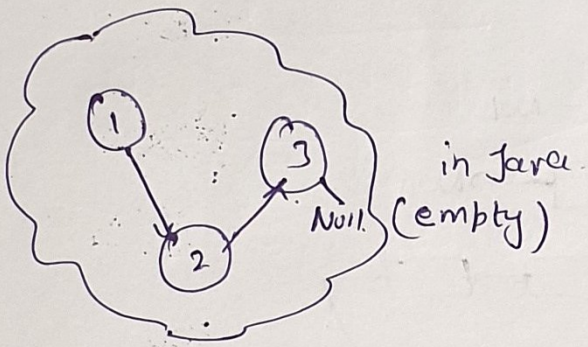
Linked List

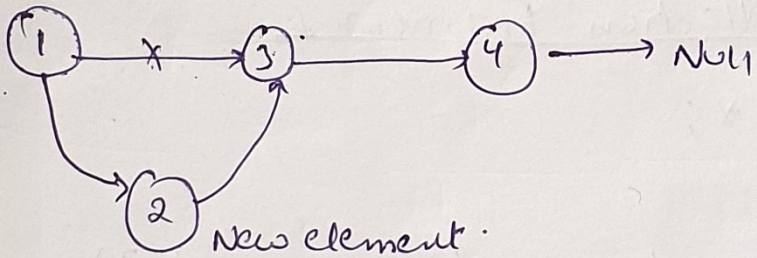
- We have list of node that is connected with each other with Links.
- In arrayList whenever we have an element to be ~~inserted~~ insert in middle, we have to move other elements one position backward.

∴ Time complexity for Insert operation → O(n).

- for searching operation in Arraylist time complexity is O(1). This is because we can search element directly with index. arr[0].
- In LinkedList memory is non-contiguous; we store values at different memory chunks



Insertion in Linked List



* for Insertion operation we P LinkList rather Array List.

for Insertion we are performing just 2 & 3 operations when no. of operation are constant complexity is O(1)

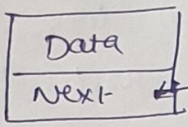
Search in Linked List. For searching we need to tra each and every node to find the address of next time complexity O(n)

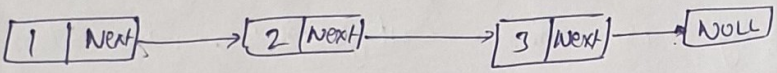
* for searching we Prefer Arraylists rather then Link

Properties of Linked List

1. Variable size
2. Non-contiguous Memory
3. Insert → O(1)
4. search → O(n)

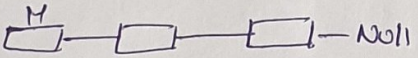
Structure of Linked List →

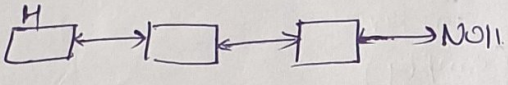


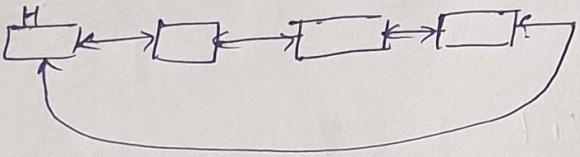


- form a chain Type structure

Types of Linked List.

1. singular 

2. Double 

3. circular 

Linked List in collection framework

```
import java.util.*;
  class LL {
  Public static void main(String args[])
{
LinkedList<String> list = new LinkedList<String>();
      list.addFirst("a");
      list.addFirst("is");
     System.out.println(list);
}
```

O/P.
[is, a]

```
list.addLast("this");
System.out.println(list);
System.out.println(list.size()
//loop
for(int i=0; i<list.size(); i++)
{
System.out.println(list.get(i)+" -> ");
}
System.out.println("null");
}
list.removeFist();
System.out.println(list);
list.removeLast();
System.out.println(list);
list.remove(3);
System.out.println(list)
}
}
```

[is, a, this] (2)

(3)

for searching

[if(list.get(i)==value)]

o/p

[is → a → this → null]

[a, this]

[ ]

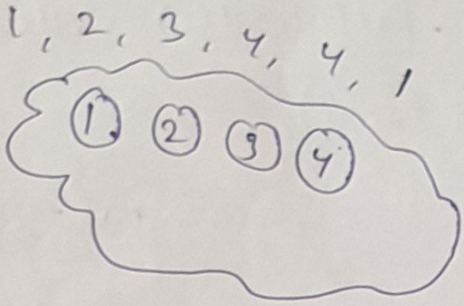
Index.

→ Remove element from particular node

HashMap, HashTable, Hashing.

HashSet

- Set word is drived from Math



- Set did not allow duplicate values.
- Set is a structure which have only unique values
- To implement this Property (Uniqueness) in Java we use HashSet
- Internal Implementation of HashSet & HashMap is done with HashTable
- HashSet is important because of its time complexity.
  - insert / Add → O(1)
  - Search / contains → O(1)
  - Delete / Remove → O(1).

  } All operations have constant Time complexity thats why it is important.

