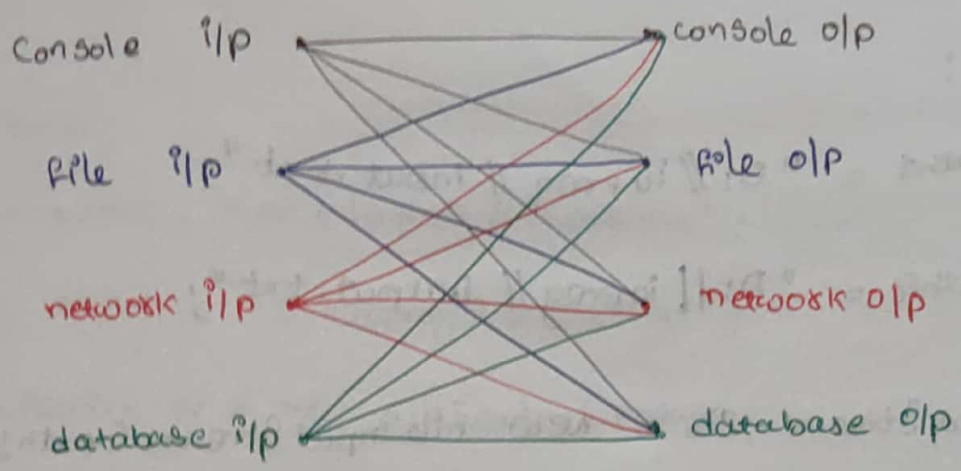
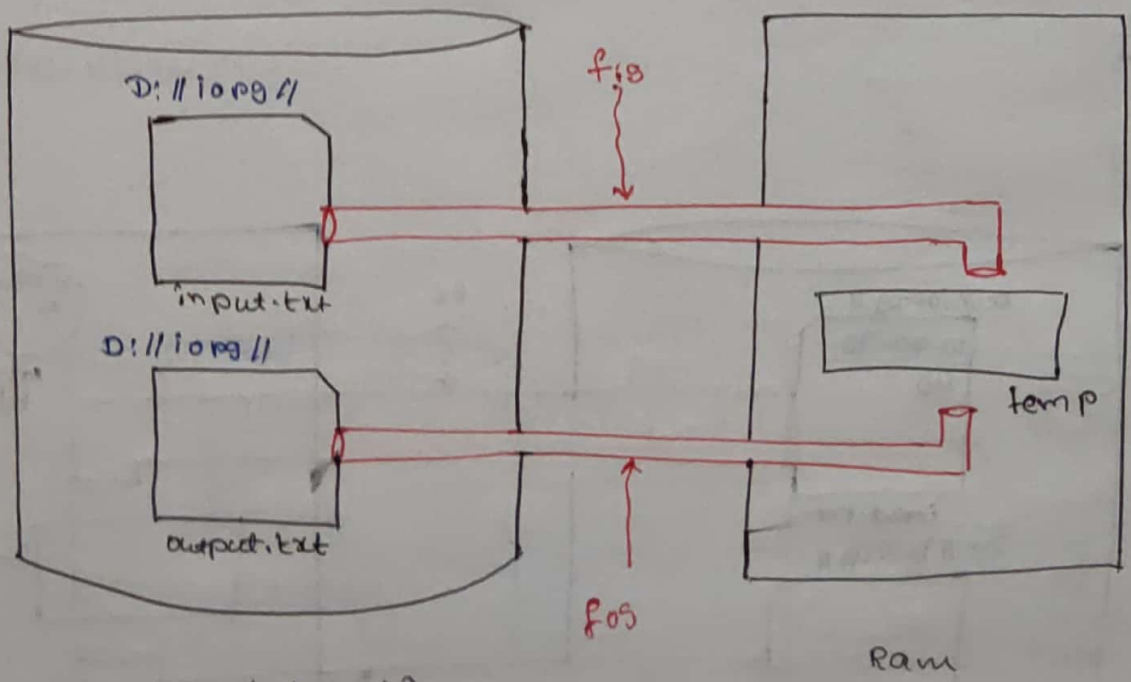
Advance Java

file handling and inbuilt streams

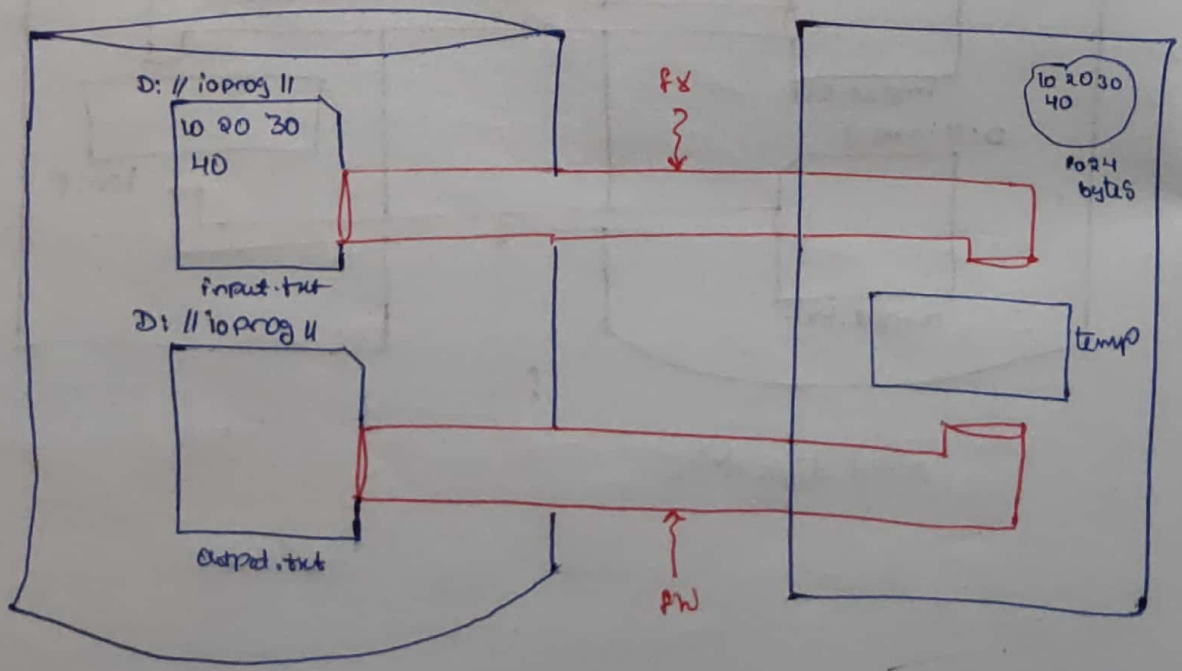


Transfering data one file to another file



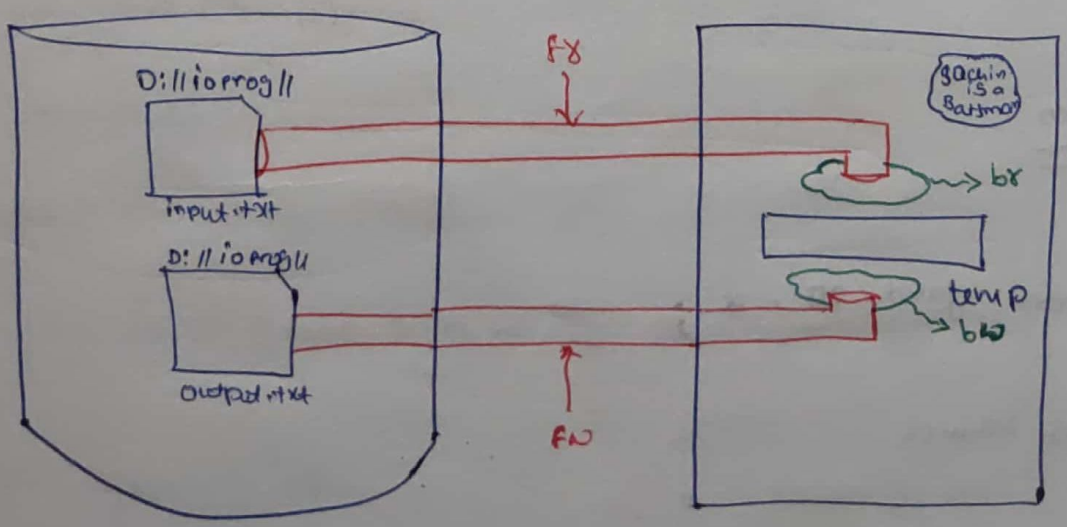
Program

```
import java.io.*;
class Launch
{
    P S V m (String args[])
    {
      int temp;
      String Path1 = "D://ioprog//input.txt";
      String Path2 = "D://ioprog//Output.txt";
      FileInputStream fis = new FileInputStream(path1);
      FileOutputStream fos = new FileOutputStream(path2);
      while ((temp = fis.read()) != -1)
      {
        fos.write(temp);
      }
    }
}
```



Program

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

class Launch
{
  p s v m (String ... args)
  {
    int temp;
    String path1 = "D://ioprog//input.txt";
    String path2 = "D://ioprog//Output.txt";
    FileReader fr = new FileReader(path1);
    FileWriter fw = new FileWriter(path2);
    while ((temp = fr.read()) != -1)
    {
      fw.write(temp);
    }
    fw.flush();
  }
}
```



Program

```
import java.io.*;
class Launch
{ p s v m (String ... args).
 { String temp = null;
  String path1 = "D://ioprog//input.txt";
  String path2 = "D://ioprog//output.txt";
  FileReader fr = new FileReader (Path1);
  BufferedReader br = new BufferedReader (fr);
  fileWriter fw = new fileWriter (path2);
  BufferedWriter bw = new BufferedWriter (fw);
  While ((temp = br.readLine()) != null)
  {
    bw.Write(temp);
  }
  bw.flush();
}
```

Program

```
import java.io.*;
class Launch
{
  p  s  v  m  (String args[])
 }
```

```
String temp = null;
String path1 = null;
String path2 = null;

FileReader fr = null;
FileWriter fw = null;

BufferedReader br = null;
BufferedWriter bw = null;

try
{
  path1 = "D:// ioprog // input.txt";
  path2 = "D:// ioprog // output.txt";
  fr = new FileReader (path1);
  br = new BufferedReader (fr);
  fw = new FileWriter (path2)
  bw = new BufferedWriter (fw);

  while ( (temp = br.readline()) != null )
  {
    bw.write (temp);
  }
} bw.flush();

catch (FileNotFoundException e)
{
    S.o.println ("Please enter the right file location");
}
catch (Exception e)
{
  S.o.println ("Some problem occured");
}
```

Finally

```
{
   try
   {
      fr.close();
      br.close();
      pw.close();
      bw.close();
   }
   catch (Exception e)
   {
      s.o.p ln ("Failed to terminate connection");
   }
}
```

Try with Resources

```
import java.util.*;
class Demo2
{
 Scanner scan1;
 public void acceptInput()
 {
  try (Scanner scan2 = new Scanner (System.in));
   {
     s.o.p ln ("Enter the first input");
     int a = scan2.nextInt();
     s.o.p ln (a);
     scan1 = scan2;
  }
```

```
catch (Exception e)
{
  s.o.pln (" some problem occured");
}
s.o.pln ("Enter the second Input");
int c= scan1.nextInt(); // ← scanner closed.
}
}
class Demo
{
p s v m (String... args)
{
    Demo2 d2 = new Demo2();
    d2.acceptInput();
  }
}
```

Program

