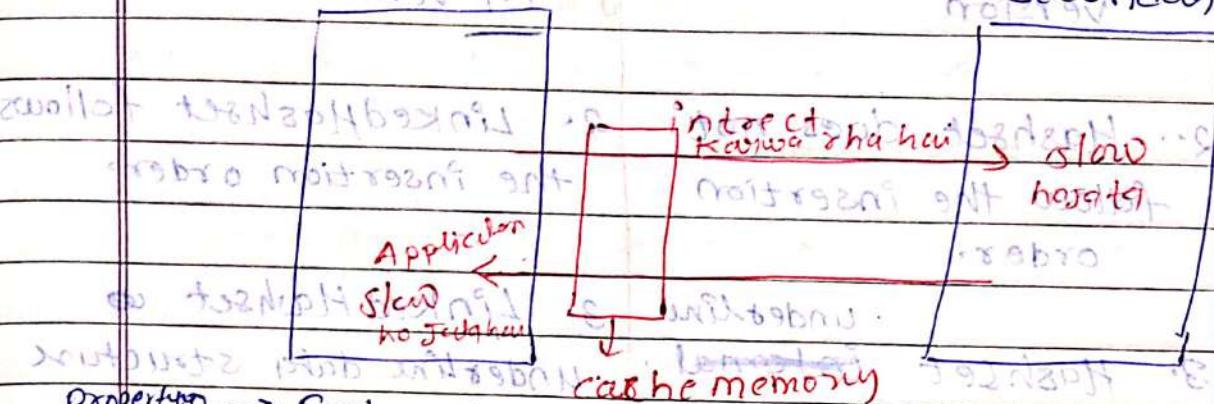
Ques. Explain what is meant by linked hashset.

* When we should use linked Hashset? - H

Ans: In application Primary memory (RAM)

Secondary memory (HDD)

Hard Disk.



Properties → Fast

→ Costly

→ We cannot store duplicate elements.

→ It follows the insertion order.

D) If we have to create cache based applications then we can use Linked Hashset.

↓
Linkedlist Jo hui cache based
Application ke liye use kola hui

Difference between HashSet & LinkedHashSet



1. HashSet was introduced in JDK 1.2 version.

1. LinkedHashSet was introduced in J2SE 1.4 version.

2. HashSet does not follow the insertion order.

2. LinkedHashSet follows the insertion order.

3. HashSet ~~internal~~ data structure is Hashtable.

3. LinkedHashSet ~~data structure~~ is "Hashtable + LinkedList".

5/ ~~both uses concept of synchronized interface~~
~~both have linked list and you can't change~~

* SortedSet :-

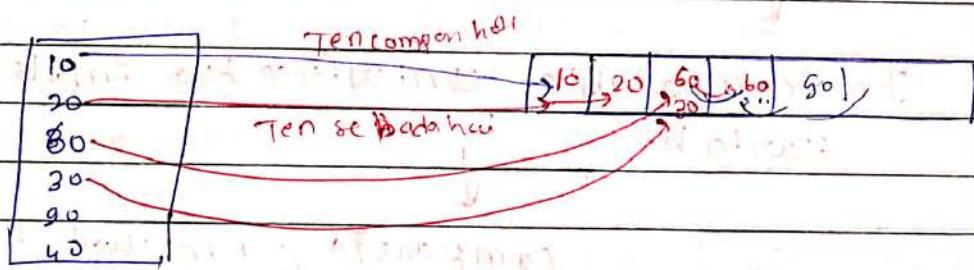
Date _____
Page _____

- SortedSet is a child interface of Set interface which is present in java.util package.
- Syntax: public interface SortedSet extends Set

→ SortedSet was introduced in JDK 1.2 version.

* Properties of SortedSet :-

1. SortedSet is not an index based data structure.
2. SortedSet does not follows the insertion order.
3. SortedSet follows the sorting order.

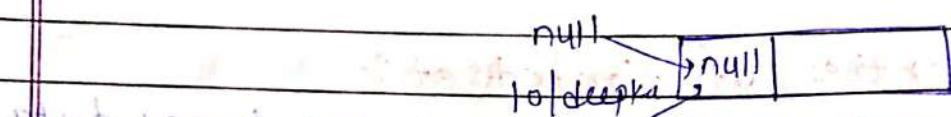


When 2 objects are compared then
these type of value are selected:

4. SortedSet can store same data types or homogeneous elements. (If we provide different data type element it will provide exception i.e. Java.lang.ClassCastException.)

5. SortedSet cannot store the duplicate elements.

6.2 We should not store null values in sorted set because sorted set follows the sorting order so while comparing the elements with null value, it will provide the Java.lang.NullPointerException.



7. SortedSet allows Comparable objects.

It means primitive data type is also Comparable. It inherit Comparable Interface.

Jo Comparable Interface ko implements karta hai

compareto → method hota hai

By default, but if we insert non-comparable objects, then it will provide an exception.

8. SortedSet is non-synchronized.

It is an API collection.

It is implemented by TreeSet class.