→

```
import java.util.HashSet;
Public Class Hashing.
{
Public static void main (String args[])
{
// Creating.
   HashSet<Integer> set = new HashSet< >( );
```

← import java.util.Iterator

```
// Insert
  set. add (1);
  set. add (2);
  set. add (3);
  set. add (1);

// Search - contains function
  if (set. contains (1))
  {
  System. out. println ("set contains 1");
  }
  if (!set. contain(6))
{
System. out. println ("does not contains");
```

→ False ; Negation op. make it True & Next st executed.

```
// Delete → Remove function
   set. remove(1);
   if (! set. contains(1)
   {
   System. out. println ("we deleted 1");
   }
   // size
   System. out. println ("size of set is :" + set. size());
```

→ O/P is 2.
Because at start we add 4 element but it will stay only 3, Because 1 is duplicate & we remove 1 also.

```
// Print all elements
   System. out. println (set);
```

→ [1, 2, 3] o/p

//Iterator → used to traverse set ; Becoz set does not have any index; It uses a special Iterator for this import Iterator.

import java.util.Iterator

iterator is a method of set

* Iterator it = set.iterator();

// Iterator has two function ① hasNext ② next

Exp: set = [1, 2, 3]

it.next() → 1
it.next() → 2
it.next() → 3

initially it = NULL
Next function will point to 1st element in set, then it point to 2.

- Set never store values in Particular order
  if values 1, 2, 3
  out values 2, 3, 1

hasNext return true & false value

it.hasNext(); → if there is an element in set it will return True otherwise false.

start → It will give true
Because it hold True value.

Loop
```
while (it.hasNext()) {
    Print (it.next());
}
```

*
```
while (it.hasNext()) {
    Print (it.Next());
}
```

HashMap

- HashMap is a structure that store 2 type of information (1) key (2) value
- The senerio where we need to store 2 Pair of values we use HashMap

Exp :- we want to save Roll no of student

Roll No (key)	Names (values)
64	Neha
65	Kritika
66	Saran
78	Rohan
71	Suhail

The column which have unique value serve as key. Name serve as values. Value can be duplicate but key is always unique

Exp2 Petrol Pumps

Fuel (key)	Price (value)
CNG	70
Diesel	80
Petrol	90

- Fuel type is unique Therefore take it as key
- Price could be same ∴ Price serve as value.

```
import java.util.HashMap;
Public class Hashing
{
Public static void main(string arg[])
{
// country (key), Population (value)
HashMap <String, Integer> map = new HashMap<>();
```

String ↑ key Type, Integer ↓ value type.

// Insertion

```
map.put("India", 120);
map.put("US", 30);
map.put("China", 150);
System.out.println(map);
```

(reinsert key)

```
map.put("China", 180);
System.out.println(map);
```

- Put function is used for insertion.

o/p
{China=150, US=30
India=120}

- [HashMaps are unordered Pair that why ordering is changed in HashMap]

// In this case value of Population of China will be updated.

map.Put()

- key already exist → update value
- key doesn't exit → new Pair inserted.

// Search operation → containsKey(key value) function (lookup)

```
if map.containsKey("China")
{
System.out.println("key is Present in the map");
}
else
{
System.out.println("key is not Present in the map");
}
```

// After searching the key if we want to know about value of key we use the function get

- get()
  - key Exist → Print value of key
  - key doesn't exit → Null.
- containsKey
  - key Exist → True
  - doesn't exit → False.

System.out.println(map.get("China")); // key exist
System.out.println(map.get("Indonesia"); // key does not exist

o/p → 180 NULL

// Iteration in HashMap (Traversing)

For loop syntax for HashMap.

for(Map.Entry<String, Integer> e : Map.entrySet())

- String, Integer → Type of key
- Integer → Type of value
- e → element (variable)
- colon
- entrySet() → give set version of map.
- each pair is stored in variable e

```
for(Map.Entry<String, Integer> e:map.entrySet())
System.out.println(e.getKey());
System.out.println(e.getValue());
```

```
China
180
India
120
US
30
```
→ o/p

// another method.

```
Set<String> keys = map.keySet();
for(String key : keys)
{
System.out.println(key + " " + map.get(key));
```

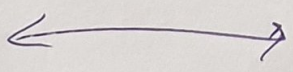
keyset.

- key → key value
- return the value corresponding to the key in map like for India key corresponding value is 120.

// Remove pair in map

```
map.remove("China");
System.out.println(map);
```

→ Remove both key & value.



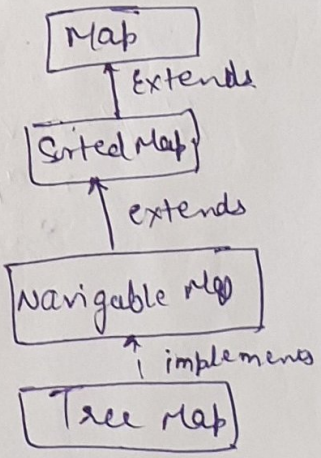
Tree Map

Tree set
Java Map Interface

Tree Map

① Java TreeMap class is a tree Based Implementation,
② Provide mean of sorting key-value Pair
Red- Black → Algorithm (self Balancing Bineary search Tree)
- color of every nod in the Tree is either Red or Black.
- Root Node must be Black in color
- Red node cannot have Red color neighbour node
- All the Paths from root node to null should consist the same no. of Black nodes

③ Java TreeMap contains only unique elements
④ Java Tree Map maintains ascending order..
⑤ Tree Map is a class of collection framework



import java.util.TreeMap ← Package / class

(6) // Creation

Syntax TreeMap<Key, Value> numbers = new TreeMap<>();

Exp TreeMap<String, Integer> evenNumber = new TreeMap<>();

(7) // Insertion → Put() → Insert specific key/value
PutAll() → Insert all entries from Map
Put if Absent() → Insert if specified key is not present in map.