```
class Launch
{
  p s v m (String args[])
  {
    String path = "D://ioprog//Dog.txt";
    File f= new File(path);
    if( f.exists() == true && f.isFile() == true)
    {
      s.o.pln ("path is valid and it contains File");
    }
    else
    { s.o.pln ("path is invalid & contains folder");
    }
}}
```

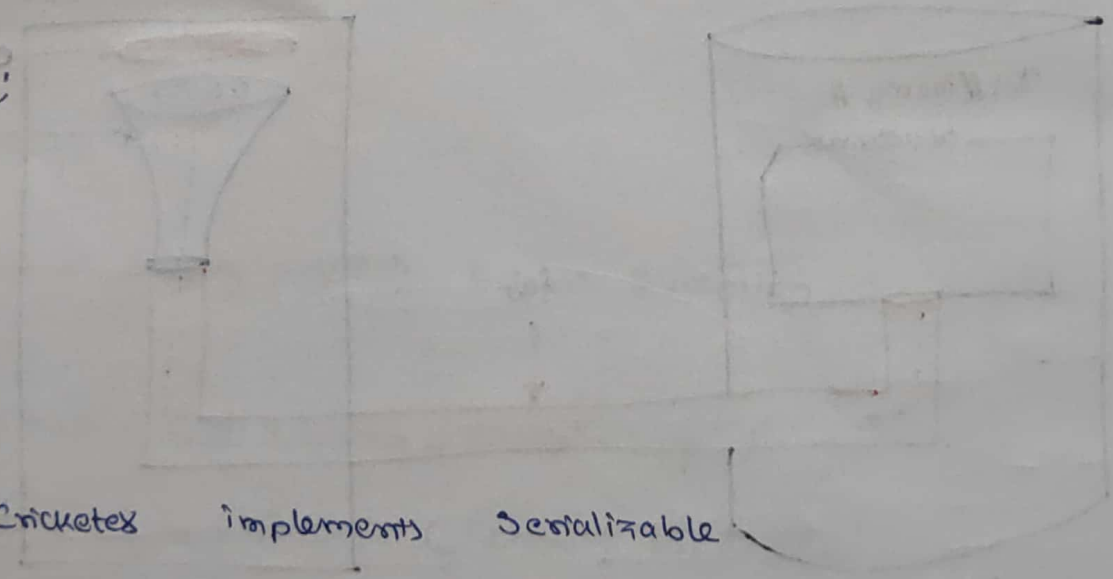
Serialization & De-serialization

→ All the programs written so far consisted of only temporary objects or non-persistant objects. becoz all the objects that were created where memory allocated on the ram and all the objects were de-allocted mesory once the program execution was complete by the garbage collector. Hence a permanent copy of objects were not created ob in other words. persistant objects were not created.

Program

```
class Cricketer
{
 String name;
 int age;
 float avg;

Public Cricketer (String name, int age, float avg)
{
   this.name = name;
   this.age = age;
   this.avg = avg;
}
```

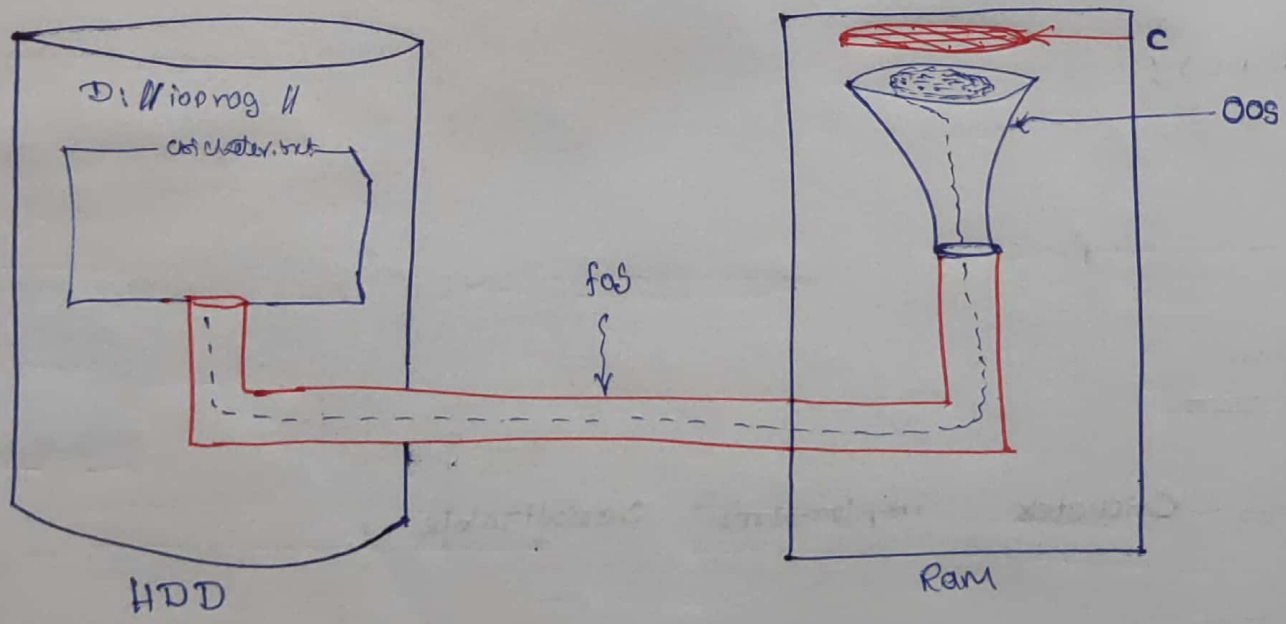
```
public void disp()
{
  s.o.p(name);
  s.o.p(age);
  s.o.p(arg);
 }
}
class Launch
{
  p s v m (String... args)
  {
  Cricketer c = new Cricketer ("Dhoni", 35, 77.5f);
  disp();
  }
}
```



Program

```
class Cricketer implements Serializable
{
  ==
  ==
  ==
  ==
 }

 class Launch
 {
   p s v m (String args[]) throws Exception
   {
```

```
String path = "D://ioprog//Cricketer.txt";
Cricketer c = new Cricketer("Dhoni", 35, 9758);
c.disp();
FileOutputStream fos = new FileOutputStream(path);
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(c);
}
}
```

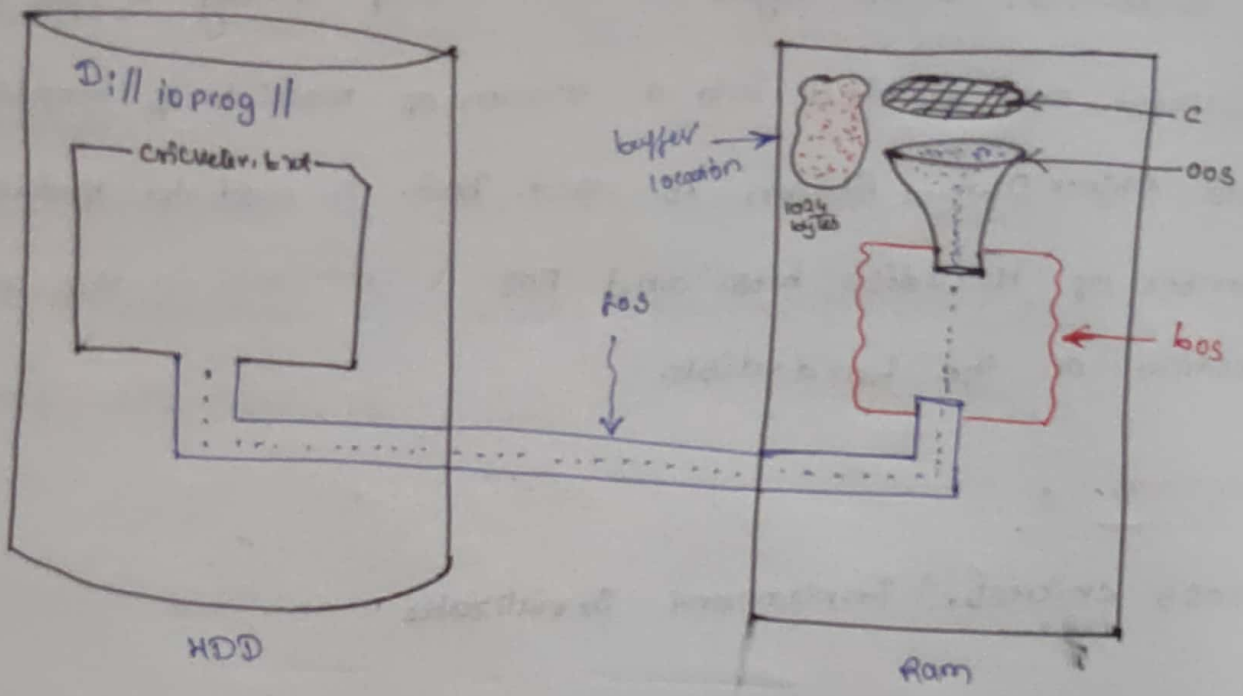


* Serialization is a process of creating a Persistant object copy of a temporary object by storing it permanently in hard-disk.

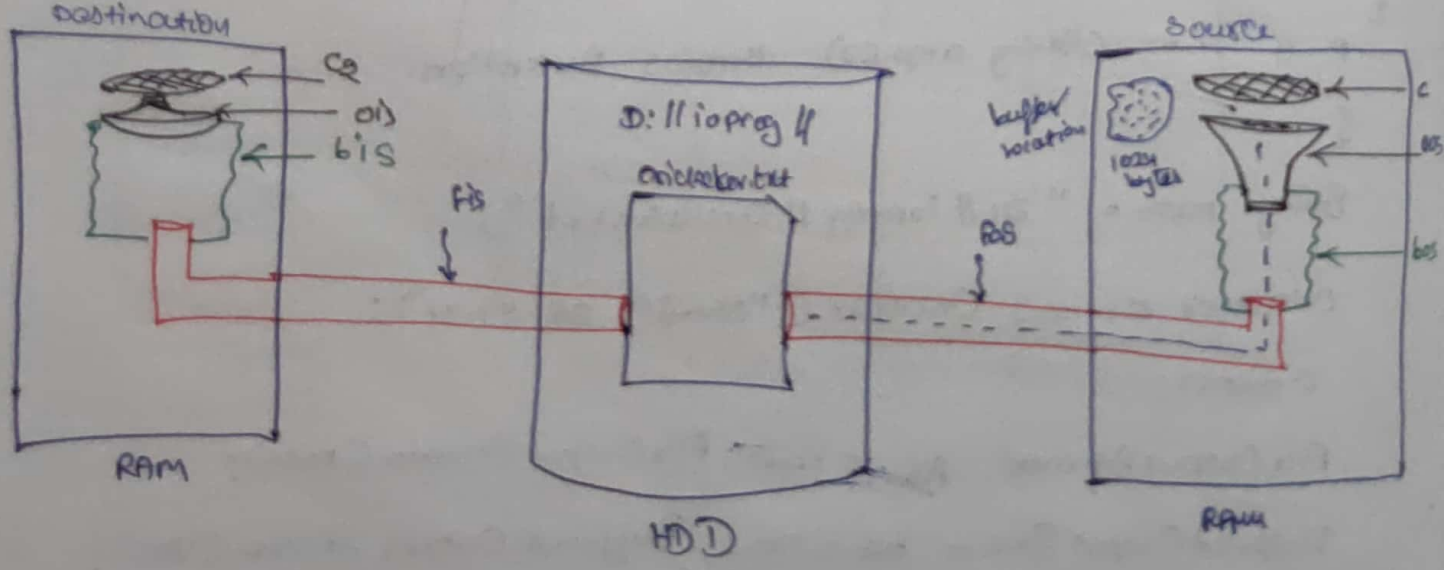
Inorder to transfer object from ram to hard-disk Streams are required. Since object is one unit of memory it has to be serialized or divided into a stream of multiple of smaller bytes. Hence Object Output Stream is used. BOS. is used to reduce the number of Harddisk hits and FOS is connected to the required location on the hard-disk.

Program

