UNIT: 2

Dashrath Nandan

☆ Collection

Collection in java is a framework that provides an architecture to store and manipulate the group of objects.

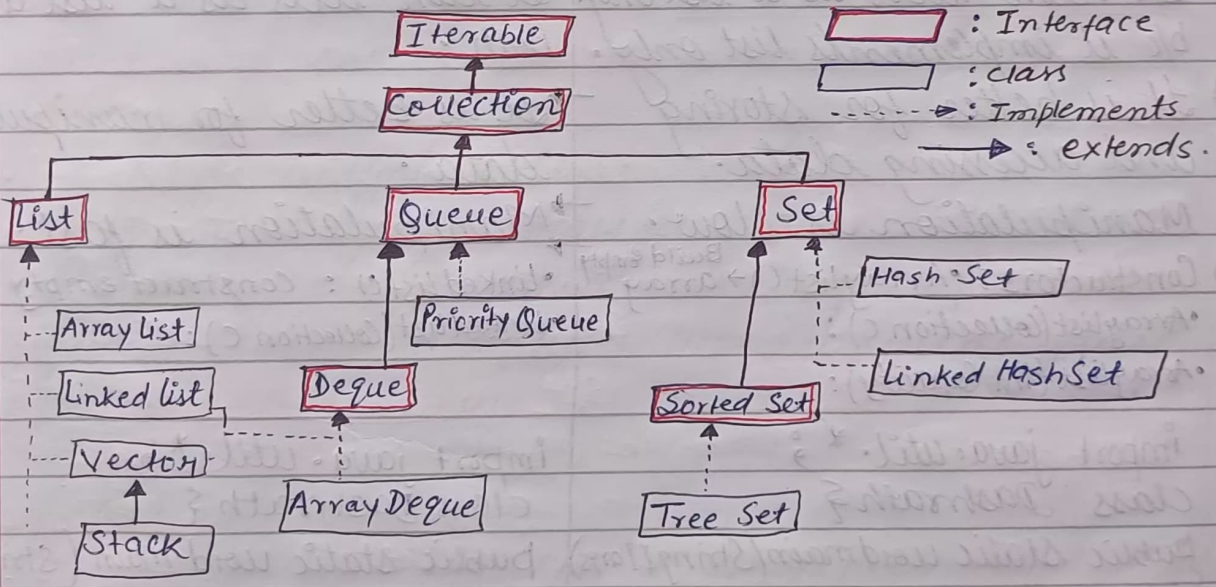
Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

(i) Interface and its implementations, i.e. classes.

(ii) Algorithm.

* Hierarchy:

The java.util package contains all classes and interface for the collection framework.



* Methods of Collection Interface:

- Public boolean add(Ee): insert an element in this collection.
- Public boolean remove(Obj. Ele): delete an element from the collection.
- public int size(): return the total number of element
- public boolean isEmpty(): Checks if collection is empty.
- public boolean equals(obj. Ele): It matches two collection

* Iterator interface:

Iterator interface provides the facility of iterating the elements in a forward direction only. Three methods are

i) Public boolean hasNext(): Return true if iterator has more elements

ii) public Object next(): return the element and move cursor pointer to Next.

iii) public void remove(): remove last element returned by iterator.

☆ Array List vs Linked List

	Array List	Linked List
i)	Diagram in Hierarchy	----------
ii)	The arraylist implements the list interface.	→ Linkedlist implements the collection interface.
iii)	It uses dynamic array to store the elements.	→ It uses doubly linked list to store the element.
iv)	It can acts as a list only. b/c it implements list only.	→ It can acts as a list and queue both.
v)	It is better for storing and accessing data.	→ It is better for manipulating data.
vi)	Manipulation is slow.	→ Manipulation is fast.
vii)	Constructors: • Arraylist() → Build empty array • Arraylist(Collection c): • Arraylist(int Capacity):	→ • LinkedList() : construct empty list. • LinkedList(Collection c):

Example

```
import java.util.*;
class Dashrath{
public static void main(String[]args)
{
ArrayList<String>list=new Arraylist<String>();
list.add("Dashrath");
list.add("Nandan");
iterator itr=list.iterator();
while(itr.hasNext()){
sop(itr.next());}}}
```

