What is data?
information with no context.

What is database?
Organized collection of data

What is RDBMS?
data stored in form of rows & columns.

SQL Introduction
↳ Structured query language

Introduction to MySQL

fast & easy way to create tables

It is open source & free

Named after Monty Widenius 's daughter: My

It is a Relational database management system.

Important MySQL Queries

Create Database

```
create database <name of database>;
```

Create table

```
Create table <name of table> ( c1 dt1, c2 dt2
  c3 dt3 ..... PRIMARY KEY(c1) );
```

Select query

```
select * from <database>.<tablename>;
```

(* → all)

Insert query

```
insert into <database>.<tablename> values
  ( v1, v2, v3 ..... );
```

delete query

delete from <database>.<table name>
where <condition>

<condition> can only be on the PRIMARY KEY

for instance if "id" is the pkey then

where id = 2;

won't work as name is not the primary key

where name = "Ram";

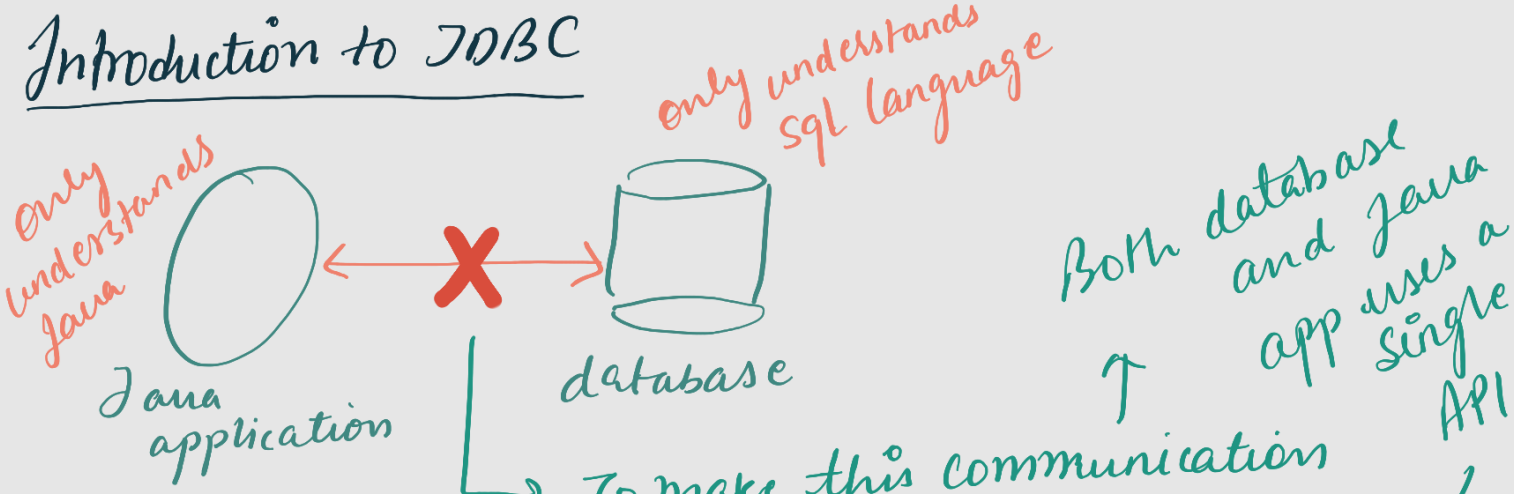
update Query

update <table name>
set colname = value
where <condition>;

drop database

drop database <database name>;