```
class cricketer implement Serializable
{
  ========
  ========
}

class Launch
{
  p s v m (String args[]) throws Exception
  {
  String path = "D:// ioprog // Cricketer.txt";
  Cricketer c = new Cricketer ("Dhoni", 35, 37.7f);
  c.disp();
  FileOutputStream fos = new FileOutputStream (path);
  BufferedOutputStream bos = new BufferedOutputStream (fos);
  ObjectOutputStream oos = new ObjectOutputStream (bos);
  oos.writeObject(c);
  bos.flush();
  }
}
```



Deserialization



Program (R)

```
class Cricketer implements Serializable
{
}
class Source
{
psvm (String args[]) throws Exception
{
String path = "D://ioprg//cricketer.txt";
Cricketer c = new Cricketer("Dhoni", 55,
                            73.5f);
c.disp();
FileOutputStream fos = new FileOutput
Stream(path);
BufferedOutputStream bos = new BuffredOutput
Stream(fos)
ObjectOutputStream oos = new ObjectOutput
Stream(bos);
oos.writeObject(c);
bos.flush();
}
}
```

(R2)

```
class Source
{
}
class Cricketer implements Serializable
{
}
class Destination
{ psvm(String args[]) throws Exception
{
Stringpath = "D://bprog//cricketer.txt";
FileInputStream fis = new FileInput
Stream(path);
BufferedInputStream bis = new
BufferedInputStream(fis);
ObjectInputStream ois = new
ObjectInputStream(bis);
Cricketer c2 = (Cricketer)ois.
    readObject();
c2.disp();
}
}
```

De-serialization :

De serialization is a process of re-constructing the serialized object present on Hard disk to its original form.

ReadObject() has a return type "Object" Hence down casting has to be performed to meet the needs of the program. Both the source and the destination should contain the class whose object is been serialized and de-serialized.

Selective Serialization & De-serialization.

There are 3 ways of achieving Selective Serialization & De-serialization.

1. To use Transient keyword.
2. Implementing Externalizable interface.
3. Implementing Serializable interface.

Transient keyword is used to perform selective serialization and De-serialization if a variable is declared as Transient then such a variable doesn't participate in Serialization & De-serialization. In other words, Transient data members are such data members which don't participate in Serialization & De-serialization.

(A)

```
class Cricketer implements Serializable
{
    transient String name;
    transient int age;
    float avg;
    ========
}
```

```
class Source
{
    ========
}
```

(B)

```
class Cricketer implements Serializ
-able
{
    transient String name;
    transient int age;
    float avg;
    ========
}
```

```
class Destination
{
    ========
}
```

Note:

During the De-serialization transient data members are initialized to default values.

Externalizable

< >

```
Public void readExternal(ObjectInput oi) throws IOException, ClassNotFound
Exception;
Public void writeExternal(ObjectOutput oo) throws IOException;
```

class Cricketer implements Externalizable.

{

String name;

int age;

Float avg;

String address;

ints runs;

int catches;

int wickets;

Public Cricketer (String name, int age, Float avg, String address, int run
int catches, int wickets)

{

this.name = name;

this.age = age;

this.avg = avg;

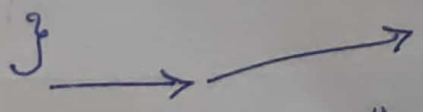
this.address = address;

this.runs = runs;

this.catches = catches;

this.wickets = wickets;

}



public void disp()

{

```
b.o.p (name);
s.o.p (age);
s.o.p (avg);
s.o.p(address);
s.o.p(runs);
s.o.p (matches);
s.o.p(wickets);
}
public void readExternal (ObjectInput io) throws IOException,
                                                  classNotFound Exception.
{
  avg = oi.readFloat();
  wickets = oi.readInt();
}
public void writeExternal (ObjectOutput oo) throws IO Exception
{
  oo.writeFloat(avg);
  oo.writeInt(wickets);
 }
}
```

Class Source

{

=

=

=

=

=

}

++ ~~Public Cricketer()~~

{

}

Class Cricketer implements Externalizable

{

=

=

=

=

=

}

Class Destination

{

=

=

=

=

=

}

→ Public Cricketer()

{

}

Note

Externalizable interface demands that atleast one zero parameter constructor has to be present inside the class which is implementing it. if it is not included de-serialization (or) re-construction of the object is not possible.

* Both selective Serialization and complete Serialization are possible by passing (or) implementing a single interface which is Serializable interface. If the no. of data members participating in Serialization is more then the no. of data members not participating in Serialization then "transient" keyword should be used.

* If the no. of variables participating in serialization is less than the no. of variables not participating in serialization then two special methods should be overridden with the respective signatures.

" Private void readObject (ObjectInputStream ois) throws IOException, ClassNotFoundException "

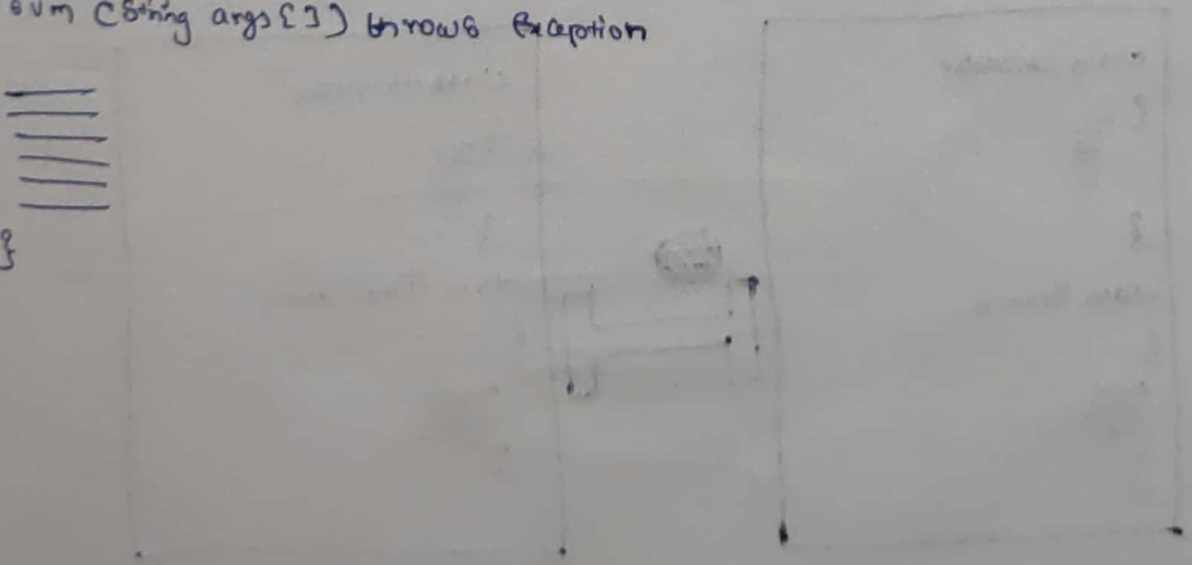
" Private void writeObject (ObjectOutputStream oos) throws IOException. "

Program

```
class Cricketer implements Serializable
{
  ===
}
Class Source
{
  ====
}
```

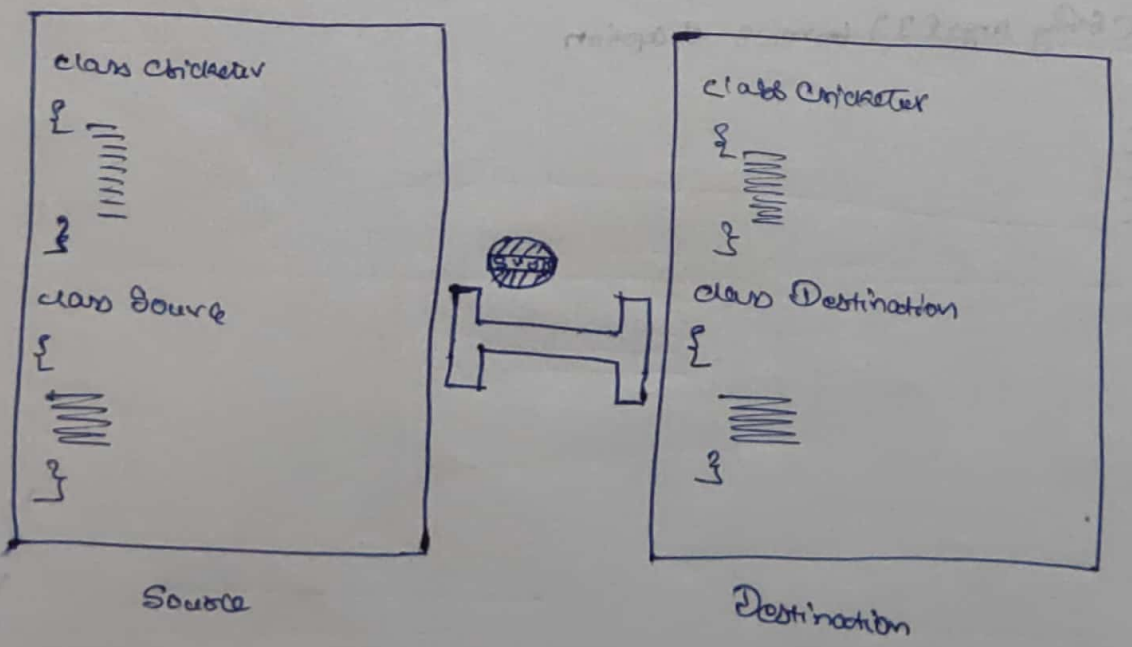
```
class cricketer implements Serializable
{
    String name;
    String addr;
    int age;
    float avg;
    int runs;
    int catches;
    int wickets;

Public Cricketer (String name, int age, float avg, String addr, int runs,
                  int catches, int wickets)
{
    this.name = name;
    this.age = age;
    this.avg = avg;
    this.addr = addr;
    this.runs = runs;
    this.catches = catches;
    this.cricketS = wickets;
}
public void disp()
{
  S.o.pln (name);
 S.o.pln (age);
S.o.pln (avg);
S.o.pln (addr);
S.o.pln (runs);
```

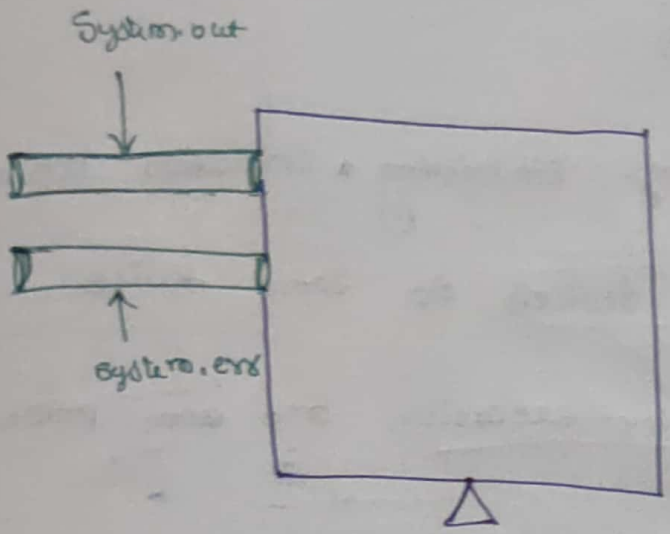
```
s.o.pln(catches);
s.o.pln(wickets);