Output:
Dashrath
Nandan

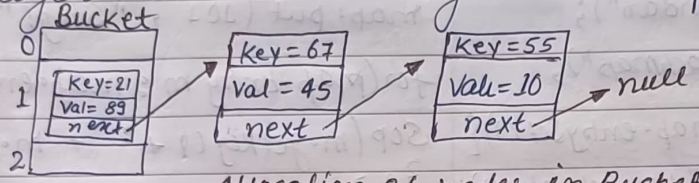
```
import java.util.*;
class Dashrath{
public static void main(String[]args){
LinkedList<String>al=new LinkedList<String>();
al.add("Dashrath");
al.add("Nandan");
iterator<String>itr=al.iterator();
while(itr.hasNext()){
sop(itr.next());}}}
```

Output:
Dashrath
Nandan

★ Java Map Interface

A map contains values on the basis of key i.e. key and value pair. Each key and value pair is known as an entry.

Hashing is the process of converting an object into an integer value. The integer value helps in indexing.



Allocation of nodes in Bucket

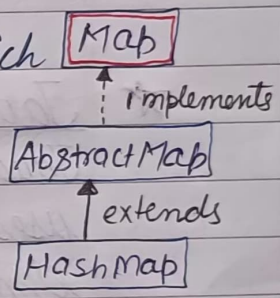
equals(): checks equality of two object.

hashCode(): Return refer. of the object in integer form.

Buckets: Array of Nodes.

✳ Java HashMap

Hash map class implements Map interface which allow us to store key and value pair, where keys should be unique.



Parameters:

K: It is type of keys.

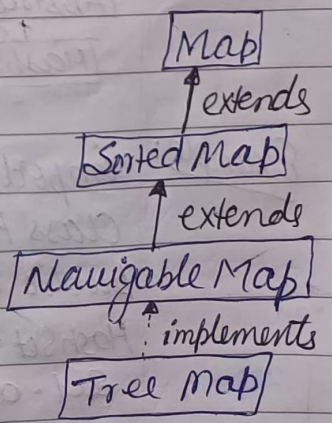
V: It is type of mapped values.

Constructors: HashMap(), HashMap(Map m), etc.

Note: HashSet contains only values whereas HashMap contains an entry (key and value).

✳ Java TreeMap class

Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.



- Java TreeMap is non synchronized.

Parameters: K, V.

Constructor: TreeMap(), TreeMap(Comparator<? super K> comparator).

Example

Java HashMap

```
import java.util.*;
public class HashMap {
public static void main(String[]args)
{
HashMap<Integer, String> map =
new HashMap<Integer, String>();
map.put(1, "Dashrath");
map.put(2, "Nandan");
SOP("Iterating Hashmap");
for(Map.Entry m: map.entrySet()){
SOP(m.getKey()+" "+m.getValue());
}}}
```

Output: 1 Dashrath
2 Nandan

Java TreeMap

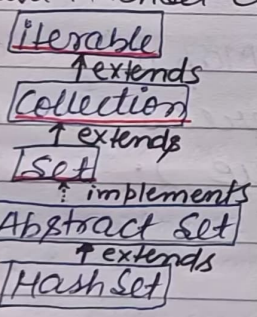
```
import java.util.*;
Class TreeMap{
public static void main (String[]args){
TreeMap<Integer, String> map = new TreeMap
<Integer, String>();
map.put(100, "Dashrath");
map.put(102, "Nandan");
map.put(101, "Singh");
for(map.Entry m : map.entrySet()){
SOP(m.getKey()+" "+m.getValue());
}}}
```

Output: 100 Dashrath
101 Singh
102 Nandan

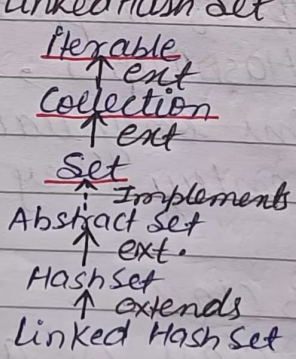
★ Java HashSet class

Java HashSet class is used to create a collection that uses a hash table for storage. Hashset stores the elements by using a mechanism called hashing.

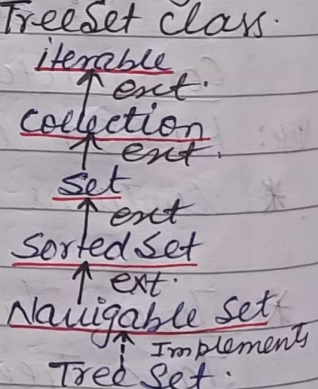
Java HashSet class



LinkedHashSet



TreeSet class.