TreeSet is a class of java.util package.

b b b d d d
es ke bich ke elan
b b b sc 1 c Data
sc size & Elan
size & Elan
Previous
Page

* Methods of SortedSet :-

1. `SortedSet<E> subset(E fromElement, E toElement);`
2. `SortedSet<E> headSet(E toElement);`
3. `SortedSet<E> tailSet(E fromElement);`
4. `Object first();` khud ko bhi print karega
5. `object last();`
- 6.

* NavigableSet :-

→ NavigableSet is the child interface of SortedSet interface which is present in Java.util package.

* Syntax :- public interface NavigableSet
Extends SortedSet { }

→ NavigableSet was introduced in Java SE 6 version

* Properties of Navigable Set :-

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

* Methods

1. public NavigableSet descendingSet() accuse

2. public Object ceiling(Object obj) Andis

3. public Object higher(Object obj) nition
with
ccce

4. public Object lower(Object obj) an
el

(if no such object then return null)

5. public Object floor(Object obj)

6. public Object pollFirst()

7. public Object pollLast()

8. Object peek()

9. Object pollFirst() E

10. Object pollLast() E

11. Object peek() E

12. Object pollFirst() E

13. Object pollLast() E

14. Object peek() E

15. Object pollFirst() E

16. Object pollLast() E

17. Object peek() E

18. Object pollFirst() E

19. Object pollLast() E

20. Object peek() E

21. Object pollFirst() E

22. Object pollLast() E

23. Object peek() E

TreeSet (Class)

Date _____
Page _____

TreeSet is the direct implementation for NavigableSet interface.

but it also provides the implementation for SortedSet, Set & Collection interface.

Syntax :- public class TreeSet extends AbstractSet
implements NavigableSet, Comparable,
Serializable { }

↳ Inherit the most common wall TreeSet.

The Underline data structure of TreeSet is Red Black Tree.

↳ Most popular with TreeSet.

Tree set was introduced in JDK 1.2 version.

↳ Properties of TreeSet.

1. TreeSet is not index based data structure.
2. TreeSet does not follows the insertion order.
3. TreeSet follows the sorting order.
it may be default sorting order or
customized sorting order.

↳ Wah ho kya hui ki default sorting hu.

ga phir hamne customized kia hui se.

↳ customized kya hui.

Integers

→ 10, 20, 20, 40, 50

50, 40, 20, 20, 10 → customized
ikawaing phidta hai.

4. TreeSet stores the same data type elements or homogenous elements.

If we provide different data type elements in TreeSet then it will provide Java.lang.ClassCastException.

5. TreeSet cannot store's the duplicate elements

6. TreeSet can store only one null value

7. TreeSet is a first in first out collection

kyu hota hai kyu ki TreeSet ke
Another koi bhi method synchronized
nhi hota hai

8. TreeSet allows more than one thread at
one time

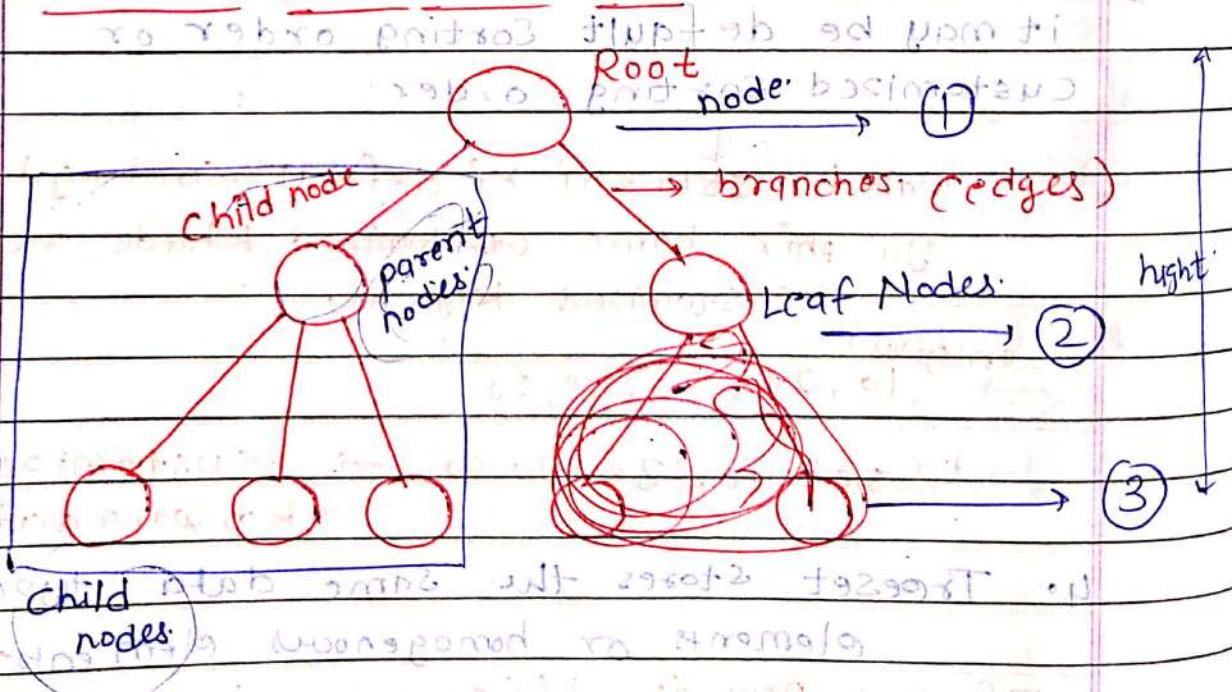
9. TreeSet allows the parallel execution