}

private void readObject(ObjectInputStream ois) throws IOException, Class
                                                    NotFoundException.
{
  wickets = ops.readInt();
  avg = ops.readFloat();
}
private void writeObject(ObjectOutputStream oos) throws IOException.
{
  oos.writeInt(wickets);
  oos.writeFloat(avg);
}
}

class Destination
{
  public static void main(String args[]) throws Exception
  {
```



Secure Hash Algorithm:

During Serialization Secure Hash Algorithm is applied on the class whose objects are being Serialized. And a "serialVersionUID" is generated. which is stamped on the object. In the destination end during the de-serialization the serialized object is reconstructed. and the same Serial Version UID is obtained. Later Secure Hash Algorithm is applied on the same class even on the destination end if the Serial Version UID's of the object and the class present in destinations are same then de-serialization is successful. If the serial Version UID's mismatch then de-serialization is not possible.



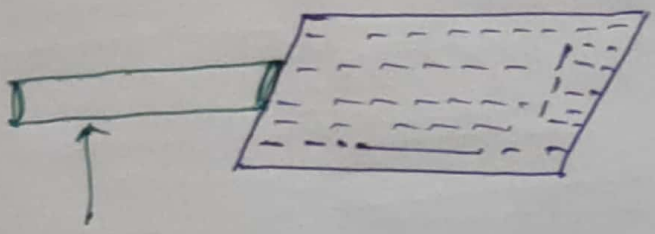


Static variable
(Print Stream)

System.out.println()

inbuilt class present in java.lang

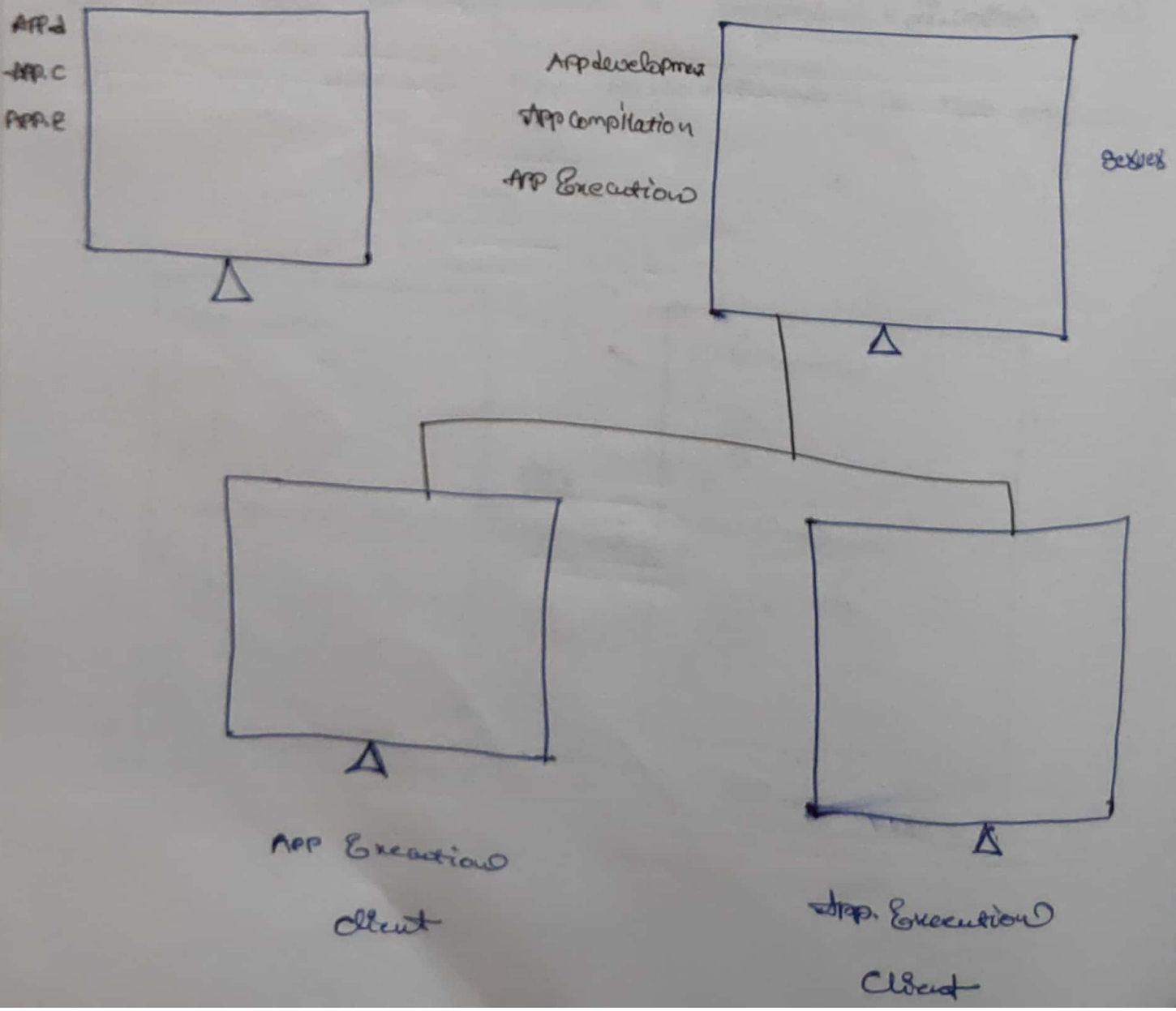
method

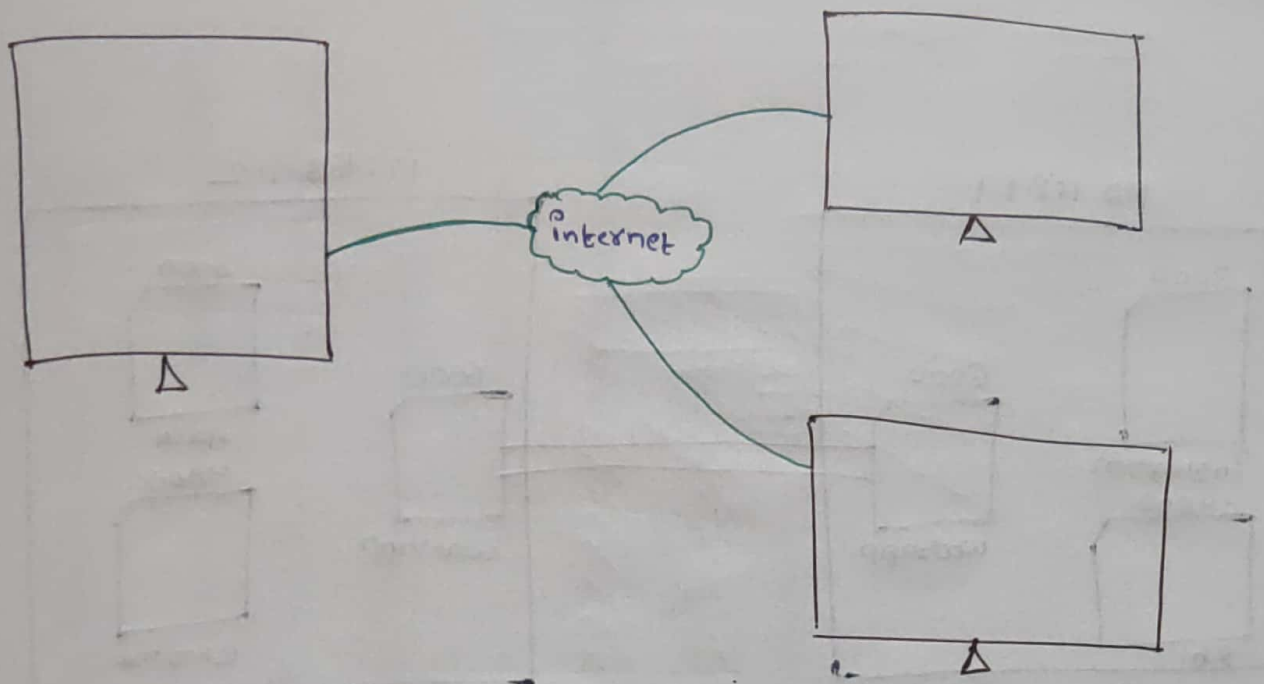


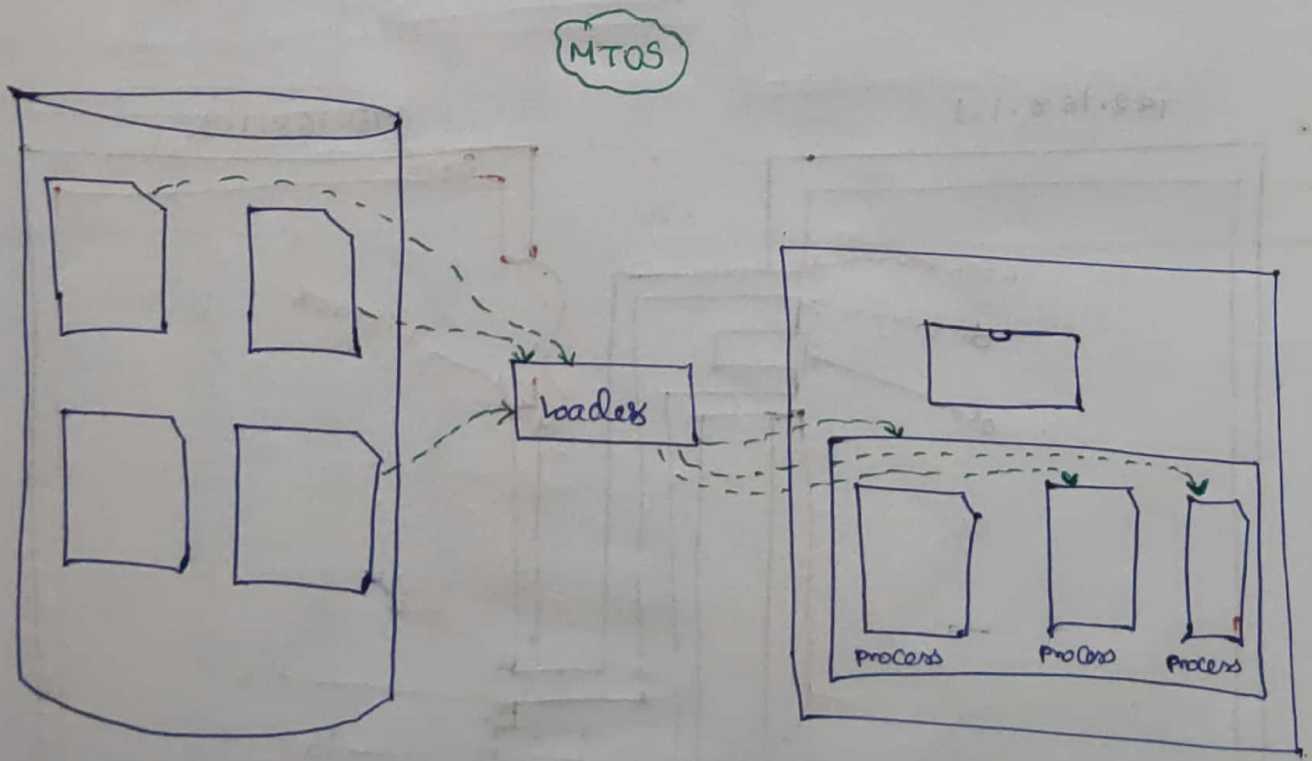
Networking

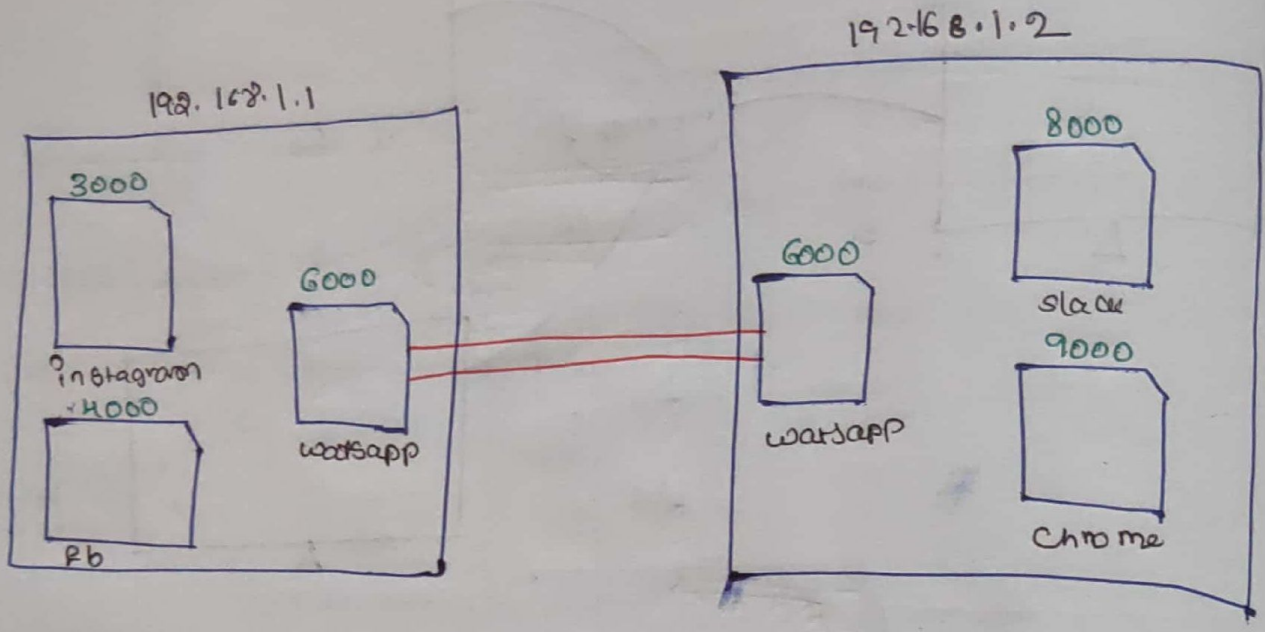
Networking is a process of establishing a connection between the processes present in different devices or same device.

Process is a program under execution and every process has a "process ID".

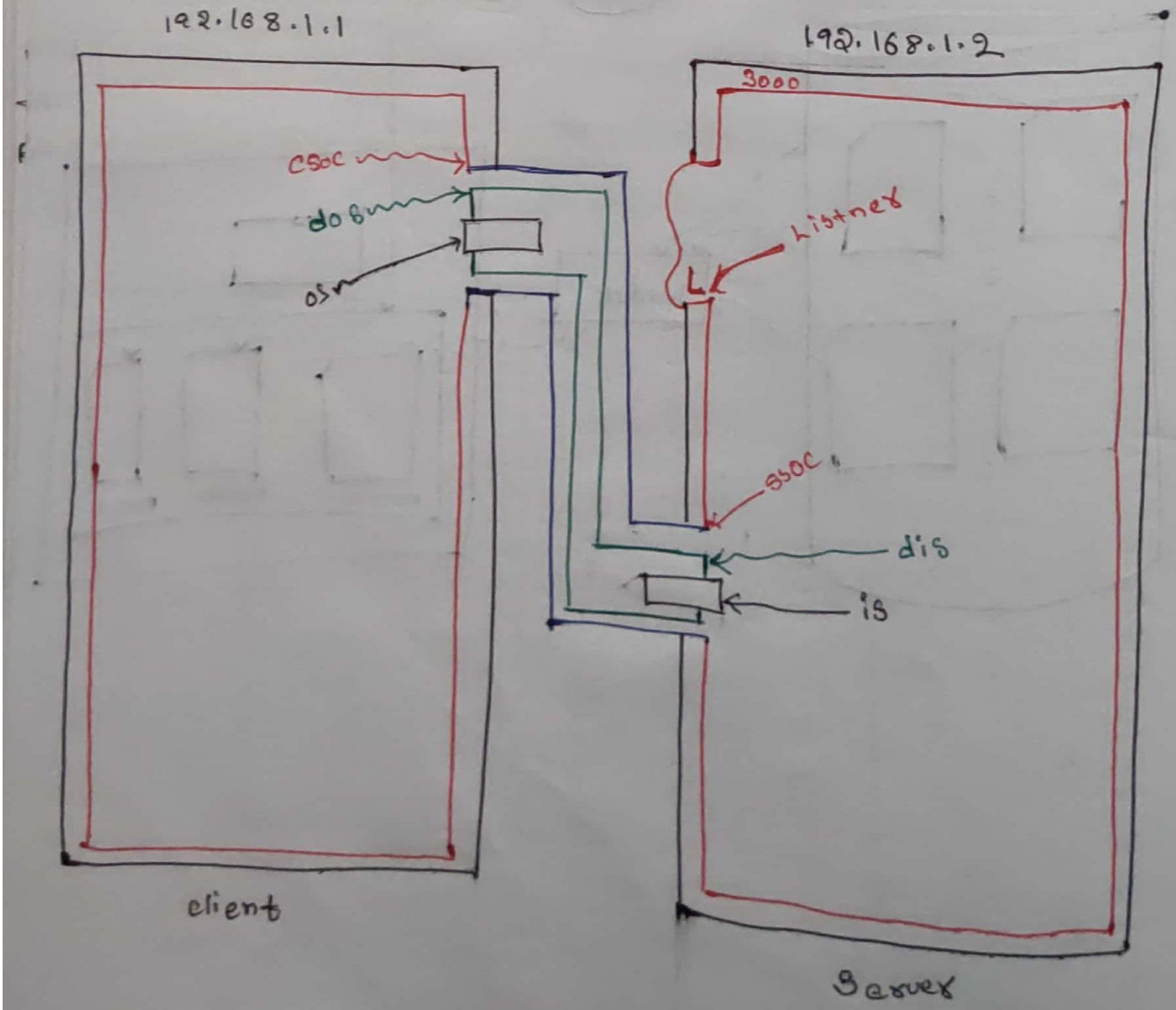








one-way communication

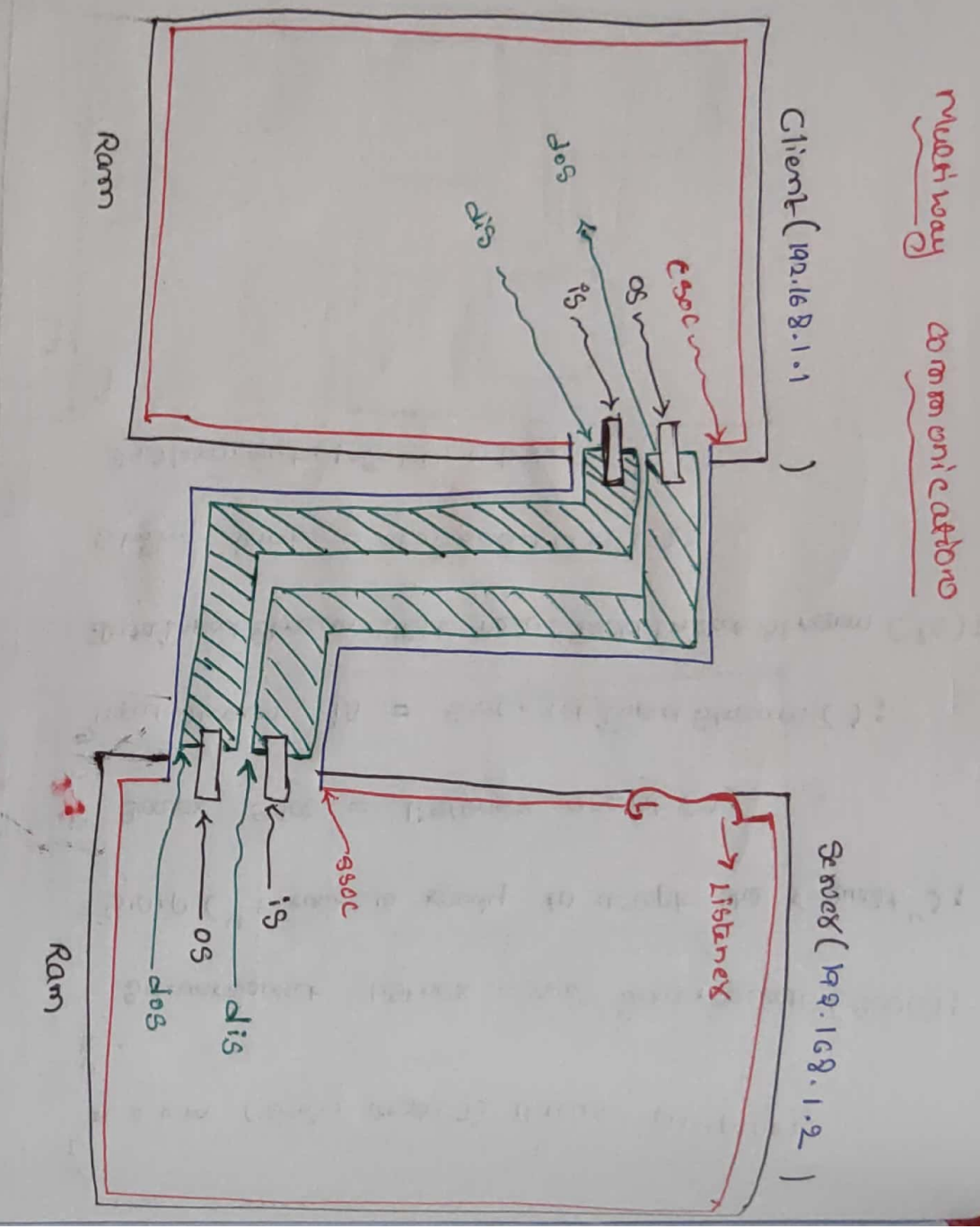