Example:

```
import java.util.*;
Class HashSet{
public static void main (String[]args){
HashSet<String> Set = new HashSet();
Set.add("one"); Set.add("Two");
Set.add("Three"); Set.add("Four");
Iterator<String> i = Set.iterator();
While(i.hasNext()){
SOP(i.next()); }}}
```

Output: Two One Four Three.

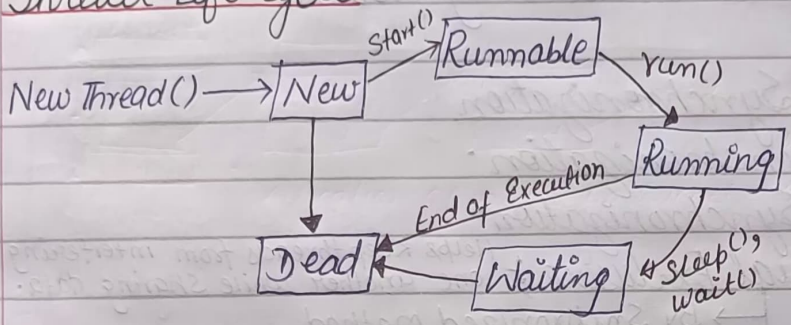
⇒ The elements iterates in an unordered collection.

☆ Multithreading

Multithreading is a java feature that allows concurrent execution of two or more parts of a program, thread.

→ Threads are light-weight processes within a process.

* Thread Life Cycle



- New: A new Thread begin its life cycle.
- Runnable: After thread is started, it becomes runnable
- Running, Waiting, Dead.

* Advantage

- Improved performance & Concurrency.
- Simultaneous access to multiple application.

Disadvantage

- Difficulty of writing code.
- Difficulty of debugging.
- Difficulty of testing.

* How to create Thread

1) By extending Thread class

```
class Multi extends Thread {
public void run() {
SOP("Thread is running"); }
PSVM(String [] args) {
Multi t1 = new Multi();
t1.start(); } }
```

Output: thread is running

2) By implementing Runnable interface.

```
class Multi2 implements Runnable {
public void run() {
SOP("Thread is running"); }
PSVM(String [] args) {
Multi m1 = new Multi2();
Thread t1 = new Thread(m1);
t1.start(); } }
```

Output: thread is running

☆ Synchronization

Synchronization in java is the capability to control the access of multiple threads to any shared resource.

- It is mainly used to -
  1) To prevent thread interference
  2) To prevent consistency problem.

* Types of Synchronization

1) Process Synchronization.
2) Thread Synchronization.
   - Mutual Exclusive ; Helps keep threads from interfering with one another while sharing data.
     - By Synchronized method
     - By Synchro block
     - By Static synchronization
   - Cooperation (Inter-thread Communication in java)

i) Synchronized Method

It is used to lock an object for any shared resource.

Example:

```
Class Table {
Synchronized void print(int n){
  for(int i=1 ; i<=5 ; i++){
      SOP(n*i);
  try { Thread.sleep(400); }
  catch(Exception e){ SOP(e); }}}

class MyThread1 extends Thread {
  Table t;
  MyThread1(Table t){ this.t=t;}
  Public void run(){ t.printTable(5);}}

class MyThread2 extends Thread
{ Table t;
  MyThread2(Table t){
  this.t = t; }
  Public void run(){
    t.printTable(100); }}

Public class TestSynchronization{
  PSVM(String[] args){
    Table obj = new Table();
    MyThread1 t1 = new MyThread1();
    MyThread2 t2 = new MyThread2();
    t1.start();
    t2.start(); }}
```

Output:

5	100
10	200
15	300
20	400
25	500

ii) **Synchronized block**

It can be used to perform synchronization on any specific resource of the method.

Ex

```
class Table{
void printTable(int n){
  synchronized(this){
    for(int i=1; i<=5; i++){
      sop(n*i);
      try{ Thread.sleep(400); }
      catch(Exception e){ sop(e); }
    }}
  : Rest code same as
    Synchronized method's code
```

Synchronization Block

iii) **Static Synchronization**

If you make any static method as synchronized, the lock will be on the class not on object.

