Pojo & Hibernate Introduction keynotes :-

=> Maven =>

Pojo => Plain old java object }
PojI => Plain old java interface }

```
class Launch          POJO          class Launch extends HttpServlet
{    ≡                              {           not a POJO
     ≡
}                                   }

                                    @Service               @Data
class Plane                         class Business         class Launch
{                   POJO            {                      {        POJO
                                            Pojo ✓         }
}
class FighterPlane extends Plane    }
{
        POJO
}
                         ≡
                     @Data
interface shapes   } class Employee
{                  } {      ≡ }
    ≡   POJI       }        setter  }
                   }        getter  }
}                           toString}
                                                        Table
                     }
```

Framework => Hibernate | spring

=> Terminologies :-

Persistence :-

Process of storing data for long term which can also be managed easily ==> Persistence

To achieve this we have used :
1. File Handling // IO operation ==> (Store data in HD) ==> java.io.*;

2. Database(JDBC) ==>(Store the data in table within DBMS)==>java.sql.*;

Persistance Store :-

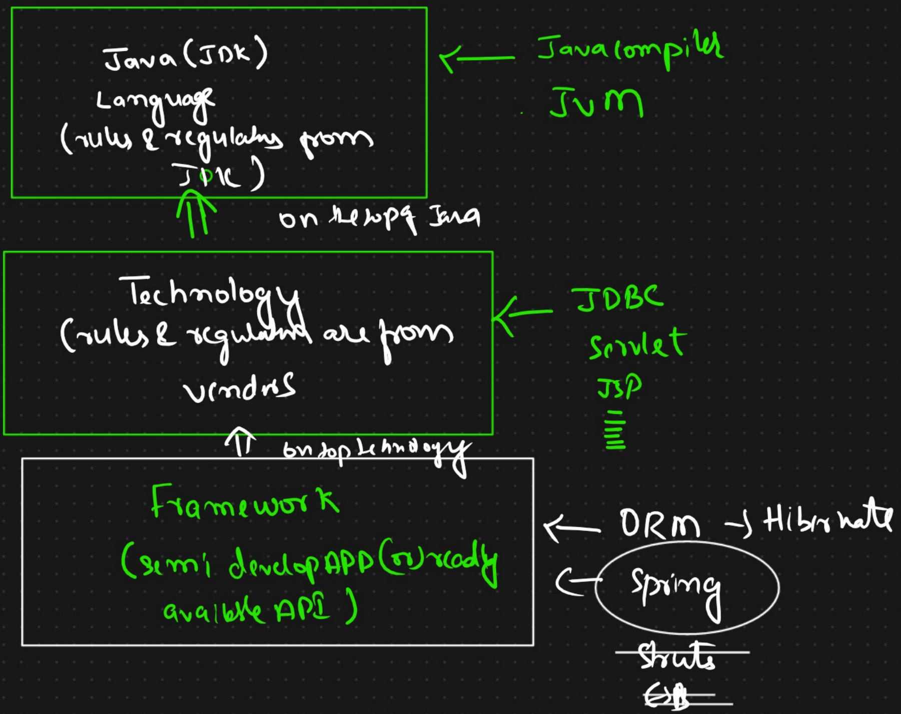
Its a place where data will be stored or saved and managed ==> (Files in HD, MySQL, PostgreSQL, Oracle ....)

Persistence data ==> The data which we have stored or saved ==> DB tables with records....

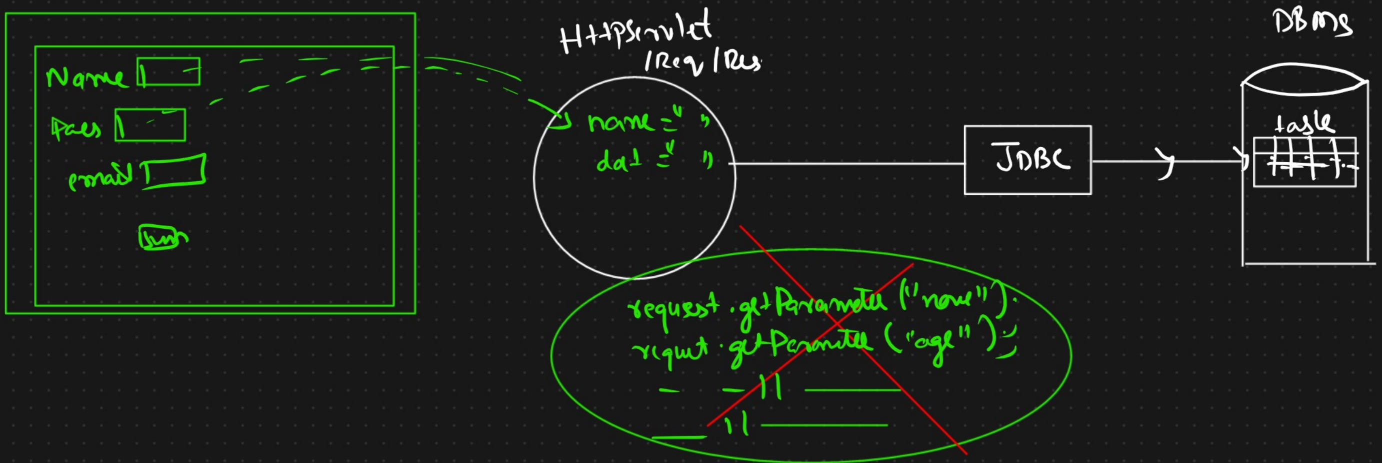
Persistence Operation==> Operations which we perform on persistence store.
(CRUD) Insert, update, delete, select)