Exp
```
en.Put("Two", 2);
en.Put("Four", 4);
en.Put if Absent("six", 6)
System.out.Println("TreeMap of even No." + en);
```
→ O/P: {Two=2, four=4, six=6}

(Variable Store all values) → numbers.PutAll(en)

System out.Println("TreeMap of numbers" + numbers);

(8) // Access TreeMap elements → ① entrySet(); → Return set of all keys & values
② keySet() → Return set of all keys
③ values() → Return set of all maps of Tree

```
System.out.Println("Key/value Mappings:" en.entrySet());
System.out.Println("keys:" + en.keySet());
System.out.Println("values": en.values());
```

O/P:- Key/value Mappings = [Two=2, four=4, six=6]

keys = [Two, four, six]

values = [2, 4, 6]

(9) get() → (1) value associated with key
(2) Return null if key not present

getOrDefault() – Return value associated with key
– Return specified value as default

```
int value1 = en.get("Two")
System.out.println("using get():" + value1);
int value2 = en.getOrDefault("five", 5);
System.out.println("using getOrDefault():" + value2.
```

[o/p using get():3]
[o/p using getOrDefault():5]

(10) Remove TreeMap Elements – remove(key) ← Remove value associated with key
– remove(key, value).

```
int value = en.remove("Two");
```

Print → value

```
boolean result = en.remov("six", 6).
```

Print → Result → True / false.

(11) Methods for Navigation.

firstKey() → Return the 1st key of Map
firstEntry() → Return the key/value mapping of the first key of map
lastKey() → Return last key of Map
lastEntry() → Return the key/value mapping of Last key.

⟷

TreeSet

- It implement SET Interface
- use Tree for storage
- Contain only unique elements
- does not allow null element
- Treeset maintain ascending order
- Implemented using Binary search Tree.

Syntax :- ① creating

```
Import java.util.TreeSet
TreeSet<Integer> numbers = new TreeSet<>();
```

Methods

② Insert Elements

add() → insert specified elements

addAll() → Insert all elements of specified collection

```
numbers.add(2);
numbers.add(4);
numbers.add(6);
System.out.println("Treeset:" + numbers);   o/p [2, 4, 6]
TreeSet<Integer> num = new TreeSet<>();
num.add(1);
num.addAll(numbers);
System.out.println("New set:" + num);   o/p → [1, 2, 4, 6]
```

③ Access Treeset elements → iterator() & import java.util.iterator. → hasNext() → Next()

```
Iterator<Integer> iterator = numbers.iterator();
System.out.println("Treeset using iterator");
while(iterator.hasNext()){
    System.out.println(iterator.next());
}
```

④ Remove elements → remove()
removeAll()

```
boolean val1 = numbers.remove(6);
boolean val2 = numbers.removeAll(numbers);
Print → val1 & val2
```

⑤ Navigation → first() → Return 1st element
last() → Return Last element

```
int F1 = numbers.first();
int last = numbers.last();
Print → F1 & last.
```

⟷

Java Map Interface:

- Map contains key/value Pair.
- Map is useful if you have to search, update or delete elements on the Basis of a key.
- There are Two Interface :- Map & Sorted Map
- classes → HashMap & TreeMap

⟷

Multithreading

- Multithreading
- Life cycle of Thread
- Advantages & Disadvantages of Multithreaded Applications
- How to create Thread
- Thread class
- Methods of Thread class
- Runable Interface

Multithreading:- Process of Executing multible Threads simultenously.

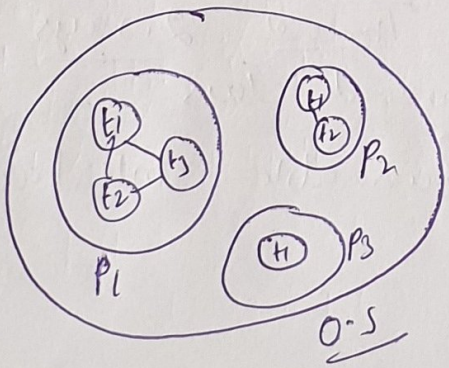
- Thread → light weight Subprocess → smallest unit of Processing
- Multiprocessing & MultiThreading → Purpose → Multitasking

- Process Based Multitasking

(a) **Multiprocessing:-** Based on no. of Processors available on the host computer. Every Process in Initiated by the user is sent to CPU (Processor). It loads the Registers on CPU with data related to the assigned Process.

(b) **Multi Threading:-** (Thread Based Multitasking) :- Thread share same memory address space.

- Each Proces has an address in memory.
- Threads are independent. If there occurs exception in one thread, it does not affect other thread.



advantages of Multithreading

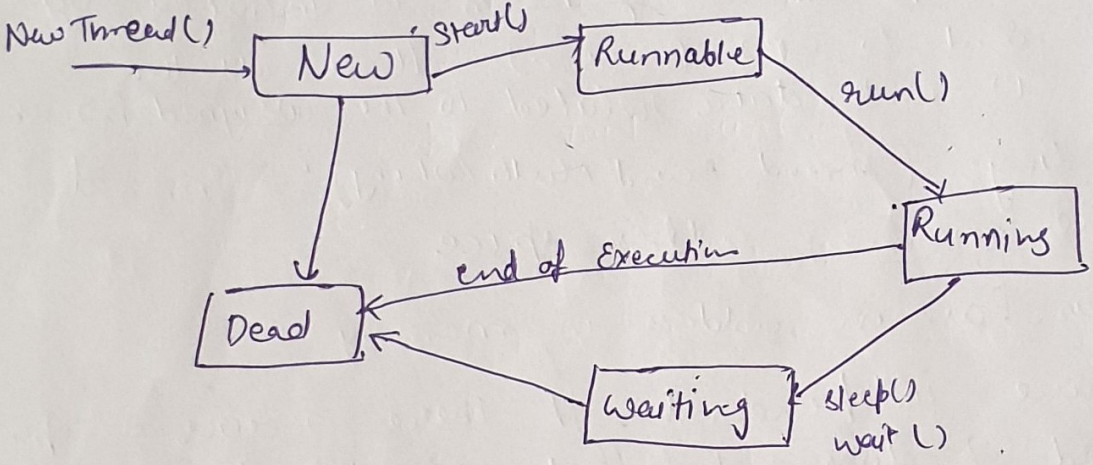
1. It does not Block user → as Threads are Independent
2. can Perform many operations together
3. If exception occurs in one thread it does not effect other thread.