Ex

```
class Table{
synchronized static void printTable(int n)
{
  : Rest code same as
  } Synch^n. method's code.
```

Lock: Synchronization is built around an internal entity known as the lock or monitor. Every object has an lock associated with it.

☆ **Thread Priority**: Each thread has a priority. Priorities are represented by a number between 1 and 10. The thread scheduler schedules the thread according to their priority.

⇒ Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and value of MAX_PRIORITY is 10.

Ex

```
import java.lang.*;
public class ThreadP extends Thread{
  public void run(){sop("Inside");}
PSVM(String[] args){
ThreadP th1 = new ThreadP();
Thread p th2 = new ThreadP();
sop("Priority of th1"+th1.getPriority());
sop("Priority of th2"+th2.getPriority());
th1.setPriority(6);
th2.setPriority(9);
sop("Priority of th1"+th1.getPriority());
sop("Priority of th2"+th2.getPriority());}}
```

Output:

```
Priority of th1 : 5 // default
Priority of th2 : 5
Priority of th1 : 6
Priority of th2 : 9
```

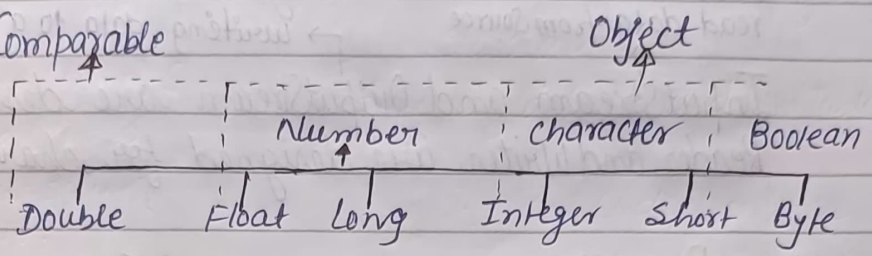
Advantage :

Collections allowed only object data.
On object data we can call multiple methods compareTo(), equals(), toString()
Cloning process only objects
Object data allowed null values.
Serialization can allow only object data.

☆ Wrapper Class

The wrapper class in java provide the mechanism to convert primitive into object and object into primitive.

* Hierarchy:



- Autoboxing: The automatic conversion of primitive data type into its corresponding wrapper class. Eg. byte to byte, char to Character, long to Long, float to Float, boolean to Boolean

- Unboxing: The automatic conversion of wrapper type into its corresponding primitive type. Reverse of boxing.

Ex:

```
public class Wrapper {
PSVM(String[] args){
 byte b=10;
 int i=30;
 char c='a';
//autoboxing:
 Byte byteobj=b;
Integer intobj=i;
Character charobj=c;
// Printing objects
SOP("Byte object:"+byteobj);
SOP("Integer Object:"+intobj);
SOP("Character Object:"+charobj);
// unboxing
byte bytevalue=byteobj;
int intvalue=intobj;
char charvalue=charobj;
// Printing primitives
 SOP("byte value"+bytevalue);
SOP("int Value:"+ intvalue);
SOP("char value:"+charvalue);}}
```

Output:

```
Byte Object: 10
Integer Object: 30
Character Object: a

byte value: 10
int value: 30
char value: a
```

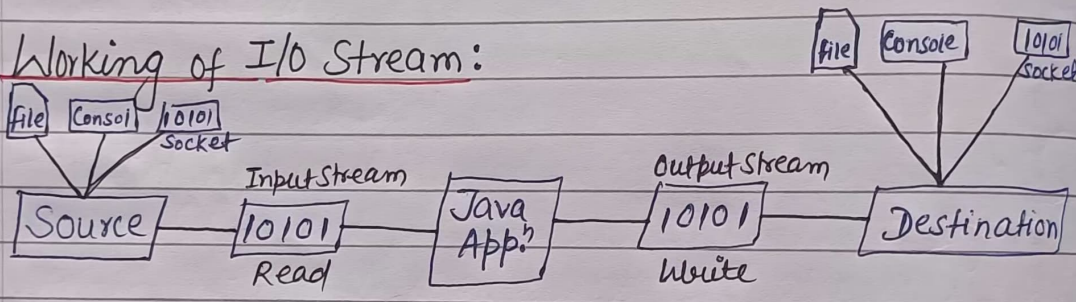
☆ Stream

A stream can be defined as a sequence of data. It is an abstraction that either produces or consumes info.

i) **Byte Streams**: Provide convenient means for handling input and output of bytes. InputStream (read data from source), OutputStream (Writing data to a destination).

ii) **Character Streams**: ..... for handling input and output of characters. Reader and Writer are designed for character stream.

