The **Singleton Pattern** in Java is a **creational design pattern** that ensures **only one instance of a class exists in the JVM**, and it provides a **global access point** to that instance.

It’s commonly used for logging, caching, configuration settings, database connections, or thread pools—where having multiple instances could cause problems.

Database DB=new Database()

Database DB1=new Database() ---not able to create

 **Single Instance** – Only one instance of the class is created.

 **Global Access Point** – Provides a static method to access that instance.

 **Controlled Instantiation** – Constructor is private to prevent direct creation.

**What is the DAO Pattern?**

**DAO (Data Access Object)** is a **structural design pattern** used to **separate database interaction logic** from the rest of the application (business logic).

Presentation layer --UI html/css -

Business logic layer ---tax calculation

Data layer ------database

It provides an **abstraction layer** between:

* The **Java application (service or business logic)**
* The **database (using JDBC, JPA, Hibernate, etc.)**

So instead of directly writing SQL queries inside your Java classes, you delegate that responsibility to DAO classes.

Database connection ---single file

Utility class ----.properties ,Utility class using this property file

Table –employee

Id ,name,age Entity class

String name;

Int id;

Int age;

----getters and setters for each variable

DAOInterface ----interface

Employeeinsert();

EmployeeUpdate();

DAOInterfaceImplementation ---java class

Class implements our interface

----we will provide all the definitions for the declation

---insert method logic

---update method logic

Main class----

Use your Implementation class and call the functions

| **id** | **name** | **department** | **salary** |
| --- | --- | --- | --- |
| 1 | Raj | IT | 50000 |
| 2 | Meena | HR | 45000 |

A **CallableStatement** is a **JDBC interface** used to **call stored procedures or functions** that are defined in the database.

It’s a subclass of PreparedStatement, and it allows us to execute **SQL stored procedures** with **IN**, **OUT**, and **INOUT** parameters.

CallableStatement cstmt = connection.prepareCall("{call procedure\_name(?,?)}");

**What is SQL Injection?**

SQL Injection is a vulnerability where untrusted input is inserted into SQL statements and interpreted as part of the SQL, allowing attackers to read/modify data or escalate privileges.

Select \* from emplyees where username=’root’ or password=’javafs’

String sql = "SELECT \* FROM customers WHERE email = '" + userEmail + "'";

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql);

String sql = "SELECT \* FROM customers WHERE email = ?";

try (PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setString(1, userEmail);

try (ResultSet rs = ps.executeQuery()) {

...

}

}

Navigation methods

**Navigation methods (most useful)**

* rs.next() — move to next row (returns false at end).
* rs.previous() — move to previous row (only for scrollable types). [1]—move forward [-]

rs.first(), rs.last() — jump to first/last row (scrollable).

* rs.absolute(int row) — move to given row index (1-based). If negative: -1 = last row, -2 = second-last, etc.
* rs.relative(int rows) — move relative to current row (positive/negative).
* rs.isBeforeFirst(), rs.isAfterLast(), rs.isFirst(), rs.isLast() — check position.
* rs.getRow() — current row number (0 if before first or after last).

ResultSetMetaData

What are the columns extract the column name from my table

ResultSetMetaData rmd=rs.getMetaData();

Int cols=rmd.getColumnCount();

For(int i=1;i<=cols;i++)

{

s.o.p(md.getColumnLabel(i) +”,”+ md.getColumnTypesName(i))

}

P0 project

Java ,SQL,JDBC

Date of subimission on 31st

UI also swing

Standalone application

NoSQL ---

Not Only SQL

---database system for storage and retrieval of data that is modelled in means other than the tabular relations

Characters

Schema -less -no predefined structure

---High performance for large volumes of data

Supports unstructured,semi structured or structured data

Types of NoSQL database

Document based : MongoDB,CouchDB

Key value stored : Redis,DynamoDB

Column family : Cassandra,HBase

Graph -based : Neo4j

NoSQL and SQL

Data model Table rows and columns Documents,key value

Schema fixed ,predefined Dynamic

Transaction ACID BASE (Basically Available,Soft state,Eventually consistent)

Query Language SQL varies of DB (MongoDB uses JSON based queries)

MongoDB

----document oriented NoSQL database that stores data in flexible JSON like document

Stored data as BSON(Binary JSON)

Schema -less

Rich query language

Easy integration with modern programming (Python,java )

Developed by MongoDBInc

Primary use –storing unstructured data such as user profiled,log

SQL ---table(rows,columns)

----Collections and documents

Core concept

Database ----contains collections

Collection ----contains documents

Document ---stores actual data in key value pairs

Document Data Model

---Data is stored in documents which are JSON like objects

{

“name”:”Remya”,

“age”:35,

“Skills”:[“java”,”php”,”Python”]

}

Database ---logical container for collections(similar to schema in SQL)

Collection ---a group of related documents (similar to a table in SQL)

Use school

Db.students.insertOne({name:”Remya”,grade :”A”})

Database ----School

Collection ---Students

Document ---- {name:”Remya”,grade :”A”}

BSON ----binary representation of JSON like document

Data storage and retrieval efficient

Support more data types than JSON (Data,ObjectId)

Faster for encoding and decoding

ObjectId

---unique 12 byte identifier automatically generated for each document

“\_id”: ObjectId(58786hsgdfhgf897)