Persistence Logic==> The code/logic which we write to perform persistence operation
(IO ==> Streams class . Serialialization ...)
(JDBC code, hibernate code, spring orm, spring jdbc, spring datajpa....)

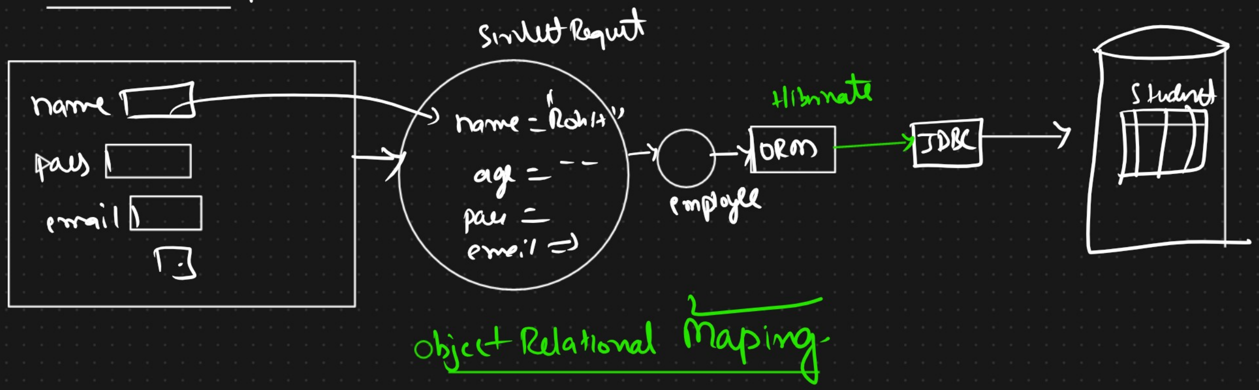
Persistence technology/framework==>
The tech/framework which we use in order to perform persistence logic
exam==> JDBC
Hibernate
Spring ....



using technology



Framework :-



==> Limitations associated with JDBC :-

==> We need to write SQL queries by following the syntax of db.
(Strong knowledge on SQL is also needed)
==> There is lot of boilerplate code ==> A code which repeats multiple times in a project with same or small changes (Same steps we have to follow for every operation)
==>JDBC throws only one Exception which is SQLException==> We donot have detailed Exception hierarchy for different problem

==>While closing the jdbc connection we need to analyze code properly to avoid null pointer exception.

==>JDBC ==> Local Transaction ==> We donot have proper global transaction management.

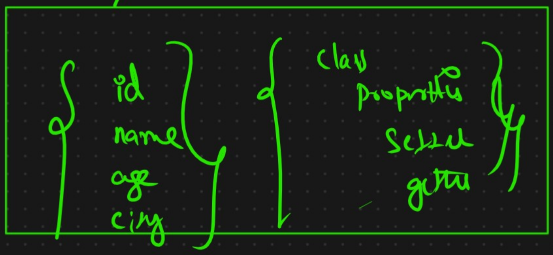
==>Java ==> OOPs ==> Object.
Using JDBC we cannot send Object to database==> Queries expects us to pass values not Object.