* Working of I/O Stream:

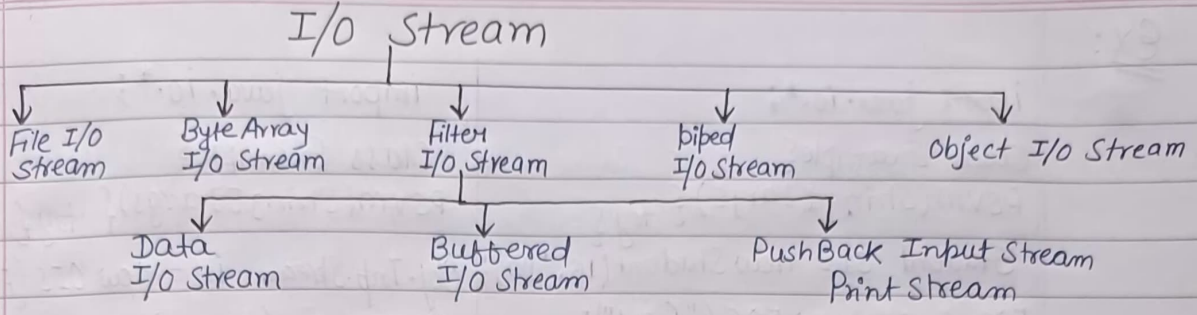


* Byte Stream

```
import java.io.*;
public class CopyFile{
PSVM(String []args) throws IO Excep.
{
 FileInputStream fin = null;
FileOutputStream fout = null;
try{ fin = new FIS ("input.txt");
fout = new FOS ("input.txt");
 int c;
while((c = in.read()) != -1){
  fout.write(c); finally{
if(fin != null){ fin.close();}
if(fout != null){ fout.close();}}}}
```

Character Stream

```
Same code
.
.
.
File Reader fin = null;
File Writer fout = null;
.
.
.
.
Same code
```

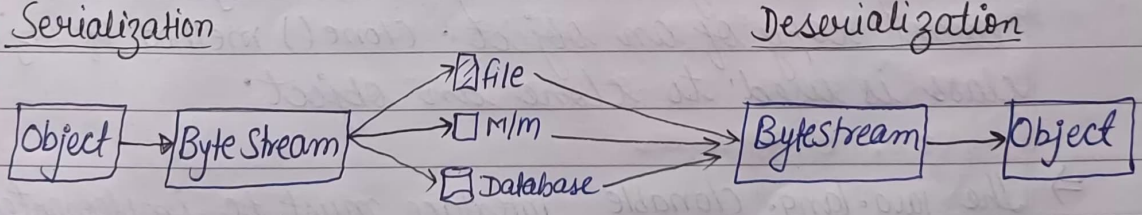


- Buffered byte Stream; allows java to do I/O operations on more than a byte at a time, hence increasing performance.

- Standard Streams, where the user's program can take input from a keyboard and produce output on the computer screen. Standard Input - System.in ; Standard Output - System.out ; Syst.err

☆ Serialization

Object Serialization is the process of saving an object state to a sequence of bytes, as well as the process of rebuilding those bytes into a live object at some future time. You can only serialize the object of a class that implements Serializable interface.



Deserialization: It is the reverse process of serialization, where the byte stream is used to recreate the actual java object in memory.

Ex:

```
import java.io.*;
class Example{
PSVM(String[]args){ try{
Student s1 = new Student(16,"Dn");
FOS fout= new FOS ("f.txt");
Object OS out= new OOS(fout);
out.writeObject(S1);
Out.flush(); Out.close();
SOP("Serialized); }
catch(Exception e) { SOP(e);}}}
```

Output: Serialized

```
import java.io.*;
Class Example{
PSVM(String[]args){ try{
Obj.Inp.Stream in = new OIS(FIS("f.txt");
Student s = (Student) in.readObject();
SOP(s.id+" "+s.name);
in.close(); }
catch(Exception e){ SOP(e);}}}
```

Output: 16 Dn

* **The Keyword: transient**

transient keyword is used in Object Serialization. The transient keyword provides us with the ability to control the serialization process and give us flexibility to exclude some of object from serialization.