```
class Client
{
p s v m (String args[]) throws Exception
{
Socket csoc = new Socket ("192.168.1.2", 3000);
OutputStream os = csoc.getOutputStream ();
DataOutputStream dos = new DataOutputStream (os);
Scanner scan = new Scanner (System.in);
String temp = scan.nextLine ();
dos.writeUTF (temp);
}
}
```

```
class Server
{
p s v m (String args[]) throws Exception
{
ServerSocket listner = new ServerSocket (8000);
S.O.P ("Server is ready to accept the request");
Server ssoc = listener.accept ();
InputStream is = ssoc.getInputStream ();
DataInputStream dis = new DataInputStream (is);
String temp = dis.readUTF ();
System.out.println (temp);
}
}
```

```
import java.util.Scanner;
import java.io.*;
import java.net.*;
class Server
{
 ps vm (String args[]) throws Exception
 {
  ServerSocket listener = new ServerSocket (3000);
  S.o.pln (" Server is ready to accept the request ");
  Socket SSOC = listener.accept();
  InputStream IS = SSOC.getInputStream ();
  DataInputStream dis = new DataInputStream (IS);
  String temp = dis.readUTF();
   S.o.pln (temp);
  OutputStream OS = SSOC.getOutputStream ();
  DataOutputStream dos = new DataOutputStream (OS);
  Scanner scan = new Scanner (System.in);
  String temp2 = scan.nextLine ();
  dos.writeUTF(temp2);
 }
```

Multiway communications



```
* import java.io.*;
import java.util.scanner;
import java.net.*;
class client
{
 p s v m (String args[]) throws Exceptions.
 {
  Socket csoc = new Socket ("192.168.1.2", 3000);
  Output Stream os = csoc.getOutputStream (os);
  DataOutputStream dos = new DataOutputStream (os);
  Scanner scan = new scanner (System.in);
  String temp = scan.nextline();
  dos.writeUTF(temp);
  InputStream is = csoc.get InputStream ();
  DataInputStream dis = new DataInputStream (is);
  String temp2 = dis.read UTF();
  System.out.println(temp2);
 }
}
```

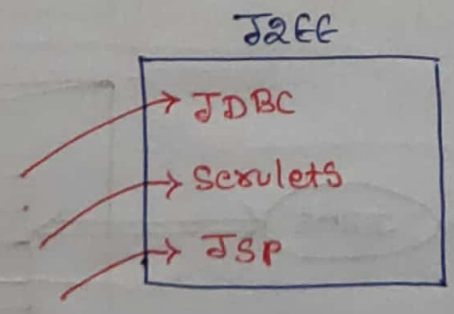
Multiway communication program - 2

⇒
```
class Client
{
  p s v m ( String args [] ) throws Exception
  {
    ========

    for (int i=0; i<=2; i++)
    {
      String temp = scan.nextLine();
      dos.writeUTF(temp);
      String temp2 = dis.readUTF();
      So.p(temp2);
    }
  }
}
```

⇒
```
class Server
{
  p s v m ( String args [] ) throws Exception.
  {
    ========

    for (int i=0; i<=2; i++)
    {
      String temp = dis.readUTF();
      S.O.p(temp);
      String temp2 = scan.nextLine();
      dos.writeUTF(temp2);
    }
  }
}
```

J2EE

→ Before the invention of internet programming languages were designed for scientific applications. After the invention of internet since computers started to reach a lot of end users a relization was made that commercial applications can also be built using programming languages. Hence, inorder to enable java programming language to built business applications new features such as JDBC, servlets, JSP were introduced. these features introduced into java are collectively known as "J2EE." (or) "Java's Enterprise Edition".



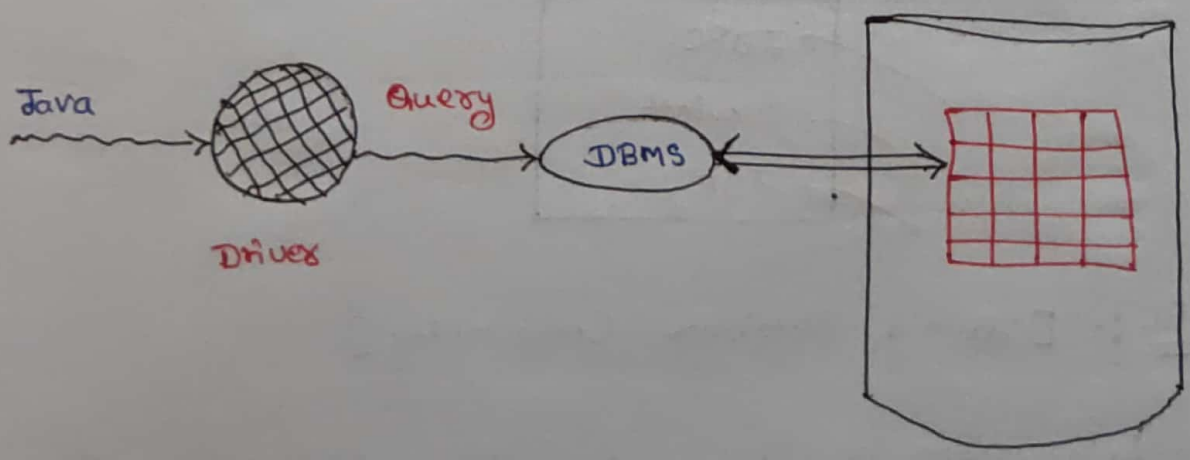
JDBC :- [Java's Database Conectivity]

→ Database which is nothing but a large collection of data is present on hard disk for permanent storage. All the interactions with the database are going to occur only

through DBMS so that security is provided to database and so that data can be stored in a formated manner in the form of rows and columns.

DBMS understands only query and query related instructions. Hence a programming language such as Java, python etc., can't interact with DBMS. directly hence drivers is used.

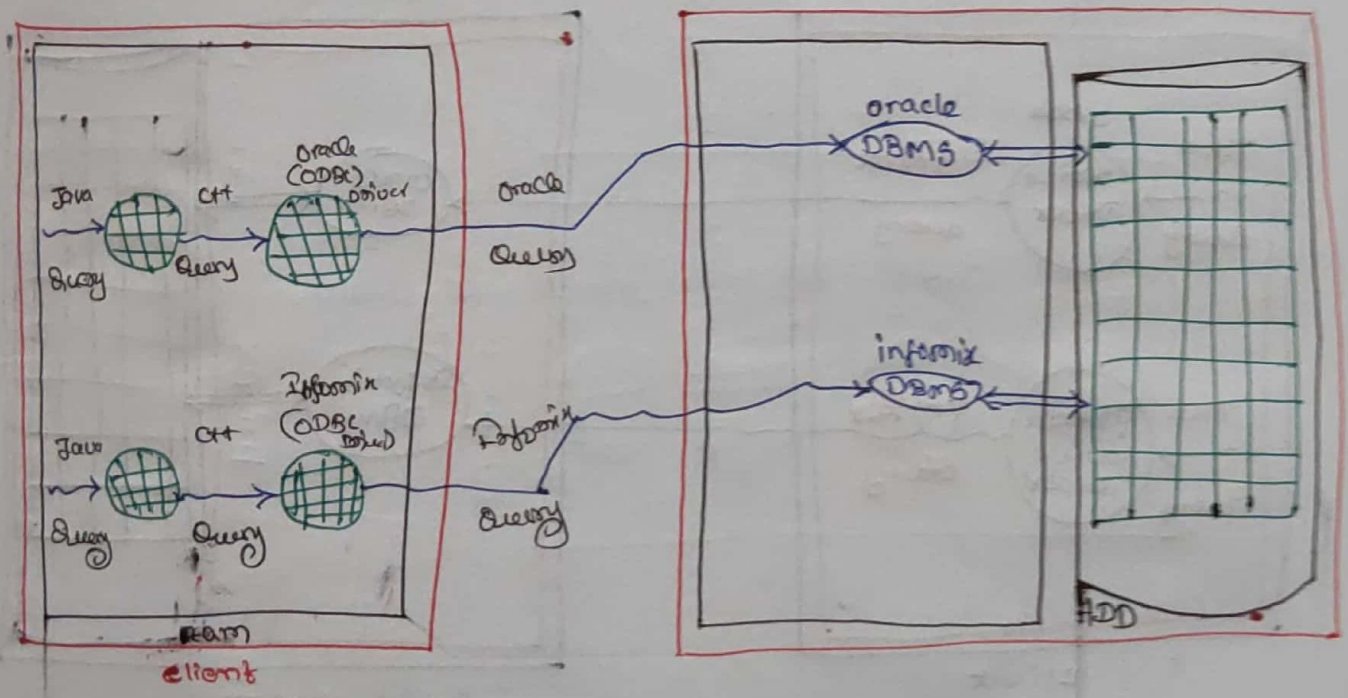
Drivers is a software which converts a programming language instructions into a query which DBMS can understand.



Type's of Drivers :-

Type - I

JDBC - ODBC Bridge



Advantage :

* For the first type Java programming language was able to connect to DBMS.

Disadvantage :

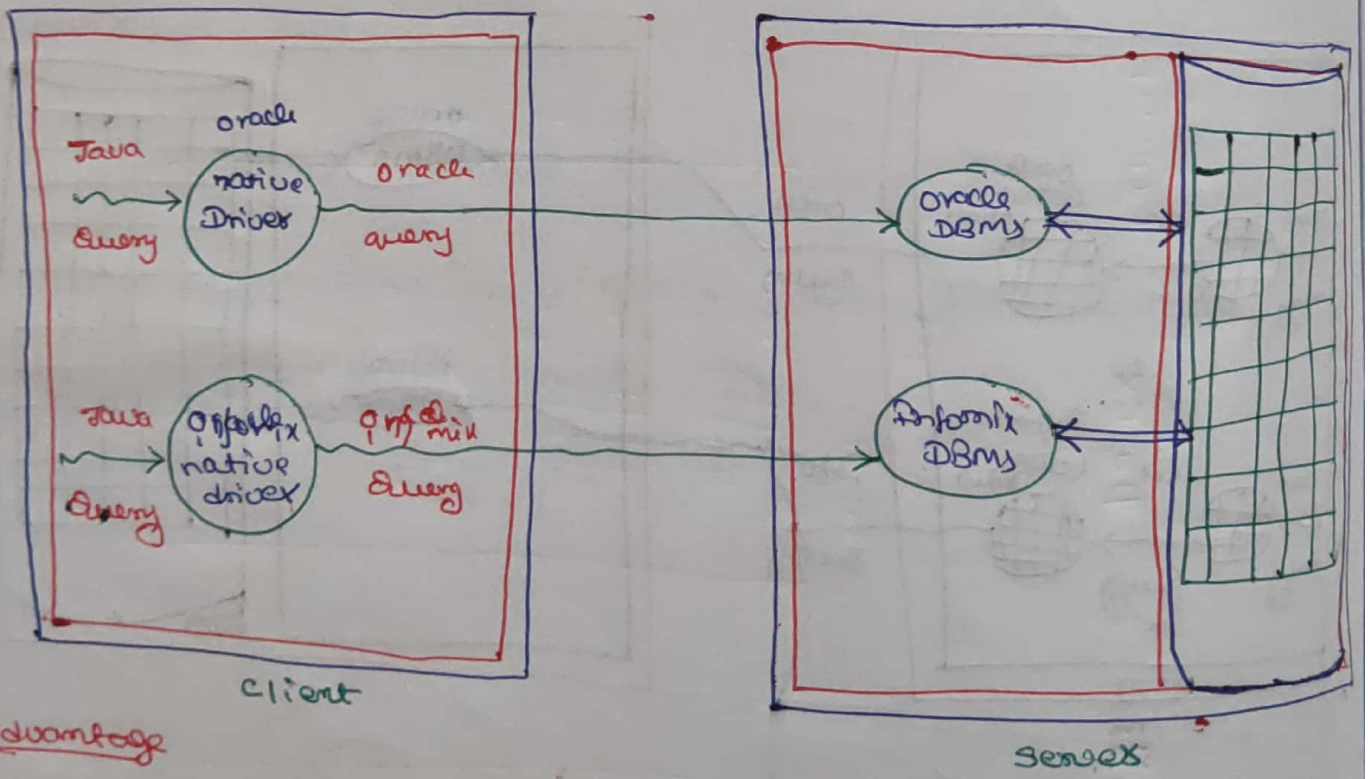
* It was not utilising the time efficiently, since the number of conversions are more.

* The operational cost involved was also more. since, ODBC driver DLL not available free of cost.

* Memory occupy on the client computer was more.

Type-2 Driver

Native Driver



Advantage

* Time is efficiently utilised, since the no. of conversions is less.

* There is no operational cost involved, since ODBC driver is not used.

* Less Ram ~~memory~~ space is occupied on client computer.

disadvantage

If a small change is done in the server w.r.t DBMS the same corresponding change has to be reflected in client computer.

Type-3 Driver

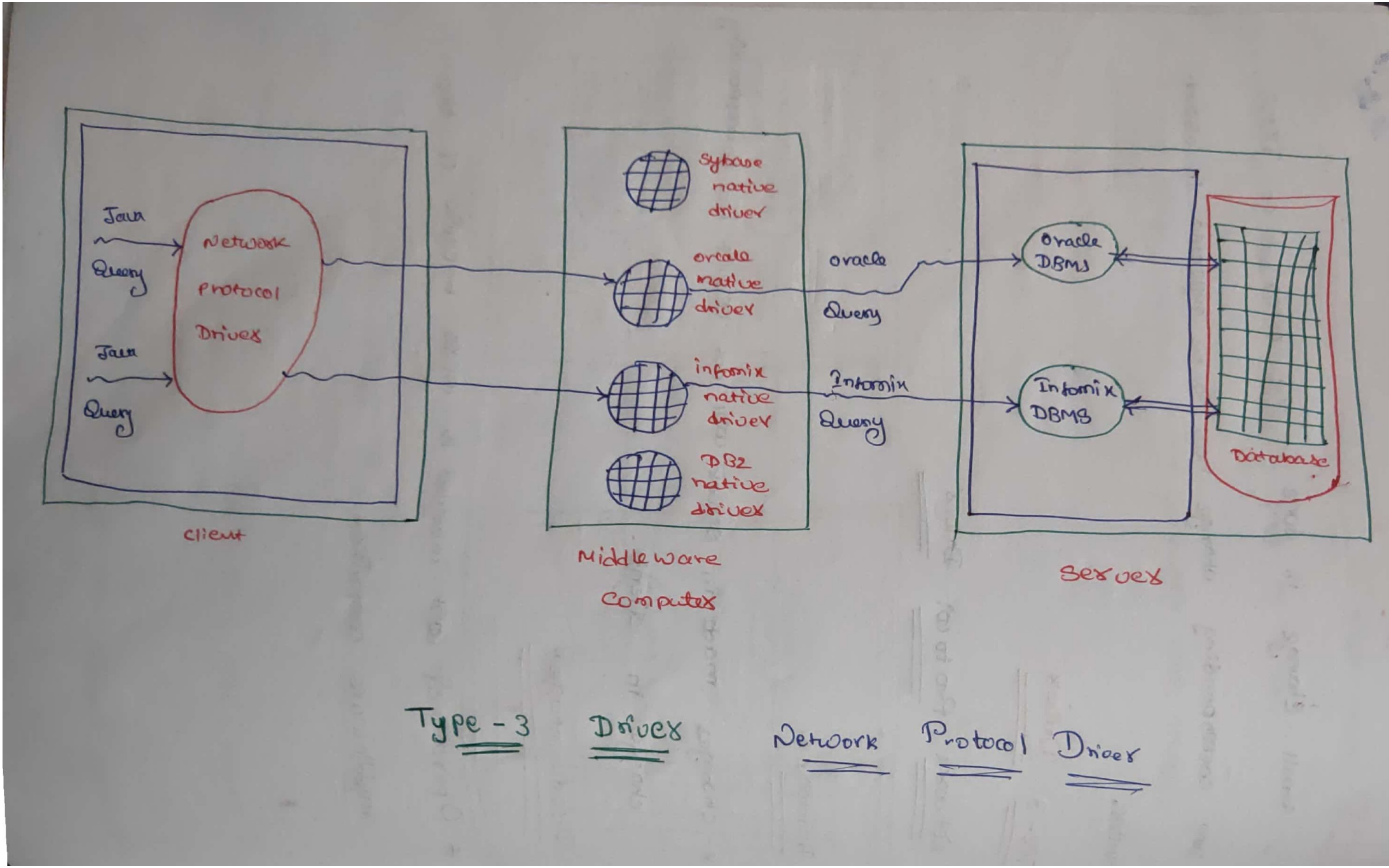
Network Protocol Drivers

Advantage :-

* changes made in server will not result into corresponding change in client.

Disadvantage :-

* Operational cost involved is more because of the middleware computer.



Type - 3 Drivers

Network Protocol Driver

Type-4 Drivers

Thin Drivers

Advantage

* Middleware computer is not involved hence operational cost is less.

Disadvantage

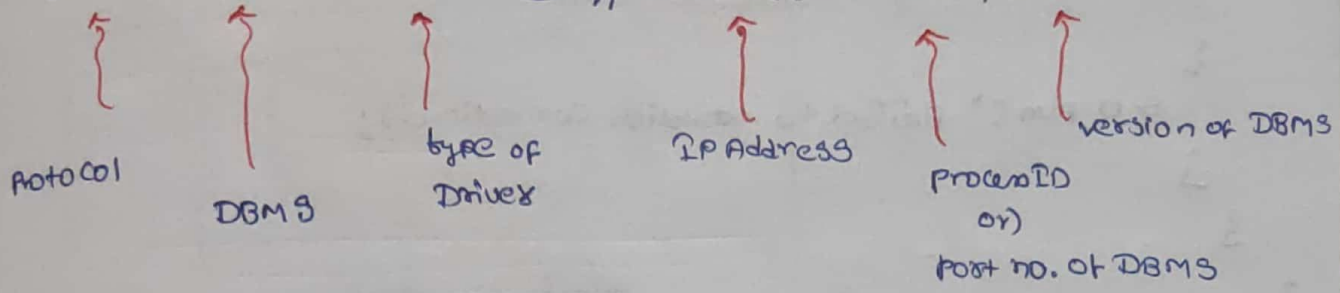
* Demands an Internet connection ("Real time Projects")

* oracle.jdbc.driver.OracleDriver *

Loading a driver :-

