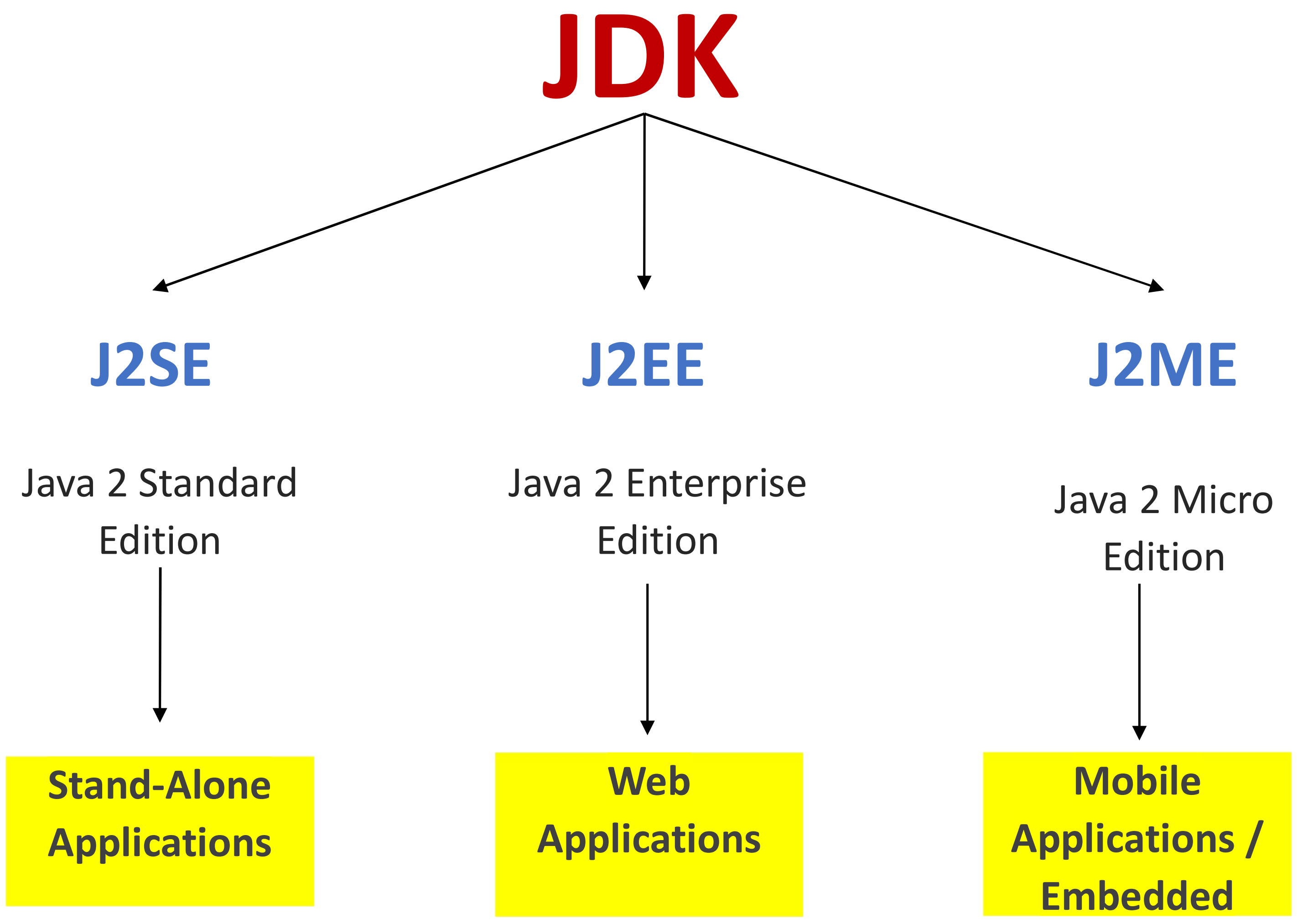
