J2EE

JAVA 2 ENTERPRISED EDITION

Pre-requisites

- Core Java
- SQL
- Web Technologies

TECHNOLOGIES INVOLVED IN THIS J2EE

- JDBC (Java Database Connectivity)
- Servlets
- JSP (Java Server Page)

PUNITH B

J2SE VS J2EE

- **J2SE (Java Standard Edition)**
  - ➢ The core java or J2EE contains only the core fundamentals of JAVA.
  - ➢ Less library content.
  - ➢ Using Standard edition, we can build only standalone application.

- **J2EE (Java Enterprise Edition)**
  - ➢ J2EE is the enterprise edition which is used to desing web applications.
  - ➢ More library content.
  - ➢ Using Enterprise edition, we can build real time and dynamic web application.

- **SERVER:** It is a system which is used to perform that provides a service for the application for accepting client request and providing the client response.

  Ex: MySQL Server(Database), Tomcat Server(Application Server) etc..

- **HOST**: It is a platform which is used to run the server.
  - **Local Host:** The server will be limited to that particular system.
  - **Remote Host**: The server will be in the remote place and it is used to connect to that remote server.

- **Port Number:** It is a way to identify the specific server or to connect to a specific server.

  Ex: Oracle DB server – 1521 , MySQL – 3306, Apache Tomcat - 8080

- **URL (Uniform Resource Locator):** It specifies the address of a particular resource.

In JDBC, the URL pattern is :

main_protocol:sub_protocol://localhost/IP_address:portnumber/database_name?user=username&password=password;

Jdbc:mysql://localhost:3306/dbname?user=root&password=tiger;

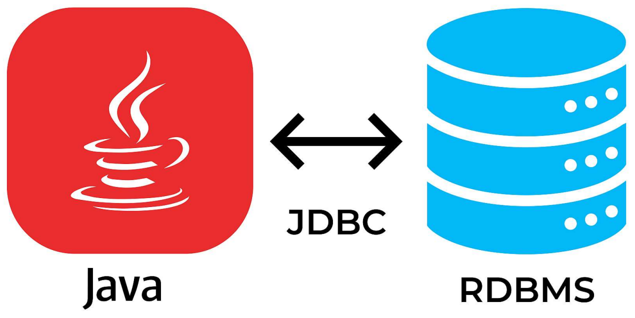
API (APPLICATION PROGRAMMING INTERFACE)

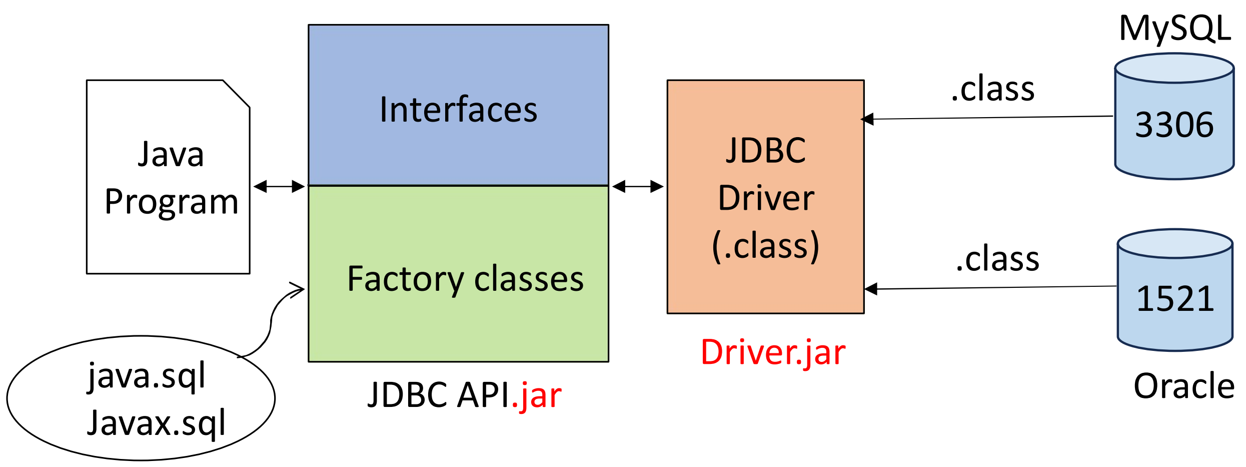
- API is used to establish connection between one application and another application.
- The backbone of API is Abstraction.
- The result of the API is loose coupling.