10. TreeSet reduces the execution time which
makes our application fast

11. TreeSet is not threadsafe

12. TreeSet does not guarantee for data
consistency

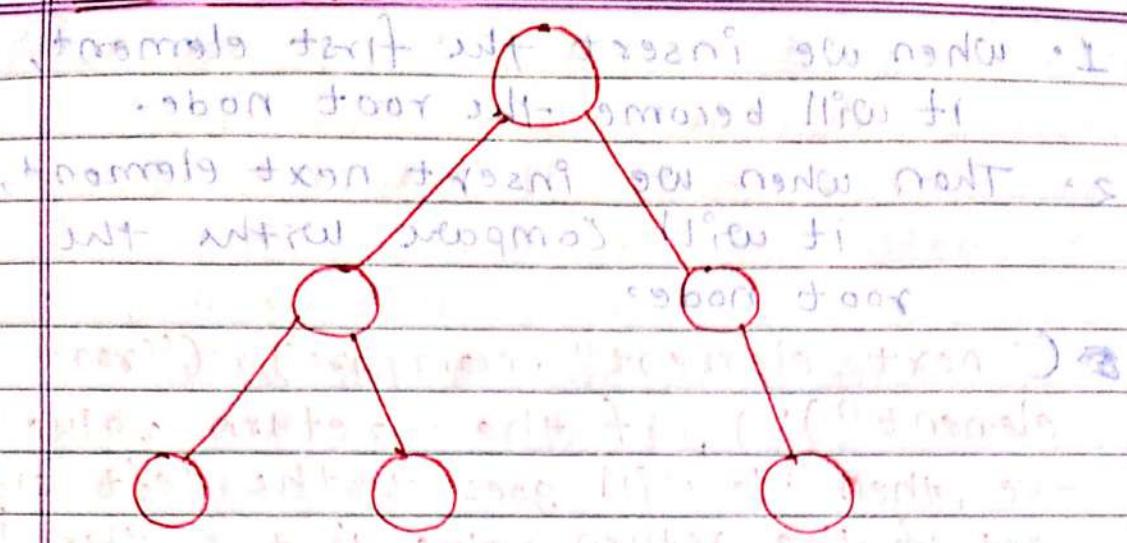
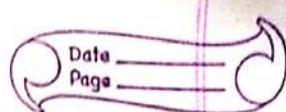
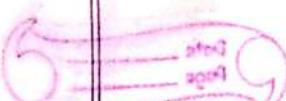
* Working of TreeSet



balanced Tree

Treeset

Date _____
Page _____



```
TreeSet ts = new TreeSet();
```

```
ts.add("EEE"); // first element, it will create  
// as root node
```

```
ii) ts.add("CCC"); // "CCC".compareTo("EEE") is -ve.  
// so it will add in left side.
```

```
iii) ts.add("BBB"); // "BBB".compareTo("EEE") is +ve.  
// so it will add in right side.
```

```
iv) ts.add("DDD"); // "DDD".compareTo("EEE") is +ve.  
// so it will add in right side.
```

```
v) ts.add("CCC"); // "CCC".compareTo("BBB") is +ve.  
// so it will add in right side.
```

```
vi) ts.add("AAA"); // "AAA".compareTo("BBB") is +ve.  
// so it will add in left side.
```

```
vii) ts.add("FFF"); // "FFF".compareTo("AAA") is +ve.  
// so it will add in right side.
```

gehi - small val

large val

1. When we insert the first element, it will become the root node.

2. Then when we insert next element, it will compare with the root node.

~~("next element" . compareTo ("root element"));~~, if the return value is -ve, then it will goes to the left side and if the return value is +ve then it will goes to the right side.

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

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

* Constructors:-

1. `public TreeSet()` // It will create an empty TreeSet object where the elements are inserted according to natural default sorting order.



ek empty TreeSet object create kar dega. Jaha par element default sorting order me ko insert ho jaiye ge.

2. ~~empty~~ public TreeSet (comparator compare)

// it will create an empty TreeSet

object where the elements will be

inserted according to the customized

sorting order.

↓ sorting + sort

Agar kuch mera hi sorting order

create karne hai to mai use

inkarun ga. ATM of bank - prop - sort

3. public TreeSet (Collection c) //

We can pass any other collection objects.

↓

yapakoi bhi collection objects pass karwa

uske hain pahli 2nd 3rd

4. public TreeSet (SortedSet s) //

↓

sortedset ka koi bhi object pass

karwa skte hai

Methods of TreeSet:

1. Contains all the methods of Collection, Set, SortedSet & NavigableSet interface.

When we should use TreeSet:-

when we want to store the ~~elements~~ !

large number of elements in sorting order.