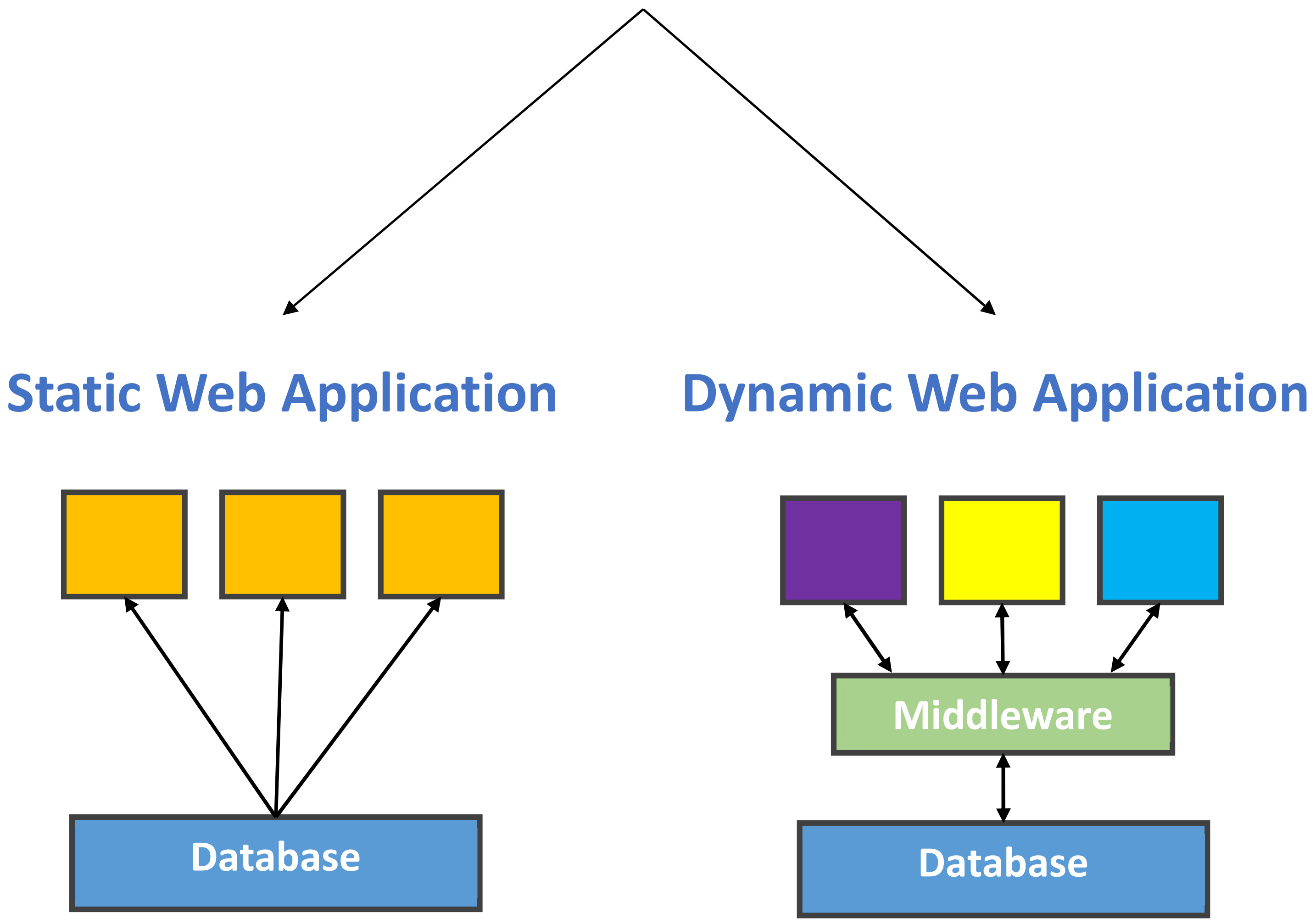
INTRODUCTION TO ADVANCE JAVA