Ex: JDBC API, Apache Poi, Jexcel etc..

JDBC API

JDBC API is used to establish connection between Java and Database.



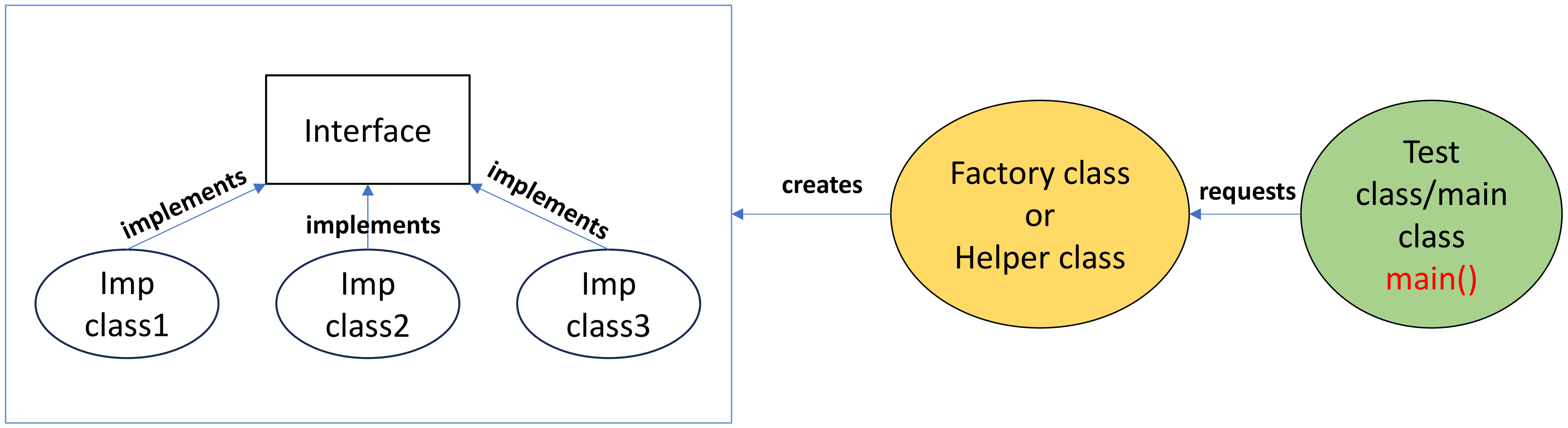


- JDBC API is given by Sun Microsystems and is available in JRE library in the form of jar file.
- JDBC API consists of two packages. They are
  - java.sql
  - javax.sql
- JDBC API consists of Interfaces and Factory classes.
- Implementation classes will be given by respective database servers or venders.
- These implementation classes given by database servers are known as JDBC Drivers.

PUNITH B

FACTORY DESIGN PATTERN

- Factory is used to create multiple objects of same type.
- Factory design pattern consists of three types of logics:
  - Implementation Logic (Interfaces and implementation classes)
  - Object Creational Logic (Factory Class)
  - Customer Utilization Logic (Test class or main class).



```
//Implementation Logic
public interface IPayment
{
   void doPayment();
}

public class UPI implements Ipayment
{
   @Override
   public void doPayment() {
   System.out.println("Payment successful through UPI");
   }
}

public class DebitCard implements Ipayment
{
   @Override
   public void doPayment() {
   System.out.println("Payment successful using Debit Card");
   }
}
```

PUNITH B

```
public class CreditCard implements Ipayment
{
   @Override
   public void doPayment() {
   System.out.println("Payment successful using Credit Card");
   }
}


//Object Creational Logic
public class PaymentMode //Factory or helper class
{
  public static IPayment payment(String in)
  //factory or helper method
  {
    if(in.equalsIgnoreCase("UPI")){
    return new UPI();
    }
    else if(in.equalsIgnoreCase("Credit")){
    return new CreditCard();
    }
    else if(in.equalsIgnoreCase("Debit")){
    return new DebitCard();
    }
    else{
    System.err.println("No such payment option available");
     return null;
    }
  }
}
```