↓ baahut sare element ko sorted

order me store karwana ~~hain~~ to

chahate hain to use kar skte

and due to this retrieval operation is fast

(X)

Cases for "null" values insertion in TreeSet:-

1. We can store only one null value.
2. We can insert null value only at First position, but if we insert the null value at second position, then it will provide an exception.
3. Java.Lang.NullPointerException.

any other element then it will provide "Java.Lang.NullPointerException"

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

1.6 version ke bad null value user nahi kar skte.

Kyu baaki ke element insert

to memory nahi hoga.

garibat ki zaroori nahi hoga.

Statement 1: null blinks off model

Statement 2: off size of tree on node ke ek part o3 in elements to remain apna.

Statement 3: off size of tree on node ke ek part o3 in elements to remain apna.

Statement 4: off size of tree on node ke ek part o3 in elements to remain apna.

Statement 5: off size of tree on node ke ek part o3 in elements to remain apna.

Statement 6: off size of tree on node ke ek part o3 in elements to remain apna.

~~Q~~ What is difference between HashSet, LinkedHashSet & TreeSet?

HashSet:

1. HashSet underline data structure is "Hashtable".

2. HashSet does not allow insertion order.

3. HashSet does not allow the sorting order.

4. HashSet allows the heterogeneous objects.

5. HashSet allows the null values for insertion.

LinkedHashSet & TreeSet:

1. LinkedHashSet is underlined data structure is "Hashtable + LinkedList".

2. LinkedHashSet allows the insertion order.

3. LinkedHashSet does not allow the sorting order.

4. LinkedHashSet allows the heterogeneous objects.

5. LinkedHashSet allows the null values for insertion.

TreeSet:

1. TreeSet underline data structure is "Balanced Tree".

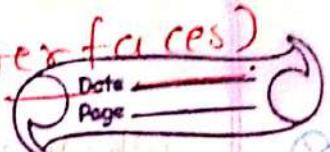
2. TreeSet does not allow the insertion order.

3. TreeSet allows the sorting order.

4. TreeSet does not allow the heterogeneous objects.

5. TreeSet allows the null value insertion but at first position. (but this is applicable for Java 1.6 version).

Comparable & Comparator (Interfaces)



Comparable:-

- 1. Comparable is an interface which is present in Java.lang package.

- 2. It contains only one method i.e. `compareTo()`.

* Prototype: `public int compareTo(Object obj)`

`obj1.compareTo(obj2);`

ye three values return karta hai
+ve - if obj1 is greater than
obj2

-ve - if obj2 is greater than
obj1

→ Jitne bhi wrapper class hai and including
String wah kya kam karte hai? Comparable
Interface ko inherit karte hai and
compareTo method ko implement kiga
hoga.

- 3. String and all wrapper classes (.) implements Comparable interface.

- 4. Problems with Comparable Interface.

1. By implementing Comparable interface, the properties of original class will get changed.

2. By this way we can sort only for one entity for example

We can sort the student object either with name or rollno at one time.

* Comparators

→ 1. Comparator is an interface which is present in java.util package.

return () in Comparator interface contains 2 methods.

1. public int compare(object obj1, object obj2)

→ if compare() return 0 then both objects are equal.

2. public boolean equals(object obj)

→ returns true if

both objects are equal.

* Map to compare and add different elements according to length and alphabetical order.

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

e.g. customize how to use compare.

comparator ko use kरजे.

* Comparator (2) का सफारी बढ़ावा देता है।

यह एक अल्फावेटिक सौचार्य है।

(प्रथम वाक्य में)

long x3

* What is difference between Comparable & Comparator interface?

is different from Comparable in terms of functionality.

Comparable

Comparator

- | Comparable Interface | Comparator Interface |
|---|--|
| 1. Comparable interface contains compareTo() method. | 1.) Comparator interface contains compare() method. |
| 2. Comparable interface is present in java.util package. | 2.) Comparator interface is present in java.util package. |
| 3. By using comparable interface original class properties will not be changed. | 3.) By using comparator interface original class properties will not be changed. |
| 4.) Comparable interface can sort only one entity at one time. | 4.) Comparator interface can sort multiple entities at one time. |
| 5.) Comparable interface is used to implicit sorting. | 5.) Comparator interface is used for customized sorting.
↳ Explicit. |

Queue (Interface); Priority Queue (class)