Stand-Alone Application – An application that doesn't require any external/additional software to operate or an application that is independent and does not interact with an external application is called as Stand-Alone Application.

Web Application – An application that interacts with an external device through Network is called as Web Application.

Web Applications



Static Web Application – An application that returns the same output for every user request is called as Static Web Applications.

e.g., W3School, JavaTpoint, Portfolio website, Wikipedia etc.,

Dynamic Web Application – An application that returns different output based on who the user is (user-interactive) called as Dynamic Web Application.

e.g., Instagram, Facebook, etc.,

Types of Web Architecture:

There are 3 types of web Architecture, they are:

1. **2-tier**

   An application that is having frontend (i.e., client side) and a backend (database) is a 2-tier architecture.

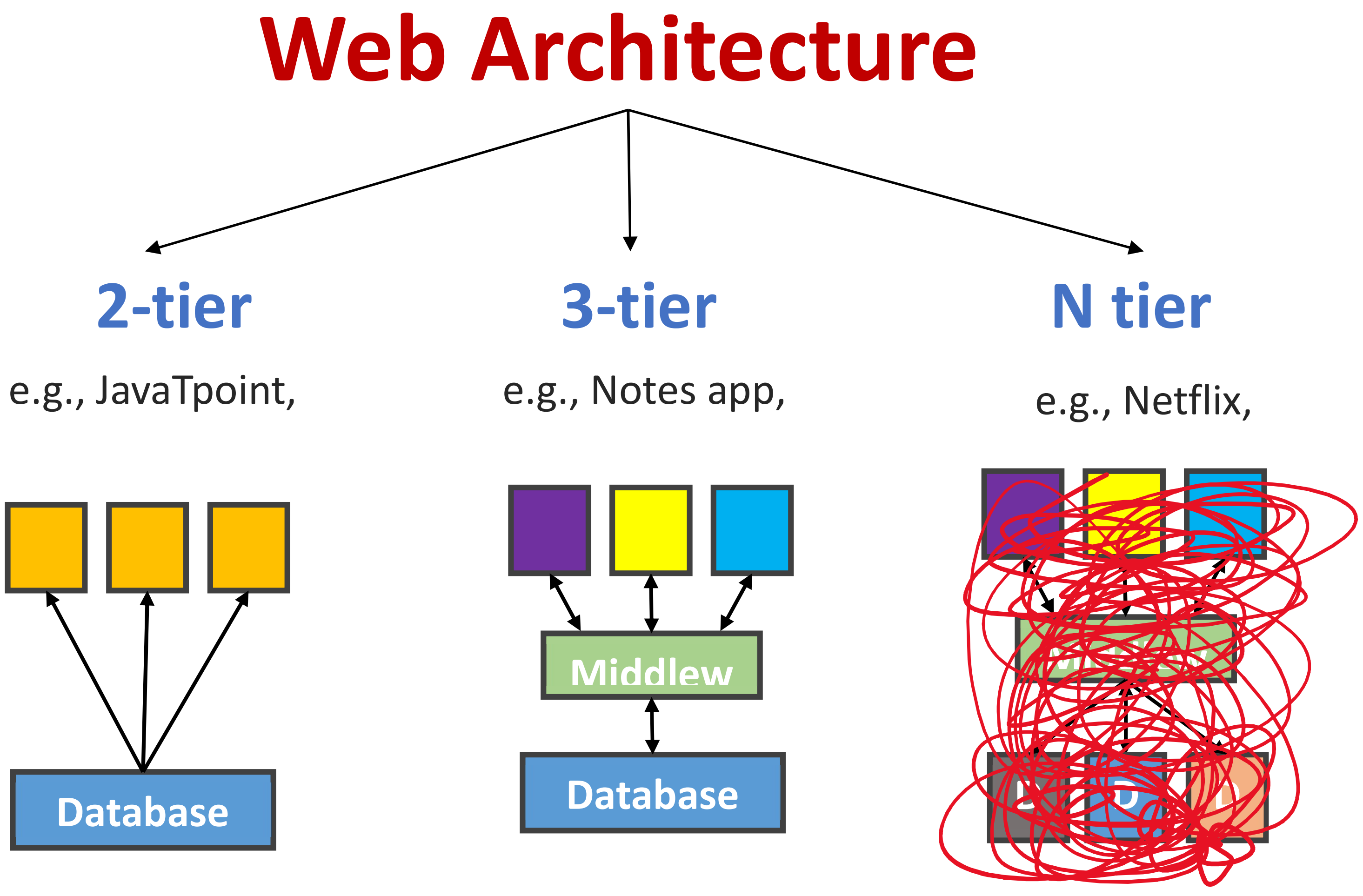
2. **3-tier**

   An application that is having a frontend (i.e., client side), a middleware (i.e., logical side) and backend (database) is a 3-tier architecture.

3. **N-tier**

   N-tier is also called multi-layer or multi-tier architecture. Here there will be multiple layers in the middleware and backend based on the responsibilities.

   e.g., presentation layer, logic layer, business layer, and the data access layer.



Database Vendors

- Database Venders are the software that provides a database and allow you two store your data and interact with it.
- There are many database venders such as –
  - MySQL
  - Oracle
  - SQLite
  - PostgreSQL
  - Etc.,

INTRODUCTION TO JDBC

What is JDBC?

JDBC is an acronym from "**Java Database Connectivity**", it is a "**Standard API Specification**", used to transfer data between the Java Application and the Database.

In order to establish the connection and to interact with the database there are 7 Steps to be followed

7 Steps of JDBC:

- Importing the packages
- Loading and registering the driver
- Establishing the Connection
- Creating the Statement
- Executing the Query
- Processing the Result

- Closing the Connections

Step 1

Importing the Packages:

Java provides an inbuilt package – java.sql, which has to be imported in order to interact with the database.

Main Components:

- Driver (Interface)
- DriverManager (Class)
- Connection (Interface)
- Statement (Interface

Step 2

Loading and Registering the Driver:

There are two ways we can register the driver they are –

1. Using registerDriver(driver) method of DriverManager class

Syntax:

```
Driver driver = new com.mysql.cj.jdbc.Driver();

riverManager.registerDriver(driver);
```

2. Using forName("fully qualified class name") of class Class.

Syntax:
```
Class.forName("com.mysql.cj.jdbc.Driver");
```

Step 3

Establishing the Connection:

- We can open a connection to the database with the static getConnection() method of the JDBC DriverManager class.
- This getConnection() method returns an object of type Connection interface, it accepts the String parameters such as userid, password and database URL,
- This identifies the JDBC driver to use, and the name of the database to which we want to connect.