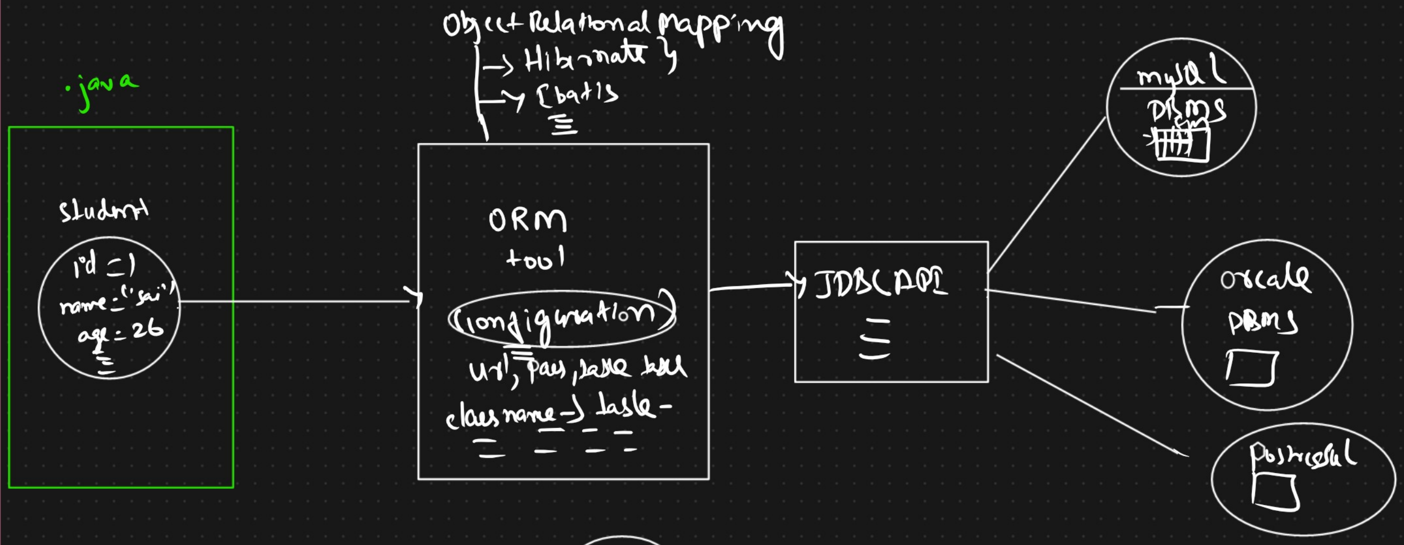
==> JDBC supports only positional Parameters , its difficult for us to inject the values==> It doesn't support named parameters.

"insert into student (id, name, age) values (?,?,?,?)"
"insert into student (id, name, age) values (:id,:name,....)" but we cannot use this in JDBC

==>While developing persistence logic we cannot make use of Java important feature (Inheritance, polymorphism, composition ....)

↳ solution to all this problem => ORM ↳ Object Relational mapping

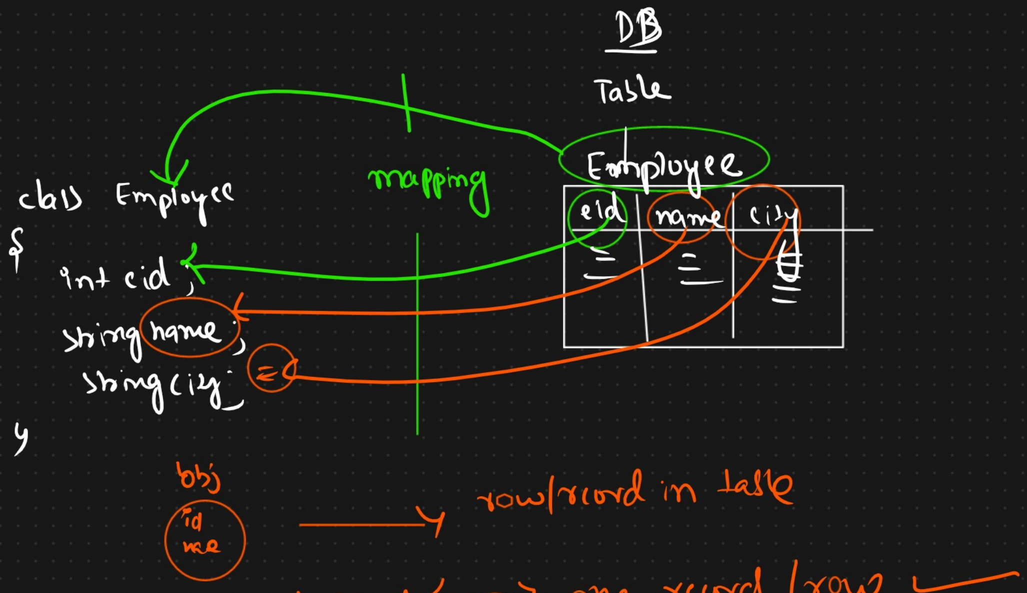




$$\begin{tikzcd} \text{Object} \arrow[r, "\text{ORM}"] & \text{row(record)} \end{tikzcd}$$

1. save(stu) ⟶ insert Query will be generated internally
2. update(stud) ⟶ update Query will be generated internally

get()
delete()



1 object <==> one record / row