Introduction to JDBC

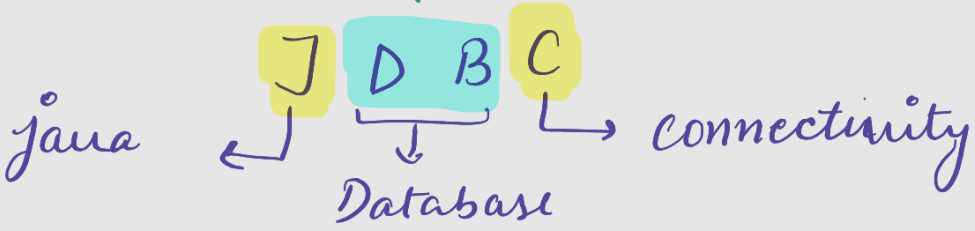


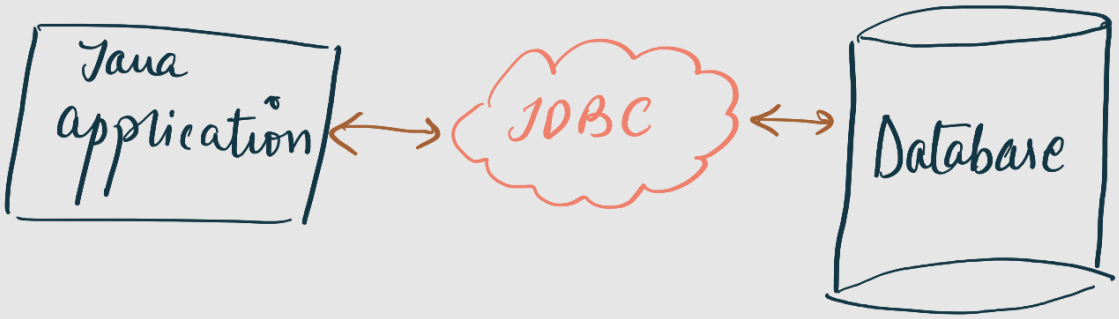
Both database and Java app uses a single API

↑

To make this communication possible

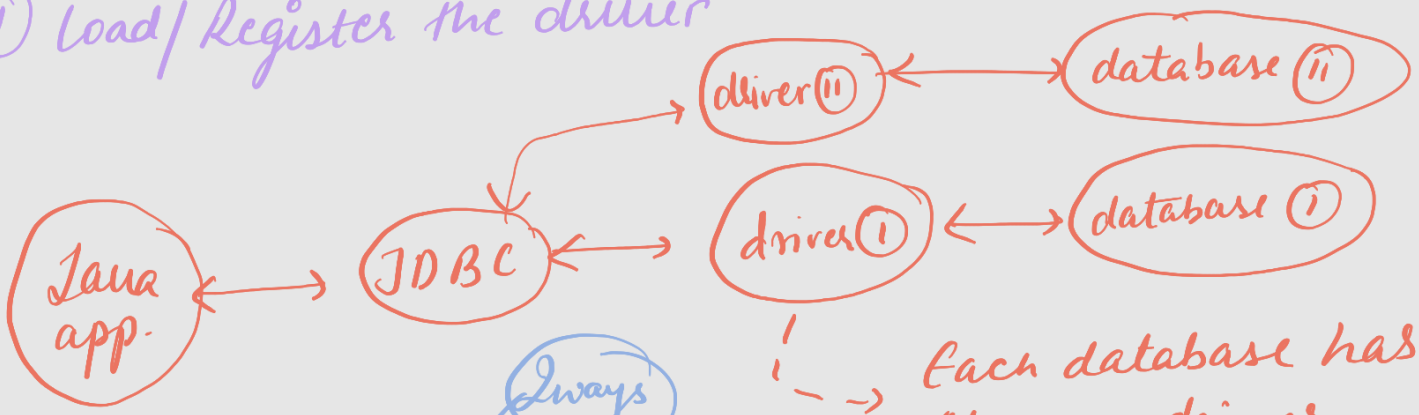
JDBC





The five steps to connect Database with JDBC

① Load/Register the driver



Always

- Class.forName(<class>)
  - public static method
  - fully qualified class
    - format of fully qualified class
      - com.org.mod.sub.class
        - mod → module
        - org → Organisation
        - sub → Submodule
    - includes class & packages
- Create a driver object

dependent on database

fully Qualified class for mysql

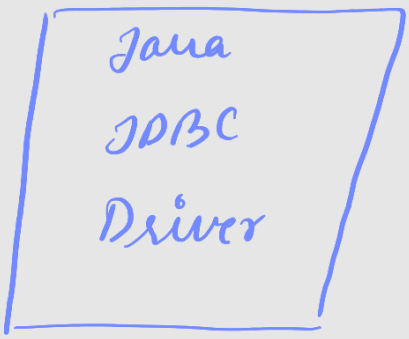
com.mysql.cj.jdbc.Driver

Create a new Driver object

```
Driver driver = new Driver();
DriverManager.registerDriver(driver);
```

DriverManager → class present in java.sql pkg.

After loading of driver both JDBC & Driver attach to the Java program but JDBC has no connection to Driver and is unknown of the Driver's location.



To establish the connection we will go through the next step

↓

Establish a connection b/w Driver & JDBC.