Syntax:

```
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306","root"
,"Geethagd@123");
```

getConnection methods:

There are basically three overloaded methods of getConnection () they are:

- getConnection ("dbURL", "user", "password");

Syntax:

```
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306","root"
,"Geethagd@123");
```

- getConnection("dbURL");

Syntax:

```
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?us
er=root&password=root");
```

- getConnection ("dbURL", Properties prop);

Syntax:

Connection.properties.

```
dbUrl=jdbc:mysql://localhost:
3306/jdbc_practice

user=root

password=Geethagd@123
```

```
FileInputStream input = new
FileInputStream("D:\\batch07\\x
yz\\src\\connection.properties"
);

Properties p = new
Properties();

p.load(input);

String dbUrl
=p.getProperty(“dbURL”);

connection =
DriverManager.getConnection

(dbUrl, p);
```

Note:

- FileInputStream() will throw FileNotFoundException,
- Load () method will throw IOException
- getConnection (), prepareStatement (), createStatement (), execute (), executeUpdate (), executeQuery (), methods will throw SQLException.
- forName () method will throw ClassNotFoundException.

Step 4

Creating the Statement:

After creating the Connection the next step is to create a statement, it is achieved through the createStatement() method of Connection interface.

The statement object acts as a vehicle between the application and database, used to transfer the data.

Syntax :

```
Statement stmt = con.createStatement();
```

Step 5

Executing the Query :

Statement object provides methods such as execute(), executeUpdate() and executeQuery() which are used to interact with the database.

1. Execute() – (ReturnType – boolean)
   Returns – false for DDL and DML operations.
   Returns – true for DQL operations.
2. ExecuteUpdate() – (ReturnType – int)
   Returns – the number of lines effected.
   Used for DML operations.
3. ExecuteQuery() – (Retruntype - ResultSet)
   Returns – object of ResultSet type.
   Used for DQL operations.

Step 6

Processing the Result:

After fetching the data, the returned ResultSet should be processed using next () method,

next () method helps in iterating over all the objects in the ResultSet.

```
ResultSet resultset = stmt.executeQuery("select * from 
person_table");

while(resultset.next()) {

int id = resultset.getInt(1);

String name = resultset.getString(2);

int age = resultset.getInt(3);

System.out.println("id = "+id+" name = "+name+" age = "+age);

}
```

Step 7

Closing the Connections:

Closing the connections can be achieved using close() method,

The connections should be closed explicitly when no longer in use because not closing connections **could cause timeouts as the connection pool may run out of available connections that can be used.**

Syntax:

```
    finally {
if(con!=null) {
try {
con.close();
} catch (SQLException e) {
e.printStackTrace();
}
}
}
```

PreparedStatement

PreparedStatement is a child interface of Statement interface, used to execute parameterized SQL Queries.

It is a pre-compiled SQL Statement that means it is compiled once and stored in preparedStatement object and used to execute multiple times.

Syntax :

```
PreparedStatement ps=connection.prepareStatement
("insert into user values(?,?)");
```

```java
int id = input.nextInt();

String name = input.next();

ps.setInt(1, id);

ps.setString(2, name);

ps.executeUpdate();
```

Batch Execution

Batch execution is way of executing the batch of statements at once, it is achieved using the methods such as addBatch() and executeBatch().

addBatch() – helps us to group the statements that are to be executed together.

executeBatch() – helps to execute all the statements that are grouped to a batch.

Syntax:

```java
PreparedStatement preparedStatement =
connection.prepareStatement("insert into user values(?,?)");

preparedStatement.setInt(1,101);

preparedStatement.setString(2,"Ram");

preparedStatement.addBatch();


preparedStatement.setInt(1,102);

preparedStatement.setString(2,"Sam");

preparedStatement.addBatch();


preparedStatement.executeBatch();
```

Callable Statement

“CallableStatement is an interface, who’s implementations are given by the object of CallableStatement type which is returned by the prepareCall() method of Connection object.

prepareCall() method will accept a stored procedure as a parameter.”

Stored Procedure: It is a type of method or function created on the Database server side

e.g., syntax:

CREATE PROCEDURE `add_user` (id int, user_name varchar (45))

BEGIN

insert into user_table values(id,user_name);

END

Syntax to call stored procedure:

```
CallableStatement statement=connection.prepareCall("call
add1Car(?,?)");

statement.setInt(1, 10);

statement.setString(2, "Audi");

statement.execute();
```

*****