```
import oracle.jdbc.drivers.OracleDriver;
public class launch
{
  public static void main(String[] args)
  {
    try
    {
      DriverManager.registerDrivers(new OracleDrivers());
      S.O.println("Drivers is ready loaded successfully");
    }
    catch(SQLException e)
    {
      S.O.println("Failed to load the Drivers");
    }
  }
}
```

jdbc : oracle : thin : @ // localhost : 1521 / XE



Establishing a connection :-

```
import

public class sample
{
p s v m (String args[])
{
String url = "jdbc:oracle:thin:@//localhost:1521/xe";
String userid = "system";
String password = "system";
Connection con = null;
try
{
   DriverManager.registerDriver(new OracleDriver());
   S.O.println("Driver loaded successfully");
}
catch (SQLException e)
{
   S.O.println(" Failed to load the Drivers");
}
try
{
  Con = DriverManager.getConnection(url, userid, password);
  S.O.println(" connection is established");
```

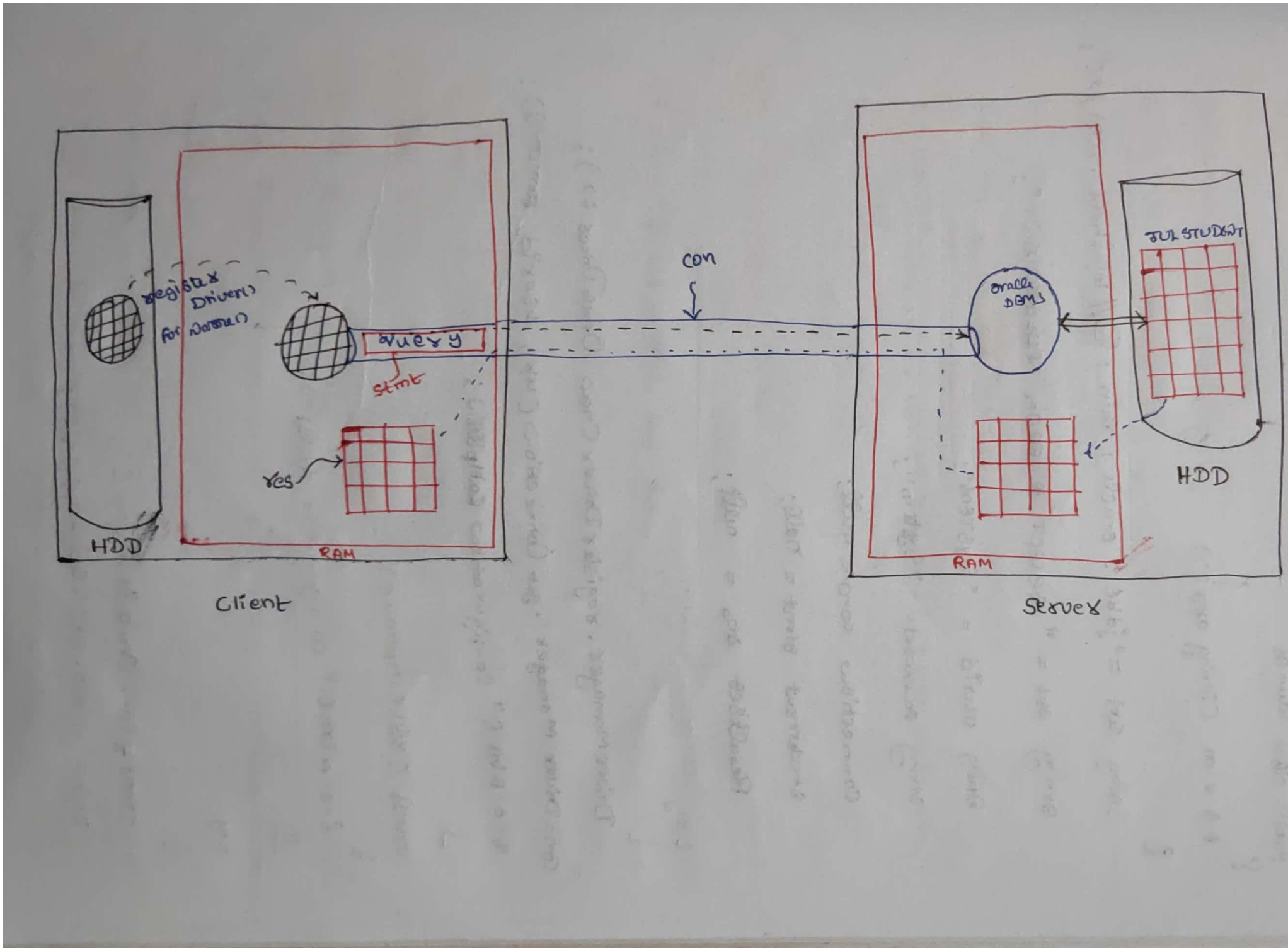
```
        }
        catch (SQLException e)
        {
          System.out.println("Failed to establish connection");
        }
    }
}
```

Accessing the Complete Table

Sample Table

JULSTUDENT

Name	USN	M1	M2	M3
Hari	1AM11CS001	100	100	100
Teja	1AM11CS002	99	99	99
Sai	1AM11CS003	88	88	88
MEGNA	1AM11CS004	35	35	35
TINA	1AM11CS005	25	25	25



Jdbc - Program - 2