* **Cloning:** The object cloning is a way to create exact copy of an object. clone() method of object class is used to clone an object.

⇒ The java.lang.Clonable interface must be implemented by the class whose object clone we want to create.

★ Annotations

- Annotations are used to add meta-data to the Java Elements i.e. instance variable, Constructors, methods, classes, etc.
- Annotations start with @

* Types of Annotation

1. **Marker Annotation:** @Override annotation assures that the subclass method is overriding the parent class method.
   → It has no method.

2. **Single Value Annotation:** An annotation that has one method.
   `@TestAnnotation("testing");`

3. **Multi-Value Annotation:** An annotation that has more than one method.
   `@TestAnnotation(Owner="Dashrath", Value="Nandan")`

* Custom Annotations

User-defined annotations can be used to annonate program elements.

`[Access specifier] @interface <Annotation Name> { Data Type <Method Name>() [default Value]; }`

Example:

```
Class Animal{
Public void run(){
  SOP("Running");}}
Class Dog extends Animal{
   @override
  Public void run(){
    SOP("Dog");}}
Class Main{
  PSVM(String [] args){
    Dod d1= new Dog();
    d1. run();}}
```

Output: Dog

Types:

1. Predefined annotations:
   @Deprecated, @override
   @SuppressWarnings, @SafeVarargs

2. Meta-annotations:
   @Retention @Documented
   @Target @Inherited

3. Custom annotations

☆ JUnit

JUnit is a unit testing framework for java programming language. JUnit has been important in the development of test-driven development.

→ Testing is the process of checking functionality of an application to ensure it runs as per requirements.
→ Unit testing is done at the developer's level.

Features:

i) JUnit is an open source framework.
ii) Provide annotations to identify test method.
iii) Provide test runners for running test.
iv) Allows you to write codes faster.

Unit Test Case: It is a part of code, which ensures that another part of code (method) works as expected.

* Annotations for JUnit testing

@Test specifies that method is test method.
@Before class method invoked only once, before starting all tests.
@Before method invoked before test case.
@After, @After class

* Assert class

The org.junit.Assert class provide methods to assert the program logic.

Methods:

Void assertTrue(boolean condition): Check that condition is true.
Void assertFalse(boolean Condition):
Void assertNull(Object obj): Check that object is null.

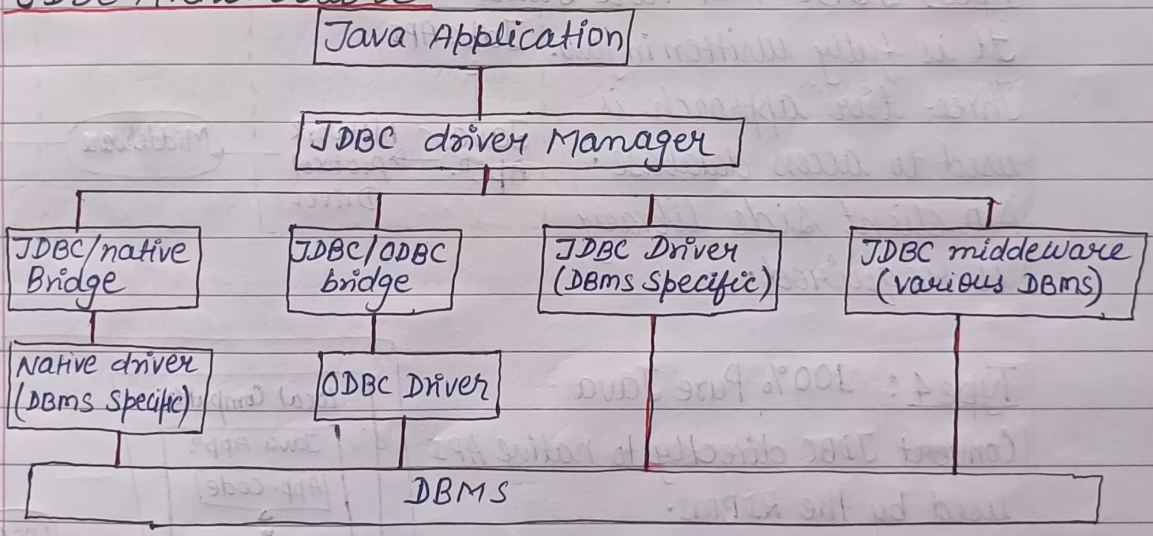
Dashrath Nandan

★ JDBC : Java Database Connectivity

JDBC is a java API to connect and execute query with database. JDBC API uses jdbc drivers to connect with the database