② Establish the connection b/w Driver & JDBC

```
DriverManager.getConnection(url, user, pass);
```

```
DriverManager.getConnection(url, properties file);
```

properties file → has user and pass.

```
DriverManager.getConnection(query);
```

query includes url, user and password

All the methods

return
"connection interface"

username
password

When the connection is established we need to move on to establishing a statement / query

③ Establish a statement

Statement
base interface
static query
no placeholder needed
slow performance

Connection.createStatement();

PreparedStatement
type of Statement
So child interface
child interface
dynamic query
placeholder used (delimeter, ?)
fast performance

connection.prepareStatement();

④ Execute the statement

execute();
Return type → boolean

Statement.execute();
PrepareStatement.execute();

executeUpdate();
Return Type → int

↓

Non Select Query

↳ update
delete etc etc.

executeQuery();
Return type → ResultSet

↓

Select Query

⑤ Close the connection

connection.close();

Short Notes on Implementation

① Create a Maven Project

Group id ⇒ package name ⇒ com.ty

Artifact id ⇒ project name

② Go to pom.xml file

add <dependencies> </dependencies>

Go to mvn repository and search for mysql

Choose mysql Connector

Choose the version acc. to the usage → copy-paste the code given

---

```
String url = "jdbc:mysql://localhost:3306";
String user = "username";
String password = "password";

//step 1 - load the driver
Class.forName("com.mysql.cj.jdbc.Driver");
// step 2 - Establish a connection
Connection connection = DriverManager.getconnection
                        (url, user, password);
// step 3 - Establish the statement
Statement statement = connection.createStatement();
// step 4 - Execute the statement
                        .executeUpdate("query");
```

statement. executeUpdate(...);

//step 5 - Close the connection

connection.close();

Batch Execution

> Execute bunch of queries together

```
//step 1
//step 2
//step 3
    addBatch();
//step 3
    addBatch();
//step 4
    executeBatch();
```

---

com.ty.dto — data transfer object — All private

com.ty.dao — data access object

com.ty.util — Helper class

com.ty.controller — Testing purposes — White Box Testing

Introduction to Hibernate

Disadvantages of JDBC

(i) Database + driver specific
(ii) No lazy/Eager loading
(iii) Automatically creation of tables, pkey is done
(iv) JDBC is a technology & not framework
(v) No mapping supported

Hibernate is a framework and supports mapping It creates tables internally and only methods are used. Hibernate is a ORM framework.

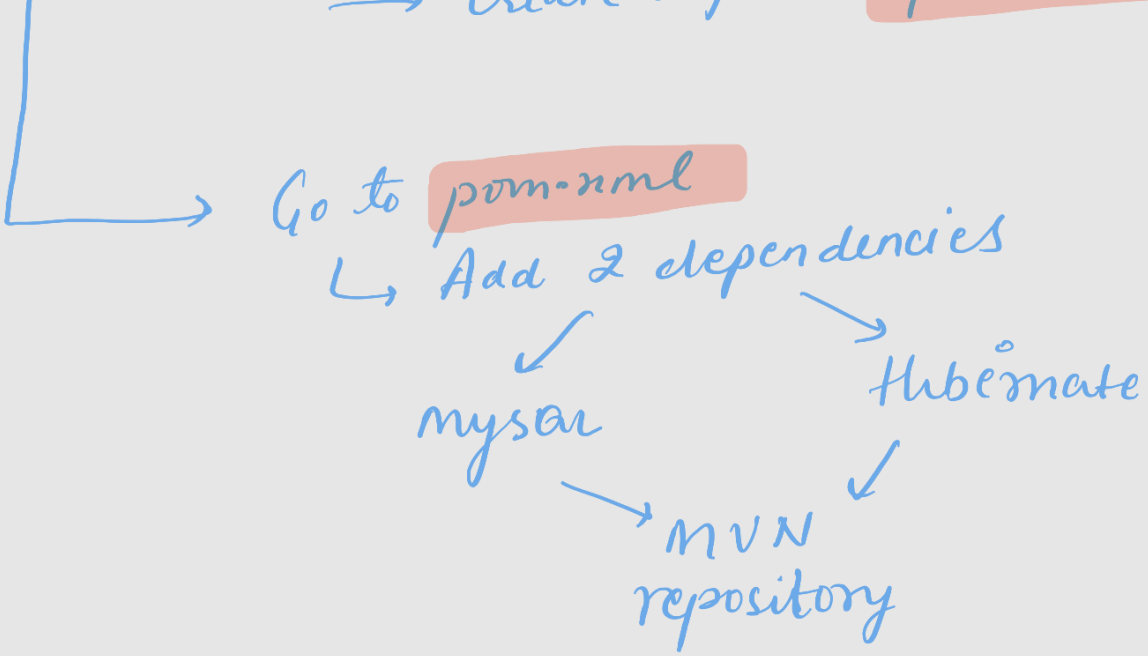
ORM framework
↳ Object Relational Mapping framework
↳ Converts data from RDBMS to OOPS