disadvantages

1. Difficult of writing code
2. Difficult of debugging
3. Difficult to manage cancurrency.
4. Difficult to testing.
5. Difficult of Porting Existing Code

~~How to Create Thread~~

Life cycle of Thread



How to create Thread 2 ways

1. By using Thread class
2. By using Runnable Interface.

Thread class is a class which name
→ Thread class already built in java
→ Many constructors
→ Many Methods
run(); → [Overried. runnable interface Run Method]
Start();
→ Package → Java.lang

Reason

(2) In Runnable Interface there is only one method.
run();

Relation

Thread class implement Runable Interface

Thread Creation :- By using Thread class

// Step 1 :- Extend the Thread class.

class Test extends Thread.

// Step2 overried run() method. for this create run Method.

```
Public void run()
{
  // Task of method
  System.out.Println("Hello");
}
```

// create main method

```
Public Static void main(String args[])
{
```

// Step3:- Create object of Test class.

```
Test t = new Test();
```

// Step 4 :- Invoke Thread

```
t.start();  → start() invoke run().
```

* t.start() → Exception.

// if we again ~~with~~ write t.start();
it will give ~~error~~ Exception. Because Thread cannot be invoked again. Because when ~~take~~ task of Thread finish it will go in dead State.

(2) Runnable Interface.

Thread creation

class Test implements Runnable

// Step1:- Implement Runnable interface.

// Step2:- Overrided run() Method

```
Public void run()
{
  // Task of Method.
System.out.println("Hello");
```

// create main()

```
Public static void main (String arg[])
```

// Step3:- Create object of Test class

```
Test t = new Test();
```

// Step4. In runnable interface we do not have start(); Therefore we create object of Thread class → Reason is that start(); method only belongs to Thread class.

```
Thread th = new Thread(t);   // reference of Test class. [Pass Parameter in constructor]
th.start();
```

// Step5:- th.start invoke run method.

Which way is Better to create Thread class.

Ans:- By Runnable Interface ; Reason is in java There is no multiinheritance , if we use Runnable interface we can do like this

```
class A Extends B implements Runnable
{
}
```

→ Methods of Thread class

Basic Methods

1) run();
2) start();
3) currentThread(); → Provide Thread Refrence ; static Method.
4) isAlive(); → Return Boolean value → T/F

Naming methods

① getName();
② setName(String name);

? Deamon Thread : is a background service Thread which run as low priority Thread & perform background operations like garbage collection

① isDeamon()
② SetDeamon(boolean b) → if T → Deamon Thread created
→ if false → Deamon Thread not created.

Priority Related Method

① getPriority();
② setPriority(int Pr);

(5) Preventing Thread Execution Method.

(1) sleep (time) → sleep thread for specified time.
(2) yield (); → Pause currently executing executing Thread & allow other thread to execute
(3) join () → wait for a Thread to die.

(4) suspend ()
(5) Resume ()
(6) stop ()
(7) Destroy ()
→ Not used in Java Now.

(6) Interrupt Method.

(1) interrupt () → Interrupt the thread
(2) isInterrupted () → check whether thread has been interrupted
(3) interrupted () → check current thread has been interrupted.

(7) Inter thread communication method :- But these methods are not of Thread class. These are methods of object class.

(1) wait ()
(2) notify ()
(3) notifyAll ()
→ But used for Thread class

Method examples :

Test class
{
=
Public static void main (String [] args)
{
SOP ("Hello");
SOP (Thread.currentThread().getName());
SOP (Thread.currentThread().setName("main1"));
SOP (Thread.currentThread().getName());
}
}

main is default Thread in Java created by JVM

o/p
main

Change the name of main Thread →

o/p
main1

...ed Synchronization

- Synchronization
- Synchronize multiple Thread
- Thread Synchronization - Types — Mutual Exclusive (① Synchronized Method ② Block level Syn ③ Static Synchronization) / cooperation (Inter-thread comm. in java)
- Mutual Exclusive
- concept of Lock in Java
- Java Synchronized Method

- Synchronization is a technique through which we can control multiple threads or among no. of threads only one thread will enter inside the synchronized area
- The main Purpose of synchronization is to overcome the problem of multithreading when multiple threads are trying to access same resource at same time in that situation it may Provide some wrong results.

Categories → 2 (Mutual Exclusive)

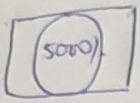
① method level Synchronization.
② Block level Synchronization.

① Method level Synchronization.

```
Public void run()
{
    // same resource
}
t1.start();
t2.start();
t3.start();
```

- Here 3 threads i.e t1, t2 & t3 are trying to access same resources; If this case It can provide wrong result.
- 3 threads invoke run method at same time.