PUNITH B

```java
//Customer Utilization Logic
public class Test {
  public static void main(String[] args) {
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter the Payment mode:");
    String payment_mode=sc.next();
    IPayment ip=PaymentMode.payment(payment_mode);
    if(ip!=null)
    {
      ip.doPayment();
    }
  }
}
```

Output:

```
Enter the Payment mode:
credit
Payment successful using Credit Card

Enter the Payment mode:
cash
No such payment option available
```

PUNITH B

Interfaces that are available in JDBC API

- **Connection:** This interface is used to establish connection between Java and database.
- **Statement:** This interface is used to create a platform for executing the SQL Queries.
- **PreparedStatement:** This interface extends Statement Interface and also used to create a platform for executing DQL queries.
- **ResultSet:** This interface is used to retrieve the data from the resultant table that will be stored in cursor or buffer memory.

Factory class present in JDBC API

The only factory class present in JDBC API is known as **DriverManager.** Basically it is a utility class which is responsible for managing the driver classes given by the database servers or venders.

Note: All these Interfaces and factory classes are available in **java.sql** package.

JDBC Driver / implementation class given by the Database servers or venders

Oracle : OracleDriver **Fully Qualified Class name:** **"Oracle.jdbc.driver.OracleDriver"**
MySQL : Driver **Fully Qualified Class name:** **"com.mysql.cj.jdbc.Driver"**

STEPS OF JDBC

1. **Loading and Registering the Driver class.**
2. **Establishing connection between Java and Database**
3. **Creating a platform for SQL queries.**
4. **Executing the SQL queries.**
5. **Generating the Result (DQL)**
6. **Closing all the costly resources (deprecated).**

1. Loading and Registering the Driver class

In this step, we have to load and register the driver class which is a part of JDBC driver which are provided by respective database servers of venders. We can load the driver by using a static method called as forName(). We can use try and catch block to handle the exception .

Syntax:
```
Class.forName("fully qualified class name of the driver");
 //throws ClassNotFoundException
```

PUNITH B

forName()

- It is a static method present in the class called as Class.
- forName() is used to load the given class i.e, push the .class file to JVM memory.
- Whenever forName() is called, it throws ClassNotFoundException.
- forName() takes fully qualified class name as a argument.

Syntax:
```
Class.forName("fully qualified class name");
 //throws ClassNotFoundException
```

2. Establishing the connection between Java and Database.

- In this step, the connection between Java and database is established by using getConnection().
- getConnection() is a factory or helper method which is used to create a reference of Connection Interface. Hence the return type is
- getConnection() is present in DriverManager class. (Factory Class)
- Whenever getConnection() is called, it throws SQLException.
- getConnection() is a overloaded method and takes URL as a argument is three different way.

```
public static Connection DriverManager.getConnection(String url);
public static Connection DriverManager.getConnection(String url, Properties info);
public static Connection DriverManager.getConnection(String url, String user, String
password);
```

Connection Interface (java.sql.Connection)

- Connection interface is a part of JDBC API and the implementation for this interface is provided by respective database servers or venders.
- It is available in java.sql package.
- The return type of getConnection() is Connection Interface.

Syntax: `Connection con=DriverManager.getConnection("url");`

3. Creation of the platform

- In this step, a platform is created in the java program to execute SQL query.
- A platform can be created either by using Statement Interface or PreparedStatement Interface.

```
Statement extends PreparedStatement
```

- **createStatement() is used to create and return the reference object of type Statement Interface.**
- **createStatement() is avaible in Connection Interface.**

Syntax: `Statement st= con.createStatement();`

4. Execution of Queries

- The different types of SQL queries are:
  - DML (Data Manipulation Language) : INSERT, UPDATE, DELETE
  - DQL (Data Query Language): SELECT
- To execute, the SQL queries we have 3 different methods that are available in Statement Interface and PreparedStatement Interface. They are execute(), executeUpdate(), executeQuery().

execute():

- It is a generic method available in Statement and PreparedStatement Interface.
- It executes both DQL and DML queries.
- The return type of execute() is Boolean.

Syntax:

```
public boolean st.execute(String query);
```

Note: This method returns true when it executes DQL queries and returns false when it executes DML queries.

executeUpdate():

- It is a specific method present n Statement and PreparedStatement Interface.
- This method is only for executing the DML queries.
- The return type of executeUpdate() is int. Because it returns the number of rows inserted, updated or deleted.