* JDBC	OBDC
i) Java Database Connectivity.	→ Open Database Connectivity.
ii) Used for only Java languages.	→ Used for any lang. C, C++, Java, etc.
iii) We can use it on any platform.	→ Only Windows platform.
iv) JDBC is Object-Oriented.	→ OBDC is procedural.
v) SunMicrosystem in 1997.	→ Introduced by Microsoft in 1992.

* JDBC Architecture



* Working with Database using JDBC

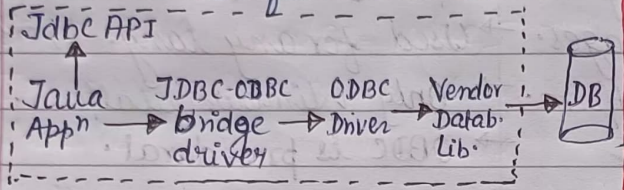
Step1: Importing Packages: import java.sql.*; , import java.math.*;
2: Loading JDBC Drivers: class.forName("oracle.jdbc.driver.OracleDriver");
3: Create Connection : DriverManager.getConnection(url, User, Password);
4: Creating a Statement Object: Statement Stmt = Con.CreateStatement();
5: Execute Query and Returning a Result Set Object.
6: Processing the Result Set.
7: Closing the Result Set and Statement object.
8: Close Connection.

★ JDBC Driver

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types-

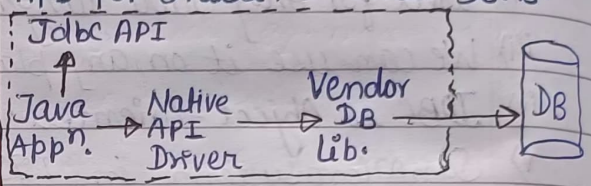
Type 1: JDBC-ODBC Bridge

It converts JDBC method calls into the ODBC function calls.



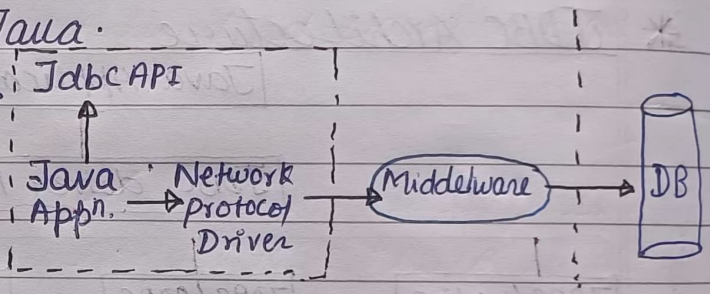
Type 2: JDBC-Native API

Native-API partly-Java Driver Convert JDBC calls on the client API for Oracle, or other DBMS.



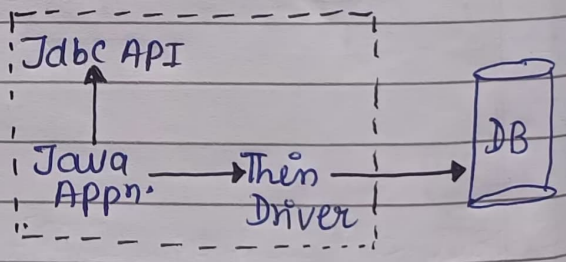
Type 3: JDBC-Net Pure Java.

It is fully written in java. Three-tier approach is used to access database. No client side library is required.



Type 4: 100% Pure Java.

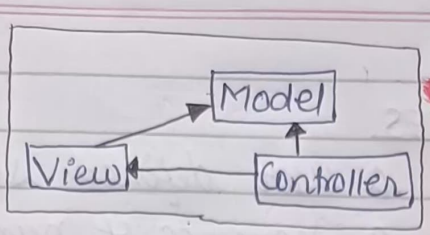
Convert JDBC directly to native API. used by RDBMS. It doesn't need any configuration on the client's machine.