Example



- Joint account of t_1, t_2, t_3
- when 3 Person at same time trying to withdraw 5000/- will give wrong result [corrupted Data]
- This situation contradict multithreading definition which says Every thread can access same resource at same time.

Solution.

```
public synchronized void run()
{
  // same resource
}
t1.start();
t2.start();
t3.start();
```

- when we synchronized run method → In this case only one thread can access resource at one time; Thread could be any t_1, t_2 or t_3. other ones need to wait till the completion of 1st thread.

② Block level Synchronization

```
public void hotel()
{
   synchronized(this)
   {
     // same resource
   }
}
t1.start(); t2.start(); t3.start();
```

synchronized block

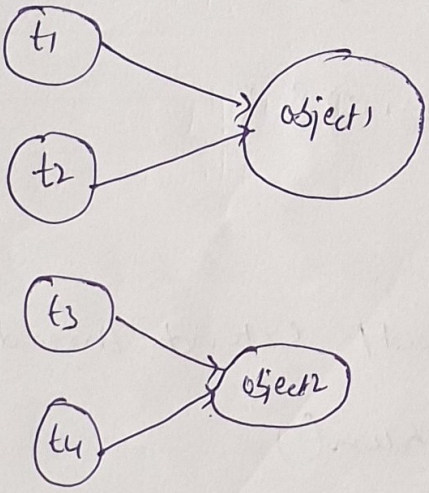
- Person who have shown can eat candy
- Candy is resource

Concept of Lock in Java

- Synchronization is built around an internal entity known as lock or monitor.
- Every object has an lock associated with it.
- By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object & release it when the thread complete its task.

Static Synchronization

- if you make any static method as Synchronized, the lock will be on the class not on Object.



In Synchronized method & block : Problem:

- t1 & t2 belong to same object so they didn't interfere with each other
- t3 & t4 do not interfere as they belong to same object i.e object2.
- But t1 & t3 can interfere
- But t2 & t4 can also interfere.

- Static Synchronization solve this problem.
- The Thread will acquire a class level lock of a java class, such that only one thread can act on the static synchronized method.

```
class Table
Synchronized static void printTable(int n)
{
 for (int i = 1; i <= 10; i++)
 {
    System.out.println(n*i);
   try {
   Thread.sleep(400);
    }
    catch (Exception e) { }
 }
 }
 }
class MyThread1 extends Thread {
Public void run()
{
 Table.printTable(1);
} }
class MyThread2 extends Thread {
Public void run()
 Table.printTable(1);
}}
```

```
Class MyThread2 extends Thread {
Public void run()
{
Table.printTable(10);
}
}
Public class TestSynchronization4
{
Public static void main (String arg[]){
    MyThread1 t1 = new MyThread1();
    MyThread t2 = new MyThread2();
    t1.start();
    t2.start();
}
}
```

Wrapper class in Java

- Primitive into object
- object into Primitive

autoboxing :- Primitive to object
unboxing → Object to Primitive

Uses

1. change value into method
2. Serialization
3. Synchronisation
4. Java util Package
5. Collection framework

Primitive Types	wrapper classes
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Primitive to wrappers

```
class Main
{
Public static void main(String args[])
{
int a=5;
double b=5.65;
// conversion
Integer aobj = Integer.valueOf(a);      //value of -> convert
Double bobj = Integer.valueOf(b);
 if (aobj instanceof Integer){
  S.O.P("object created");
 }
                                        //instance of check.
 if (bobj instanceof Double){
  S.O.P("object of Double created");
 }
}
}
```

Ex6 object into Primitive

```
class Main{
Public static void main (String[] args)
{
Integer aobj = Integer.valueof(23);
Double bobj = Double.valueof(5.55);
                                    } // create obj of wrapper class

int a = aobj.intvalue();
double b = bobj.doubleValue();
System.out.Println("The value of a" +a);
_____("_____b" +b);
}
}
```

Stream,
Character Stream
Object serialization
Cloning.

- I/O streams — Read Data
  — Write Data.

- Input stream → Read Data (source)
- Output stream → write data (Destination)
- 2 Types of streams

Byte stream (8-bits) (audio, video, char)
Character stream (16-bit unicode) (Text only)

Input Stream
Output Stream (abstract classes)

We cannot create object of abstract classes; ∴ therefore we create I/P stream from Subclasses.

Subclasses

1. BufferedInputStream
2. BufferedOutputStream
3. DataInputStream
4. DataOutputStream
5. FileInputStream
6. FileOutputStream
7. InputStream
8. OutputStream
9. PrintStream.

Most Imp Methods

read();
write();

Syntax to create InputStream

InputStream object1 = new FileInputStream();

Methods of InputStream

1. read()
2. read(byte[] array)
3. Available()
4. mark()
5. reset()
6. markSupported()
7. skip()
8. close()

Create OutputStream syntax.

OutputStream object = new FileOutputStream();

Methods of outputstream

1. write();
2. write(byte[] array);
3. flush();
4. close();

Java Character Streams

Two abstract class

/ \

Reader writer.

SubClass.