* Queue

1. Queue is the child interface of Collection

* Syntax:- public interface Queue extends

Collection { }

Word 2. Queue was introduced in JDK 1.50 version

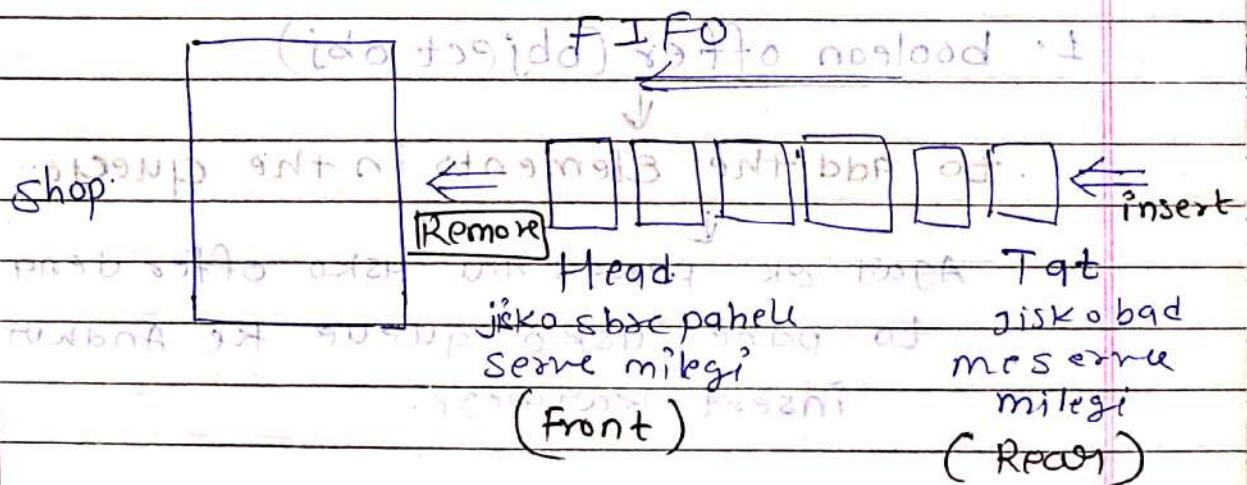
3. Queue orders the elements in FIFO (First In First Out) manner, but we

can change this algorithm according to

our requirements.

ye wala algorithm ko hum log
change kar skte hain.

* Implementation of Queue: Array



* Hierarchy of Queue:

on file

Priority Queue (class)

Abstract Queue (interface)

Normal Queue (class)

Word 1. Now if I want to know what is the element at index 2

"front" or "remove" function is used

* Properties of Queue:

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

* Methods of Queue:

1. boolean offer(Object obj)

↓

to add the elements in the queue.

Agar ek person hai usko offer dena hai

to phe usko queue ke andar.

insert (karwao).

- 2) object peek() → it will return the head element of the queue. If no element is found in the queue it will return null value.
- 3) object element() - It will return the head element of the queue. If no element is found, it will throw an exception i.e. "NoSuchElementException"