★ MVC Framework

The model-view-controller (MVC) is an architectural pattern that seperates an application into three main logical component- the model, the view, the controller.

* MVC Components



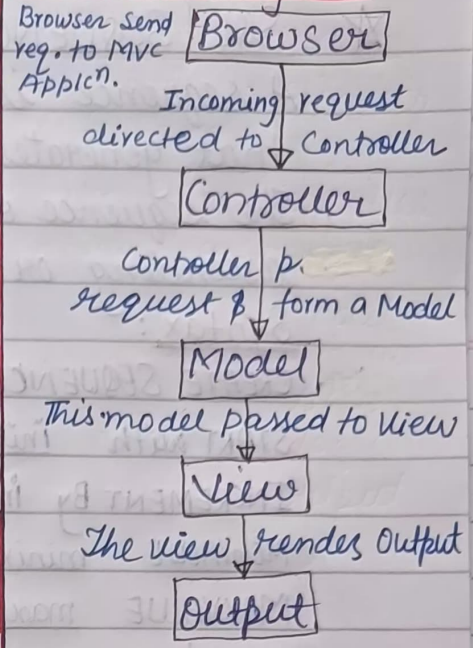
Three Components —

i) **Model:**
The Model component corresponds to all the data-releated logic the the user works with.

ii) **View:**
It is used for all the UI logic of applcn.

iii) **Controller:**
Controller acts as a interface b/w Model and View to process all the business logic and incoming requests.

MVC Flow Diagram



* Example:

```
import java.sql.*;
class db{
PSVM(String[]args){
try{ // Load the driver
class.forName("oracle.jdbc.driver.");
Connection con = DriverManager.getConnection("jdbc:oracle:thin:@
localhost:1521:xe");
// Create connection object
Statement stmt = con.createStatement();
ResultSet rs = executeQuery("$elect* from student");
SOP("value from student table are:");
While(rs.next())
SOP(rs.getString(1)+" "+rs.getString(2)+" "+rs.getData(3));
con.close();}
catch(Exception e){
SOP(e)}}}
```

Output:

```
Values from student table:
21BCS11716  Dashrath Nandan
21BCS11682  Aditya Bansal
```

☆ SQL | SEQUENCES

A sequence is a user defined schema bound object that generates a sequence of numeric values.
The sequence of numeric values is generated in an ascending or descending order.

Syntax:

```
CREATE SEQUENCE  Sequence_name  → Name of the Sequence
START with  inital_value  → Starting value from where Seq. start
INCREMENT By  increment_value  → Value by which Seq. increment itself
MinVALUE  minimum_value  → Min. value of sequence
MAXVALUE  maximum_value  → maximum value of sequence
CYCLE | NOCYCLE ;  → An exception will thrown if seq. exceeds max_value
   ↳ when seq. reaches its set_limit it start from begnin
```

☆ Dual Table

The DUAL is special one row, one column table present by default in all Oracle databases. The table has a single VARCHAR2(1) column called DUMMY that has a value of 'X'.

```
DESC DUAL;
```

Output:

Name	Null?	Type
DUMMY		VARCHAR2(1)

```
SELECT * FROM DUAL;
```

DUMMY
X

* Using basic data types:

The microsoft JDBC Driver for SQL Server uses the JDBC basic data types to convert the SQL Server data type to a formate that can be understood by the Java Programming language and vice-versa

* Data type Mappings:

SQL Server Types	JDBC Types	Java lang. Types
bigint	BIGINT	long
binary	BINARY	byte[]
bit	BIT	boolean
char	CHAR	String
date	DATE	java.sql.Date
decimal	DECIMAL	java.math.BigDecimal
float	DOUBLE	double

* Operations:

- Retrieving data as a string:
- Retrieving data by data type.
- Updating data by parameterized query.
- Passing parameters to a stored procedure.
- Retrieving parameters from a stored procedure.

Example:

```
Create SEQUENCE Sequence_1
Start with 1
increment by 1
minvalue 0
maxvalue 100
Cycle;
```

⇒ Above query will create a sequence named Sequence_1. Sequence will start from 1 and will incremented by 1 having maximum value 100. Sequence will repeat itself from start value after exceeding 100.