2. BufferedReader
3. BufferedWriter
4. FileReader
5. FileWriter
6. InputStreamReader
7. OutputStreamReader
8. PrintWriter
9. Reader]
10. Writer]

Create a Reader syntax

```
Reader Input = new FileReader();
```

Methods of Reader class

1. ready();
2. read(char[] array);
3. read(char[] array, int start, int length);
4. mark();
5. Reset();
6. skip();

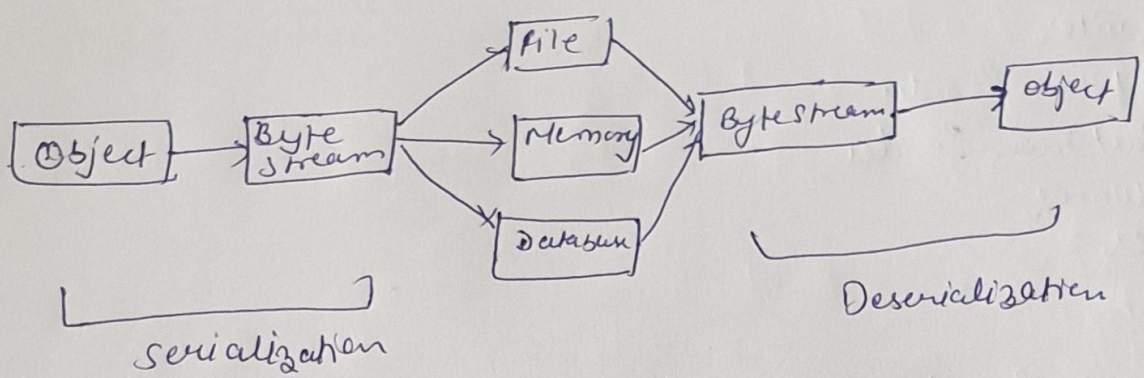
Create a Writer Syntax

```
Writer output = new FileWriter();
```

Methods of writer

1. write(char[] array);
2. write(string data);
3. append(char c)
4. flush()
5. close()

Serialization & Deserialization



- Byte stream created is Platform Independent.
- Implement Interface java.io.serializable

  class → ObjectOutputStream.] for writing (serialization)
  method → writeObject()

- for Reading

  class → ObjectInputStream] for Deserialization
  method → readObject()

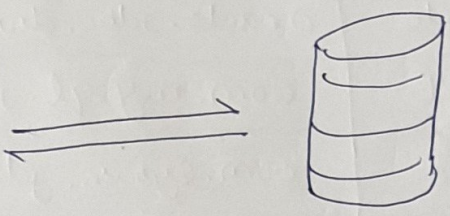
JDBC

- JDBC → Java Database connectivity
- It is a API that is used to connect and execute the queries with respective Database

e.g

```
Class dbms
{
P.S.V.m(String[] args)
{
  //code
}
}
```

Java Application



Data Base (oracle or mysql)

By using above code if we are able to create Table, insert table or update table → This is called JDBC

- Requirement to perform JDBC
  1) JDK
  2) Eclips IDE
  3) Oracle 11g
  4) .Jar file

  [we need to configure .jar file with Eclips.]

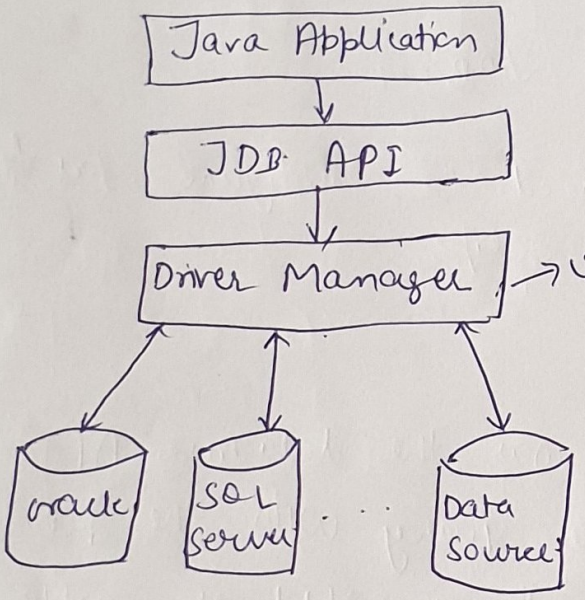
- Before JDBC, ODBC API was the database API to connect and execute the query with the database. But ODBC API use ODBC driver which is written in C language (i.e Platform dependent and unsecured). That is why java has defined its own API JDBC API that uses JDBC Drivers (written in Java language)

- JDBC more compatible with Java Applications
- JDBC use drivers to connect with Database.
- wide level of Portability ; Simple & easy to use
- Programmer need specific driver to connect to specific data base.

RDBMS	Drivers
Oracle	oracle.jdbc.driver.OracleDriver
MySQL	com.mysql.jdbc.Driver
SyBase	com.sybase.jdbc.SybDriver
SQL server	com.microsoft.jdbc.SqlServer
DB2	com.ibm.db2.jdbc.net.DB2Driver

JDBC Architecture

The main function of the JDBC to provide a standard abstraction for java applications to communicate with database.