6
Date _____
Page _____

DEQUE

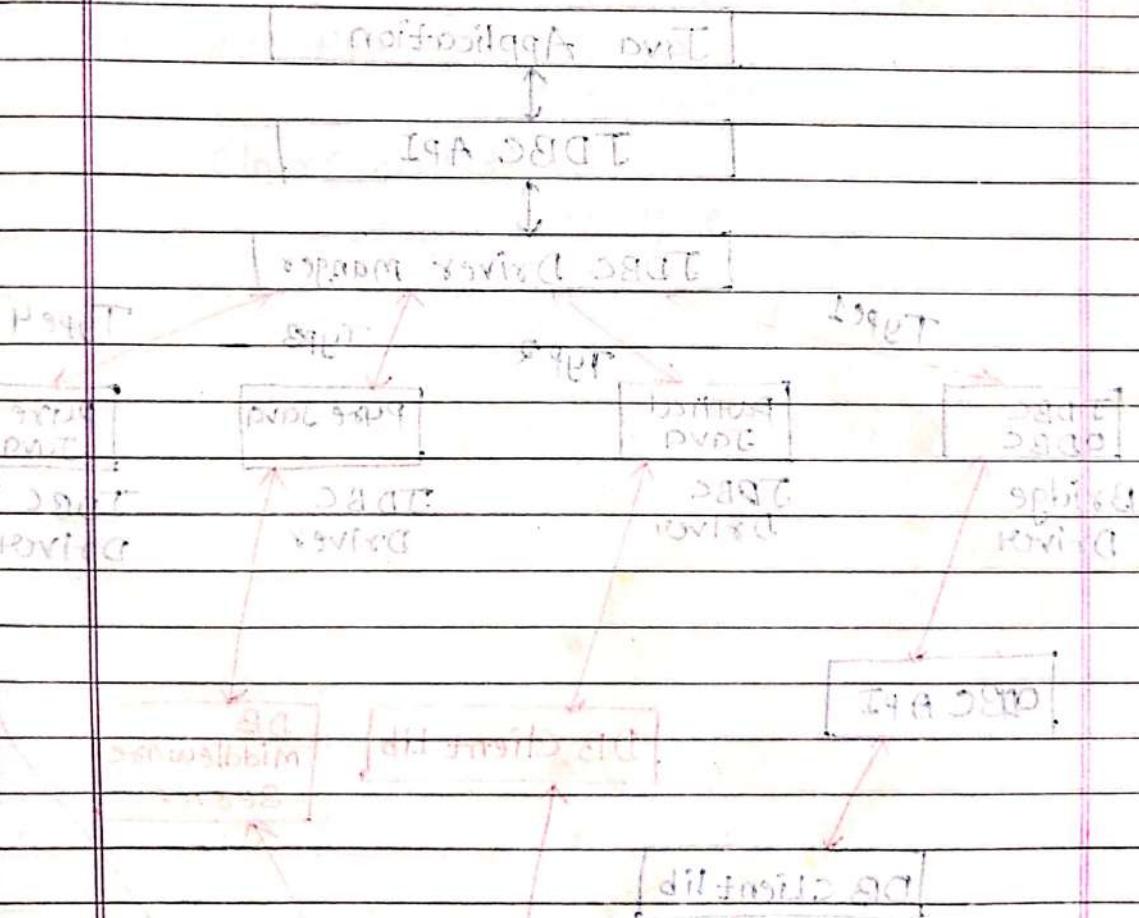
Date _____
Page _____

4. object poll() → It is used to remove the head element and also it will return that element. If no element is found, then it will return null value.

same
reason

5. object remove() → It is used to remove the head element and also it will return that element.

If no element is found, it will throw an exception i.e. "NoSuchElementException".
(Java.util.)



28/07/24
Date _____
Page _____

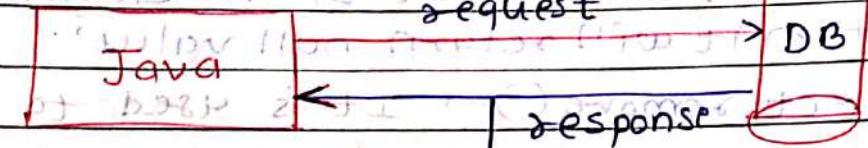
JDBC: Java Database CONNECTIVITY:

Date _____
Page _____

Import of class in JRE () log file of DB

File for JDBC driver naming by DB

Format on filename swift 03096

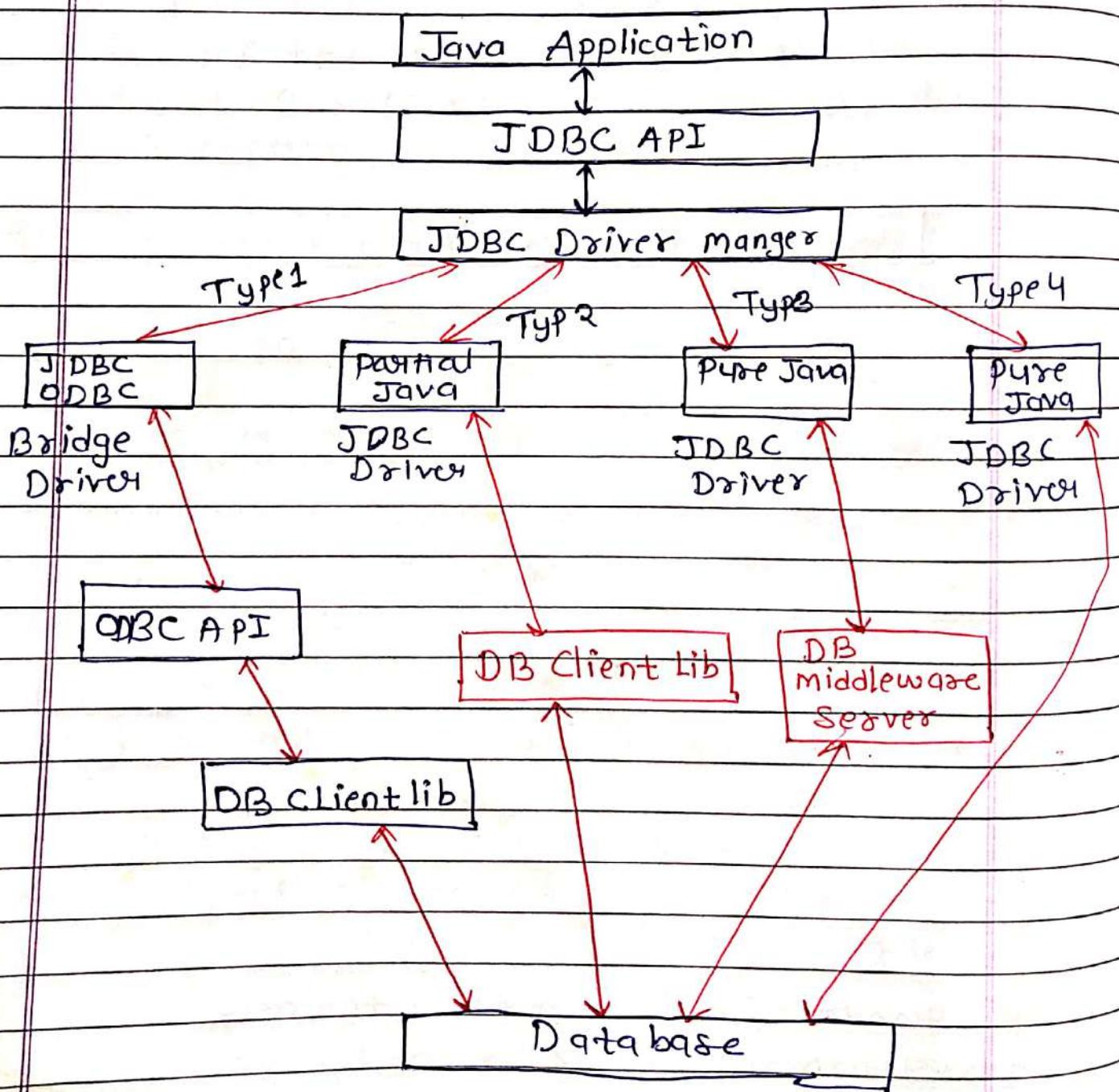


DB API format by DB and format by DB and same

format about drivers like, Oracle, MySQL, etc.

Want to (how to, how to) on API

(Custom API)



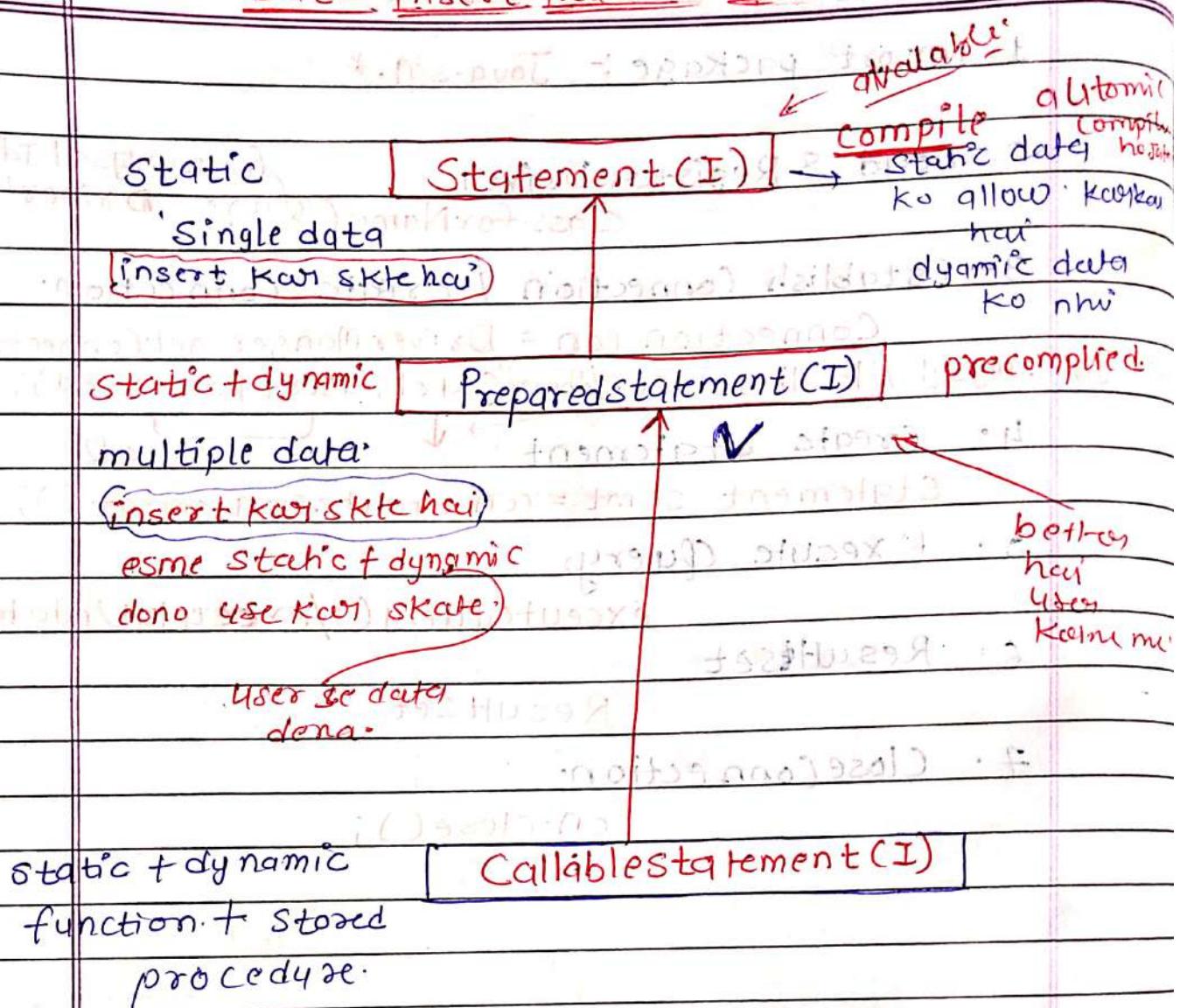
7 Step to connect with DB

1. import package :- Java.sql.*
2. Load & Register Driver
`(com.mysql.jdbc.Driver);
Class.forName("com.mysql.jdbc.Driver");`
3. Establish Connection / create Connection.
`Connection con = DriverManager.getConnection(url, "username", "password");`
4. Create Statement
`Statement stmt = con.createStatement();`
5. Execute Query
`executeQuery() / executeUpdate()`
6. Resultset
`ResultSet`
7. Close Connection
`con.close();`