data → object

---

persistence.xml file
↳ a file which contains all configuration information → database name

Maven Project → src/main/resource
→ Create a folder META-INF
→ Create a file persistence.xml



EntityManagerFactory
EntityManager
EntityTransaction
} Core Hibernate

EntityManagerFactory

- It is an interface
- already present in java.persistence
- set up connection b/w java and database
- helps to create table automatically

```
EntityManagerFactory eMF= persistence.createEntityManagerFactory( unit );
```

- EntityManagerFactory: return type
- unit: set/found in persistance file — persistence unit

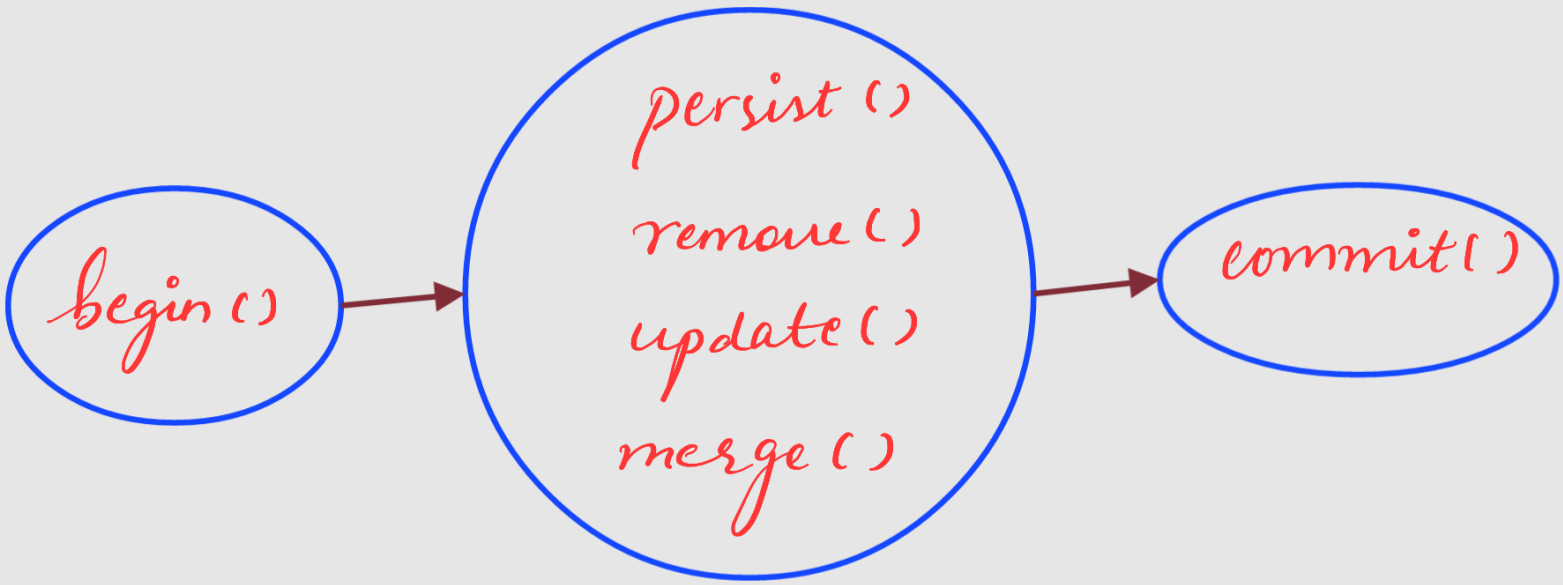
EntityManager

↳ Interface
↳ present in java.persistence
↳ No parameters
↳ Internally establish the statement

```
EntityManager eM = eMF.createEntityManager();
```

Entity Transaction
↳ interface
↳ java.persistance package
↳ Manages all the transactions

```
EntityTransaction eT= eM. getTransaction ();
```



persist()
↳ store data into table

remove()
↳ remove data from table

find()