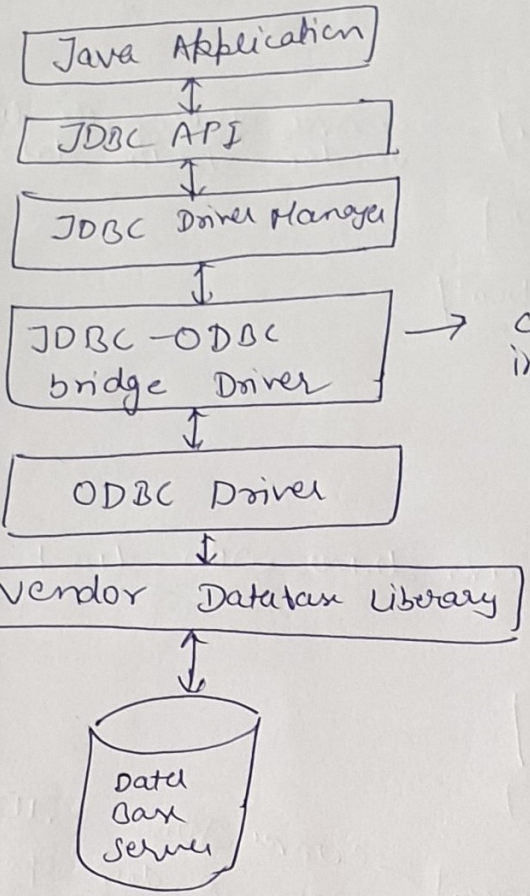
- Java Application Programed using java API
- JDBC drivers proces SQL request and generate Results

→ use specific Driver to connect with specific Database

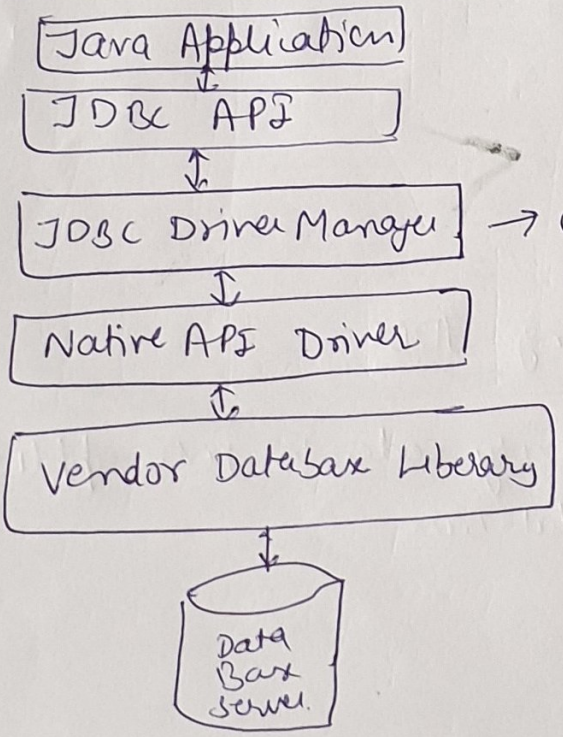
JDBC Drivers
↓
is a software component that enables java application to intract with Database.

Types of JDBC Drivers - 4.

- **Type1 Driver (JDBC - ODBC bridge Driver)**

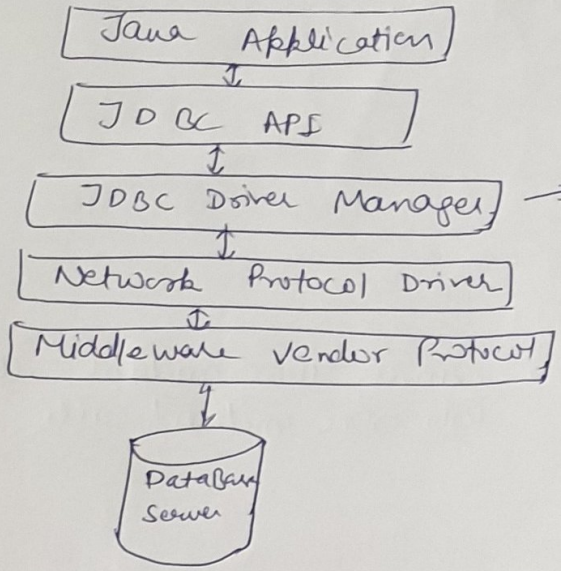


Type2 Driver (Partial JDBC Driver) (Native API drivers)



- Not written entirely in java.

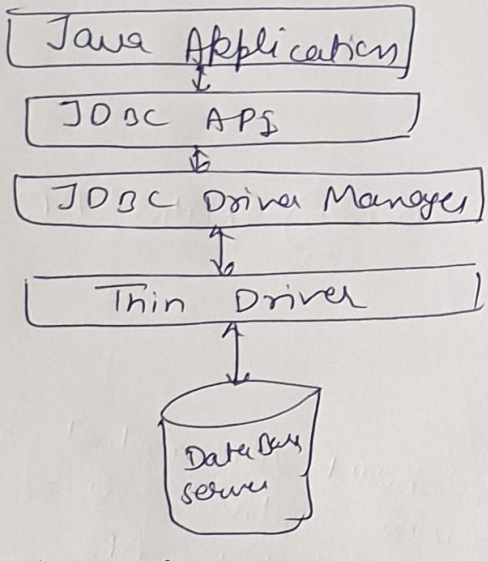
Type3 Driver (Pure java Driver for Middle ware)



- completely implemented in java

→ convert JDBC calls directly or indirectly into vendor specific database Protocol.

Type4 Driver (Pure java driver with direct database connections)



→ convert JDBC calls directly into vendor specific database Protocol.

Database programing using JDBC

→ we use JDBC connectivity code in java application to communicate with database.

→ 5 steps.

Step1:- Register the driver class
Step2:- Creating connection
Step3:- Creating statement
Step4:- Executing SQL statements
Step5:- closing connections.

Step1:- Register the Driver class

In this step, we Register the driver class with driver Manager by using forName() method of Class class

Syntax. Class.forName(Driver className)

Example:- Class.forName("oracle.jdbc.driver.OracleDriver");

Step2: Creating connection:- we can create a connection with database server by using getConnection() method of DriverManager class.