```
public class Launch
{
  P S V M (String args[])
  {
    String url = "jdbc : oracle : thin : @// localhost : 1521 /XE";
    String sql = " SELECT * FROM JULSTUDENT";
    String userid = "SYSTEM";
    String password = "System";
    Connection con = null;
    Statement stmt = null;
    ResultSet res = null;
  try
  {
    DriverManager.registerDriver(new OracleDriver());
Con = DriverManager.getConnection(url, userid, password);
S.O.Pln (" Configuration Complete");
  }
catch (SQLException e)
{
  S.O.Pln (" Configuration Failed");
}
try
{
  stmt = Con.CreateStatement();
  res = stmt.executeQuery(sql);
```

```
while (res.next() == true)
{
  String name = res.getString(1);
  String usn = res.getString(2);
  int m1 = res.getInt(3);
  int m2 = res.getInt(4);
  int m3 = res.getInt(5);
  S.o.pln ("name+" "+usn+" "+m1+" "+m2+" "+m3);
  }
 }
catch (SQLException e)
{
  S.o.pln (" Failed to fetch the data ");
 }
 }
}
```

Note:-

⇒ If the Query is complete create Statement should be used.

⇒ If the Query is contained a place holder prepare Statement

⇒ should be used.

⇒ If multiple rows are return while loop should be used.

⇒ If single row is return if-else should be used.

Jdbc - Program - 3

```
Public class Launch
{
 P S V m (String ... args)
 {
  String url = "jdbc:Oracle:thin:@//localhost:1521/XE";
  String userid = "SYSTEM";
  String password = "SYSTEM";
  String sql = "SELECT * FROM FULSTUDENT WHERE USN = ?";
  Connection con = null;
  PreparedStatement pstmt = null;
  ResultSet res = null;
  try
  {
   Driver Manager. registerDriver (new OracleDriver ());
   con = DriverManager.getConnection(url, userid, password);
  }
  catch( SQLException e)
  {
    e. printStackTrace ();
  }

  try
  {
   pstmt = con. prepareStatement (sql);
   Scanner scan = new Scanner (System.in);
```

```
S.O.pln ("Enter the USN");
String usn = scan.next();
pstmt.setString(1, usn);
res = pstmt.executeQuery();

if (res.next() == true)
{
   String name = res.getString(1);
   String usn1 = res.getString(2);
   int m1 = res.getInt(3);
   int m2 = res.getInt(4);
   int m3 = res.getInt(5);
} S.O.pln (name + " " + usn1 + " " + m1 + " " + m2 + " " + m3);
 else
 {
    S.O.pln ("Invalid USN");
 }
}

catch (Exception e)
{
   e.printStackTrace();
}
}
}
```

Jdbc - DBMS update program

```
Public class Launch
{
  P S v m (String ... args)
  {
    String url = "jdbc : oracle : thin : @ // localhost : 1521 / XE ";
    String userid = "SYSTEM";
    String password = "SYSTEM";
    String sql = " UPDATE JULSTUDENT SET NAME = ? WHERE
                  USN = ? ";
    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet res = null;
  try
  {
    DriverManager. registerDriver (new OracleDriver ());
    con = DriverManager. getConnection (url, userid, password
  }
catch (SQLException e)
{
  e. printStackTrace ();
}
try
{
  pstmt = prepared con. prepared Statement (sql);
  Scanner scan = new Scanner (System. in);
```

```
        s.o.println ("Enter the Name");
String name = scan.next();
        s.o.println ("Enter the USN");
String usn = scan.next();
        pstmt.setString(1, name);
        pstmt.setString(2, usn);
        res. pstmt.executeQuery();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
}
```

Note: In the case of non-select base query such as Insert, UPDATE, DELETE. no data is returned from the database. Hence, execute update should be used.

→ if "select" sql command is used then in order to access whatever is being returned we need ResultSet. Hence, execute Query should be used.

Jdbc - DBMS. Delete Program

```
Class Launch
{
  P S V m (String args[])
  {
    String sql = " DELETE FROM JULSTUDENT WHERE USN = ?4 ;
```

{ loading Driver & establishing connection }

```
    try
    {
      pstmt = con.prepareStatement(sql);
      Scanner scan = new Scanner(System.in);
      S.o.pln("Enter the USN");
      String usn = scan.next();
      pstmt.setString(1, usn);
int row = pstmt.executeUpdate();
      if (row == 0)
      {
        S.o.pln("Deletion Failed");
      }
      else
      {
        s.o.pln("Deletion successfully");
      }
    }
```

```
catch ( Exception e)
{
   e. printStackTrace ();
}
}
}
```

Jdbc - DBMS. INSERT Program

```
class launch
{
  p s v m ( String ... args)
  {
    String sql = " Insert into JULSTUDENT values ( ? ? ? ? ? )";
```

[loading Driver & establishing Connection]

```
    try
    {
      PStmt = con. prepareStatement (sql);
      Scanner scan = new Scanner (System.in);
      S.O.println (" Enter the name ");
      String name = scan.next ();
      S.O.println (" Enter the usn");
      String usn = scan.next ();
      S.O.println (" Enter the marks");
      int m1 = scan.nextInt ();
```

```
S.O.println ("Enter the marks2");

int m2 = scan.nextInt();

S.O.println ("Enter the marks3");
int m3 = scan.nextInt();

pstmt.setString(1, name);
pstmt.setString(2, usn);
pstmt.setInt(3, m1);
pstmt.setInt(4, m2);
pstmt.setInt(5, m3);

int rows = pstmt.executeUpdate();
if (rows == 0)
{
    S.O.println ("Data Insertion Failed");
}
else
{
    S.O.println ("Insertion Successfull");
}
}
catch (Exception e)
{
    e.printStackTrace();
}
}
}
```

Transactions

A large set of operations is only known as 'Transactions'. During a transaction achieving the "ACID" properties is very important. (Atomacity, consistency, Isolation, Durability).

Atomacity is a property where achieving 0% success or 100% success. In order to achieve atomacity we make use of two methods. (i) setAutoCommit()

(ii) commit().

commit() automatically executes since, setAutoCommit is always true. Commit() ensures that all the DB operations occurs without fail and are committed to the connections.

Program

```
class launch
{
  p s v m (String ... args)
  {
   String sql = " Insert into JULSTUDENT values( ?,?,?,?,? );
```

{ loading the Drivers & establishing connection }

```
try
{
    pstmt = con.prepareStatement("sql");
    Scanner scan = new Scanner(System.in);
    con.setAutoCommit(false);
    for(int i=0; i<=2; i++)
    {
        S.o.println("Enter the name");
        String name = scan.next();
        S.o.println("Enter the USN");
        String usn = scan.next();
        S.o.println("Enter the marks1");
        int m1 = scan.nextInt();
        S.o.println("Enter the marks2");
        int m2 = scan.nextInt();
        S.o.println("Enter the marks3");
        int m3 = scan.nextInt();
        pstmt.setString(1, name);
        pstmt.setString(2, usn);
        pstmt.setInt(3, m1);
        pstmt.setInt(4, m2);
        pstmt.setInt(5, m3);
        row = pstmt.executeUpdate();
    }
```

```
con.commit();
if(row==0)
{
    S.o.println("Insertion failed");
}
else
{
  S.o.println("Insertion successfull");
}
}
catch(Exception e)
{
  e.printStackTrace();
}
}
}
```

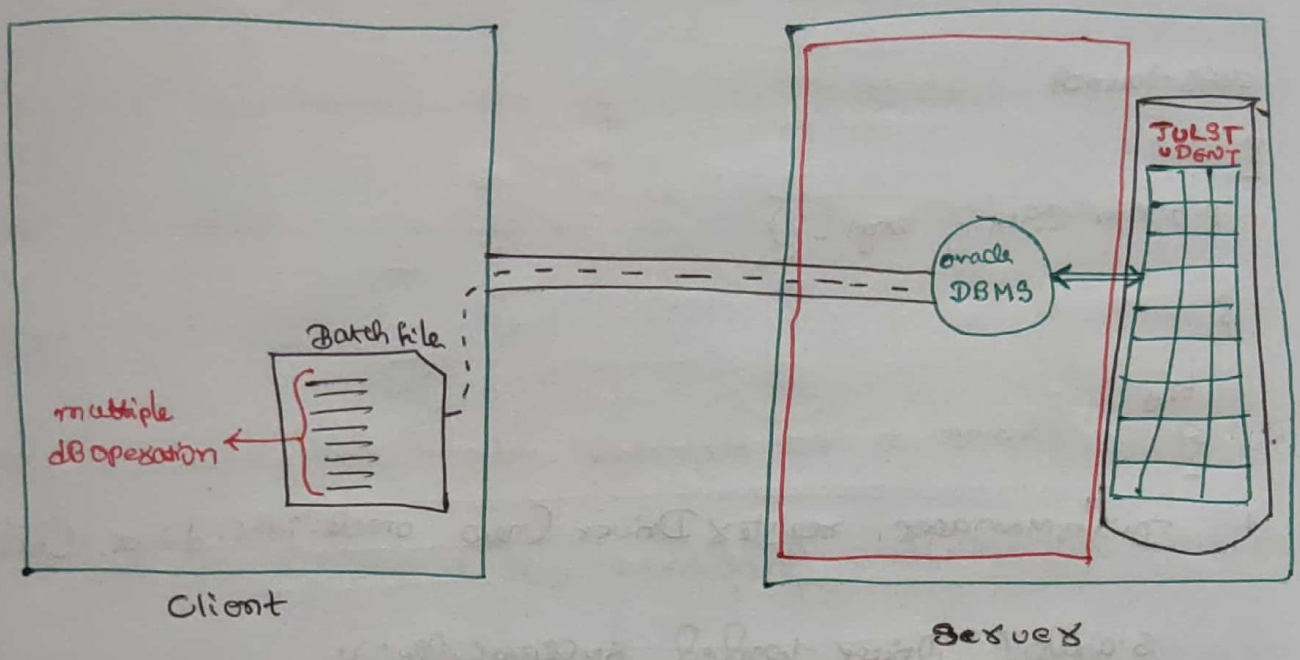
Batch update :-

→ Inorder to reduce the number of client-server interactions and provide the service better and faster a batch file is used.

→ Multiple queries are not sent to the DBMS one after the other rather multiple queries are put into a single batch file and the batch file is sent so that all the db operations occured once through a single interaction.

Program [Update]

```
Class Launch
{
  psvm (String args [])
  {
    ______
    ______
    ______     Driver is loaded
    ______           &
    ______     connection is established
    ______
    ______
try
{
  pstmt . con. prepareStatement (sql);
  Scanner scan = new Scanner (System.in);
  for (int i=0; i<=2 ; i++)
  {
    S.o.pln (" Enter the new name");
    String name = scan.next( );
    S.o.pln (" Enter the usn");
    String usn = scan. next( );
    PStmt . setString (1, name);
    Pstmt . setString (2, usn);
    PStmt . addBatch ( );
  }
  PStmt. execute Batch ( );
}
Catch ( SQLException e)
{
  e. printStackTrace ( );
}
}}
```



Other ways of loading Drivers

```
class Launch
{
  psvm (String args[])
  {
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      S.O.println("Driver loaded successfully");
    }
    catch (classNOTFoundException e)
    {
      e.printStackTrace();
    }
  }
}
```

Program - 2 {Driver} Loading

```
class launch
{
  psvm (String args [])
  {
    try
    {
      DriverManager. registerDriver (new oracle.jdbc.driver.OracleDriver());
      S.O.println ("Driver loaded successfully");
    }
    catch (SQLException e)
    {
      e.printStackTrace();
    }
  }
}
```