↳ fetch particular data from table

merge()
↳ if id found → update data
↳ if id not found → save data

When one is fetching data, eT is not req. as we are not manipulating the data

Create Query Method

```
Query q = eM.createQuery(<query>);
```

Select s from Student s

HQL
Hibernate Query Language

get list of all results

```
List<Student> ls = q.getResultList();
for(Student st : ls) {
    sysout(st.getId);
    and soon.... }
```

for each loop

Query Parameters

Parameters

Named

where s.name = :name

placeholder

q.setParameter("name", "Raj");

name is replaced by Raj

Positional Parameters

Select s from Students

where s.name = ?1;

positional parameter → place holder

q.setParameter(1, "Reena");

Question mark is replaced by Reena

Introduction to Mapping

↳ linking two tables together
↳ achieved using hibernate & JPA

java persistence API

Hibernate (+) JPA → mapping, cascade, cache, fetchType

Hibernate → entitymanagerfactory, entitymanager, entityTransaction

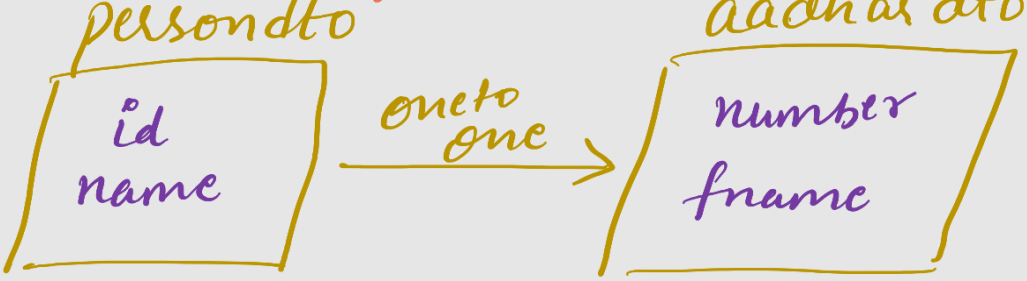
Mapping

- unidirectional → one to one, one to many, many to one, many to many
- bidirectional → one to one, one to many & many to one, many to many

Unidirectional Mapping

One to One Mapping

One person has one aadhar card.




```
@Entity → To identify the class as an entity
public class PersonDto {
  @Id → To set id as p.key
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    @OneToOne
    private AadharDto aadhar;
}
```

→ to generate id automatically so we don't need to handle it.

getters & setters required

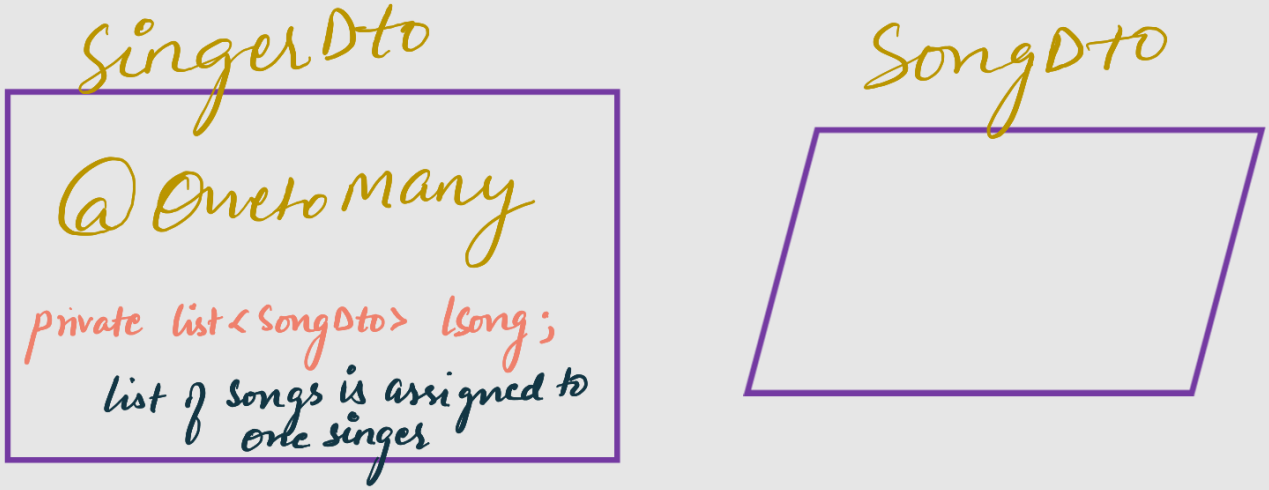
→ connect one aadhar object to one person object

```
@Entity
public class AadharDto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private long number;
    private String fName;
}
```

→ have id in aadhar as well (foreign key)

One to Many Mapping

One singer can have many songs.