Syntax:- getConnection(String url, String name, stringPwd)

Example:-

```
Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
                                              "system", "admin");
```

(1521 → port no., xe → oracle service, system → username, admin → password, thin → Driver class)

Package:- import java.sql.Connection;
import java.sql.DriverManager;

Step3:- Creating Statement

The createStatement() method of Connection interface is used to create statement. This statement object is responsible to execute SQL statement with the database.

Syntax:- createStatement()

Example:- { Statement stmt = con.createStatement()

(Packages:- import java.sql.Statement)

To add columns

```
stmt.executeUpdate("create table emp(eno number,
                    ename varchar(12), esal number)");
System.out.println("Table created Successfully");
con.close();
}
```

Insert Record

- PreparedStatement Interface
- prepareStatement(); Pass Query Here

Step 4

After the Statement object (stmt) is created, it can be used to execute the SQL statements by using executeUpdate() or executeQuery() method of Statement Interface

executeQuery() → execute select statements

executeUpdate() → execute all SQL statements.

Syntax:- executeQuery(String query);

executeUpdate(String query);

Ex:-

```
String query = "select * from emp";
Resultset rs = stmt.executeQuery(query);
// using execute statement
String query = "insert into emp values(
                504, 'Madhu', 29);
stmt.executeUpdate(query);
```

Steps Closing Connection

close() method of connection interface

Syntax :- close()

Example: con.close();

Servlet & JSP

Servlet ① → technology → Web Application → Server side → generate dynamic webpages

② Before Servlet → CGI scripting language was used → for server side programming.
CGI → c language
Servlet → Java Language.

③ Servlet → API
→ Interface
→ class

④ Web component → deploy on server to create dynamic web pages.
Web component Respond to http requests.

Servlet API

Packages → ① Javax.servlet → contain class & interfaces → But not specific to any Protocol
② Javax.servlet.http → contain Interfaces & class that are responsible for http requests only.

Interfaces in Javax.servlet Package

① servlet ② ServletRequest ③ ServletResponse ④ ServletConfig

Class in Javax.servlet Package.

① GenericServlet ② ServletInputStream ③ ServletOutputStream
④ ServletRequestWrapper ⑤ ServletResponseWrapper ---

Interfaces in javax.servlet.http Package

① HttpServletRequest ② HttpServletResponse ③ HttpSession
④ HttpSessionListener ——

Classes in javax.servlet.http Package

① HttpServlet ② Cookie ③ HttpServletRequestWrapper
④ HttpServletResponseWrapper ——

Servlet Interface

- Define methods that all servlet must implement.
- Servlet Interface needs to be implement to create servlet.
- 3 life cycle Method
  2 Non Life cycle Method] → ⑤
- 3 Life cycle Method → init, service & Destroy
  → invoked by web container.

Methods

① public void init(ServletConfig config) → initilize servlet
→ invoke only once.

② public void service(ServletRequest request, ServletResponse response)
→ Response for incoming Requests.

③ public void destroy() → invoke only once
destroy servlet.

④ public ServletConfig getServletConfig() → Return object of ServletConfig

(5) public String getServletInfo() → Info about servlet - writer, ckeyright, version etc. (3)

GenericServlet Class:

- Implement Servlet & ServletConfig interfaces
- Protocol Independent
- Not implement service method.
- It is a class in javax.servlet package (javax.servlet.GenericServlet)

- Methods of → javax.servlet.GenericServlet

(1) void init(ServletConfig) → Initilize servlet object
(2) abstract void service(ServletRequest, ServletResponse) → Put servlet in service
(3) void destroy() → destroy servlet object
(4) String getServletInfo() → get servlet associated info.
(5) ServletConfig getServletConfig() → get servletConfig object
(6) String getInitParameter(String str) → Return value of parameter named string.
(7) Enumeration getInitParameterNames() → Return all parameter Names associated with servlet
(8) ServletContext getServletContext() → object of ServletContext.
(9) String getServletName() → Return name of this servlet object.

- Generic servlet class is abstract class → whenever any class implement Interface that class is abstract class

HttpServlet class in Java

- Package → javax.servlet.http.HTTPServlet
- Class must extend HttpServlet class & override at least one of its methods (doGet, doPost, doDelete, doPut)
- HttpServlet class extends GenericServlet class & implement a serializable interface.

Methods

① doGet() → Handle GET Request on server side
→ automatically support HTTP HEAD request
↓
which Returns nobody in Response.

② doPost() → Handle Post req. on server side
→ allow clients to send unlimited data on web server.

③ doHead() → Handle Head request.
→ Response contain only Header but does not contain msg.

④ doPut() → Handle Put Request.
→ allow client to store into. on server

⑤ doDelet() → Handle Delete Requests
→ allow client to remove a document or web page on server side.

⑥ doOptions() :- This Handle Options Requests.
used to determin which Http method the server support & Return appropriate Header.

7. doTrace() → Handle Trace Request
→ Return Header sent with Trace request to the client so that they can be used in debugging.

8. getLastModified() → • Return time the HttpServletRequest object was last Modified
• if time → Unknown → Return -ve Number.

9. Service Method() :- dispatch client Requests.

- How to handle HTML form data with Java Servlet.

- Server option is not showing in eclipse.

Help → install new Software
Work with → http://download.eclipse.org/releases/2022-09
Next.

- Server option again not visible
- Windows → Show view → others
Search → server → open
- Server tab appears.

- How to configure Tomcat with eclipse

- XML 1.0 & 1.1
- DTD
Document Type declaration