29/10/21

Date _____
Page _____

Data insert kaiwana hai



→ Connection ko bharne ~~pehle~~ declare kai
set result dege:
prepared statement
sun mush.

or try catch me user use karne
or finally me user
closed kai dege

(2) try catch local ~~variable~~ variable
user finally use ~~ko~~ nhii
kai skte
usko

02/08/20

project: onlineBookStore

Books

Customer, Cart, Order

package:

pojo (plain old java object)

private data members

public constructor

Customer
Vendor
Wise

getter, setter

Con - non-param, param

toString()

pack

dao (data access object)

Interface

add, delete method

get, set, update

likewise

Class

package

TestCustomer

class

logical point

likewise add dao here

DB connection

Utility

DBconnection → class create kardege.

connect to Java code

→ com.bookstore.pojo → package

Book (class)

```
private int bookId;  
String bookName; // title, size  
String authorName; // name, address  
String publisherName;  
double rating;  
double no-of-copies;  
bookPrice; // money - non - zero
```

* Book (table) (- DBSE me legi)

* Book Dao (I) (- abstract method legi)

flameh method

boolean addBook() → All information add karne hui to (Book b); legi

boolean updateBook() → *

boolean deleteBook() → (int bookId)

List<Book> showAllBook(); → Yeha ko parameter nahi hota hai

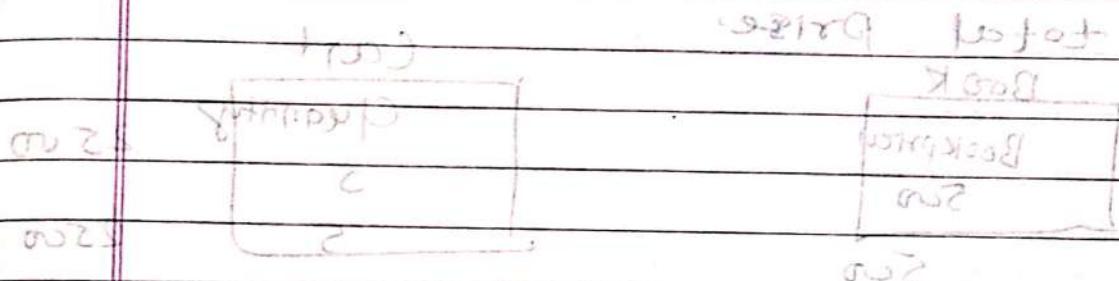
Book → ~~searchBook()~~ → (int bookId);

Book → searchBookById();

Book → ~~searchBookByName()~~ → (String bookName);

List<Book>

* Book Dao Impl(c)



* Book Test(c)

Customer & Book

(Customer (pojo))

```

int custId;
String custName;
String custEmail;
String custPassword;
String custAddress;
    
```

Customer Dao

```

boolean addCustomer(Customer c);
boolean updateCustomer (Customer c);
boolean deleteCustomer (int custId);
List <Customer> showAllCustomers();
Customer searchCustomerById (int custId);
List <Customer> searchCustomerByName (
    String custName);
    
```

Customer search (CustomerByEmail (String custEmail))

order:

total price

Book

BookPrice

500

2500

500

Cart

Quantity

5

2500

5

2500

5000

Select * from book b inner join

cart ca on b.bookid = ca.bookid where

custemail = ?;

(0.09) seconds

1 row selected

1. ~~StringJoiner~~ ~~StringBuilder~~

do string ko join karne ke kom
 Karta hain ~~is ka~~ ~~is ka~~ ~~is ka~~

2. Default Method: ~~Static~~ Method → use same

Same line of code multiple classes.

ek line ko multiple classes me

use kar skte hain

3. LocalDate date = LocalDate.now();

System.out.println(date); print current

Date bataya.

// off

LocalDate date1 = LocalDate.of(2012, 11, 23)

s. println(date1);

LocalDate date2 = LocalDate.of

(2010, Month.JANUARY, 17);

// Zone Aware

LocalDate date3 = LocalDate.now(zoneId.

of("Asia/Kolkata"));

s. println(date3);

// pure date me kar skta Day change

LocalDate date4 = LocalDate.of(Year.Day

(2019, 5)); (50), (35)

s. println(date4);

// ofEpochDay by default date--

01/09/1970

LocalDate date5 = LocalDate.ofEpochDay(5);

11 of year of DOU of second of Day + 2

X Lambda Expression:

@FunctionalInterface
interface Vehicle {

void run();

3

imp vehicle
class MyClass {

Vehicle v = () -> void = () -> void;

main() { v.run(); }

3 3

Vehicle v = new Vehicle() {
 void run() {

3 3

(F1) NAVIAT -> 3m

3

multiple interface in single class

yes. kar skte hain