Many to One Mapping

Many to One Mapping

Many actors can work in one movie

Moviedto

actordto

@manytoone

private moviedto mov;

assign all actors to one movie.

Many to Many Mapping

Many students can have many subjects.

Studentdto

assign a list of subjects to a student

@manytomany

private list<subjectdto> sub1;

Subjectdto

The same list of subjects can be assigned to multiple students as well.

Bidirectional Mapping

Person

@OnetoOne

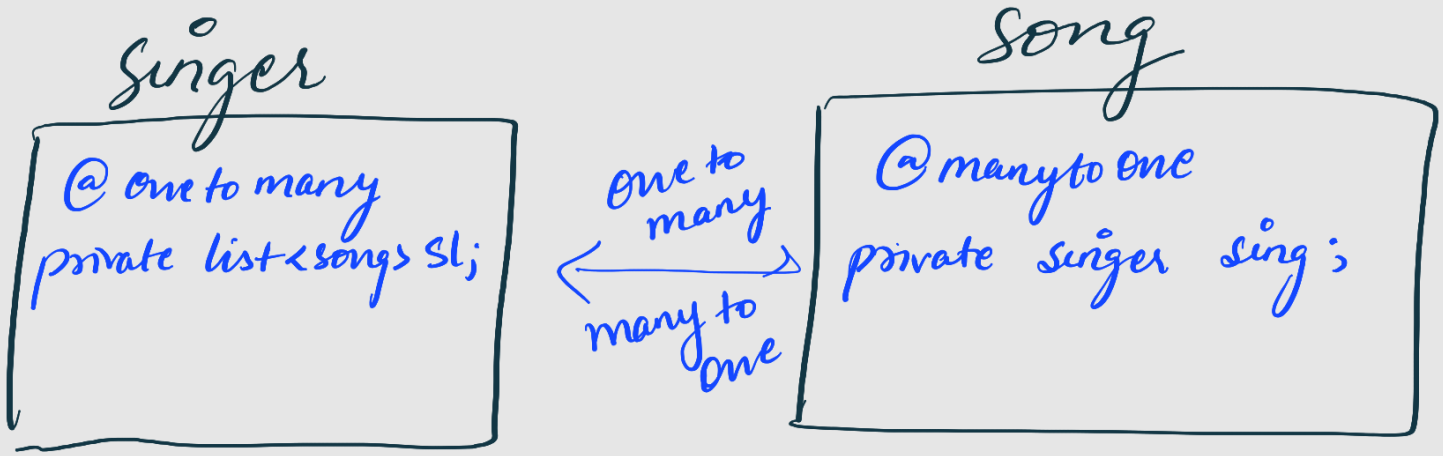
one to one

one to one

aadhar

@OnetoOne

One aadhar belongs to one person and vice versa



And so on....

@ Join Column

↳ In case of bidirectional mapping, foreign key is created in both tables

To avoid that, we use join column in second table (aadhar)

(+)

we will use "mappedBy" in first column

person dto

```
@OneToOne(mappedBy = "p")
private aadhar a;
```

aadhar dto

```
@OneToOne
@JoinColumn
private person p;
```

Introduction to Cascading

When we perform some action on target entity then the same action is applied on the associated entity.

For instance if person class has set CascadeType as

PERSIST then ⇒ when person class is saved/stored
↳ aadhar class is also stored/saved.

(mappedBy = "p", cascade = CascadeType.ALL)
all operations

Introduction to Fetch Type

* One of the strategies for fetching data from database
* present in java.persistence package

Lazy loading Fetch

When called then loaded

Data initialization occurs as long as it is possible

Eager loading Fetch

Everything loaded even if not called

Data initialization occurs on the spot

By default ⇒ Eager Fetch

(mappedBy = "p", fetchType.Lazy)

Default Fetch Types

One to One ⇒ Eager Fetch

One to Many ⇒ Lazy Fetch

Many to One ⇒ Eager Fetch

Many to Many ⇒ Lazy Fetch