Syntax:

```
public int st.executeUpdate(String query);
```

executeQuery():

- It is a specific method present in Statement and PreparedStatement Interface.
- This method is only for executing the DQL queries.
- The return type of executeQuery() is ResultSet.
- It returns the reference object of type ResultSet Interface.

Syntax:

```
ResultSet rs = st.executeQuery(String query);
```

ResultSet Interface

- Normally, the results we get from executing the DQL queries are known as Result sets.
- These Result sets will be stored in cursor or buffer memory.
- To get the data from cursor or buffer memory, ResultSet Interface can be used.
- ResultSet reference object is created by using the executeQuery().
- A set of methods of ResultSet interface to fetch the DQL processed data or resultant data from cursor memory is getXXX().

Syntax:

```
public XXX rs.getXXX(int column_index);
public XXX rs.getXXX(String column_name);

public int rs.getInt(1);
public String rs.getString("fname");
```

next()

- next() is an inbuilt method present in the ResultSet Interface.
- It is used to check whether the next record is present in the cursor or buffer memory or not.
- It returns Boolean value but not the record.

Syntax:

```
public boolean rs.next() throws SQLException;
```

Drawbacks of Statement Interface

- Whenever we pass the query inside the loop for multiple times of execution, each time the query is sent to database and compliled first, then executed, the result is returned to the program. The problem with this kind of execution is it decreases the performance of the program.
- SQL Injection attack chances are high.

SQL Injection

- In the Req 2 program, the malicious user will not be knowing the input to be given but 1=1 is always true. Hence, the hacker is able to steal the data of all the user.
- This type of SQL injection attacks are possible with Statement Interface.
- To prevent SQL Injection, PreparedStatement Interface can be used.

Ex: Consider the below query:

```
"SELECT * FROM EMP WHERE FNAME=" 'SIDDARTH'
```

As the hacker doesnot know the input to be given, he just injects a true condition inside the query.

```
"SELECT * FROM EMP WHERE FNAME="'unknown' OR 1=1;
```

PUNITH B

PreparedStatement Interface

- PreparedStatement is used to create a platform for executing the SQL queries.
- PreparedStatement pre compile the SQL query for once, and executed multiple times.
- Whenever we are dealing with the user defined values, we can go for PreparedStatment.
- PreparedStatement improves the performance of the program and avoids SQL Injection.
- We can make use of PreparedStatement to store images etc..
- PreparedStatement reference is created by a helper method called as prepareStatement() which is present in the Conenction Interface.
- It is available inside java.sql package.
- We use placeholders to set the user defined values.

Syntax:

```
PreparedStatement ps=con.prepareStatement(String query) throws SQLException;
```

prepareStatment()

- prepareStatement() is used to create the reference object of PreparedStatement.
- It is available inside Connection Interface.
- It takes query as a argument in the form of String.

Note: Incase of PreparedStatement Interface, the execute(),executeUpdate() and executeQuery() which is also present inside PreparedStatement doesnot take query as a argument.

Placeholders

- Placeholders are used to pass the user defined values for the given query.
- These placeholders reserves a place for the data which will be given by the user.
- Placeholders are represented by '?'.
- Each placeholder holds a single data.
- For placeholders, the values are set by using a pre defined method setXXX() which is present in PreparedStatement Interface.

setXXX()

- It is a predefined method present in PreparedStatement Interface.
- It is used to set the value for the placeholders.
- It takes parameter index(placeholder number) and the value(data to be assigned to placeholder) as a argument.
- setXXX() varies according to the type of data to be set for the placeholder.

Syntax:

```
void setXXX(int parameterIndex, XXX value) throws SQLException;
```

JDBC Transaction

- It is a single business unit.
- Multiple queries or transactions are written inside a single program.
- The backbone of JDBC transaction is ACID properties.

commit()

It is a predefined method which is present in Connection Interface.
It is used to permanently store the data inside the database.

```
void commit() throws SQLException;
```

Note: To enable or disable the auto-commit, we use setAutoCommit() which is present in Conenction interface.

```
void setAutoCommit(boolean autoCommit) throws SQLException;
```

rollback()

It is a predefined method which is present in Connection Interface.
It is used to rollout all the transactions until the previously used commit statement.

```
void rollback();
void rollback(Savepoint savepoint_name) throws SQLException;
```

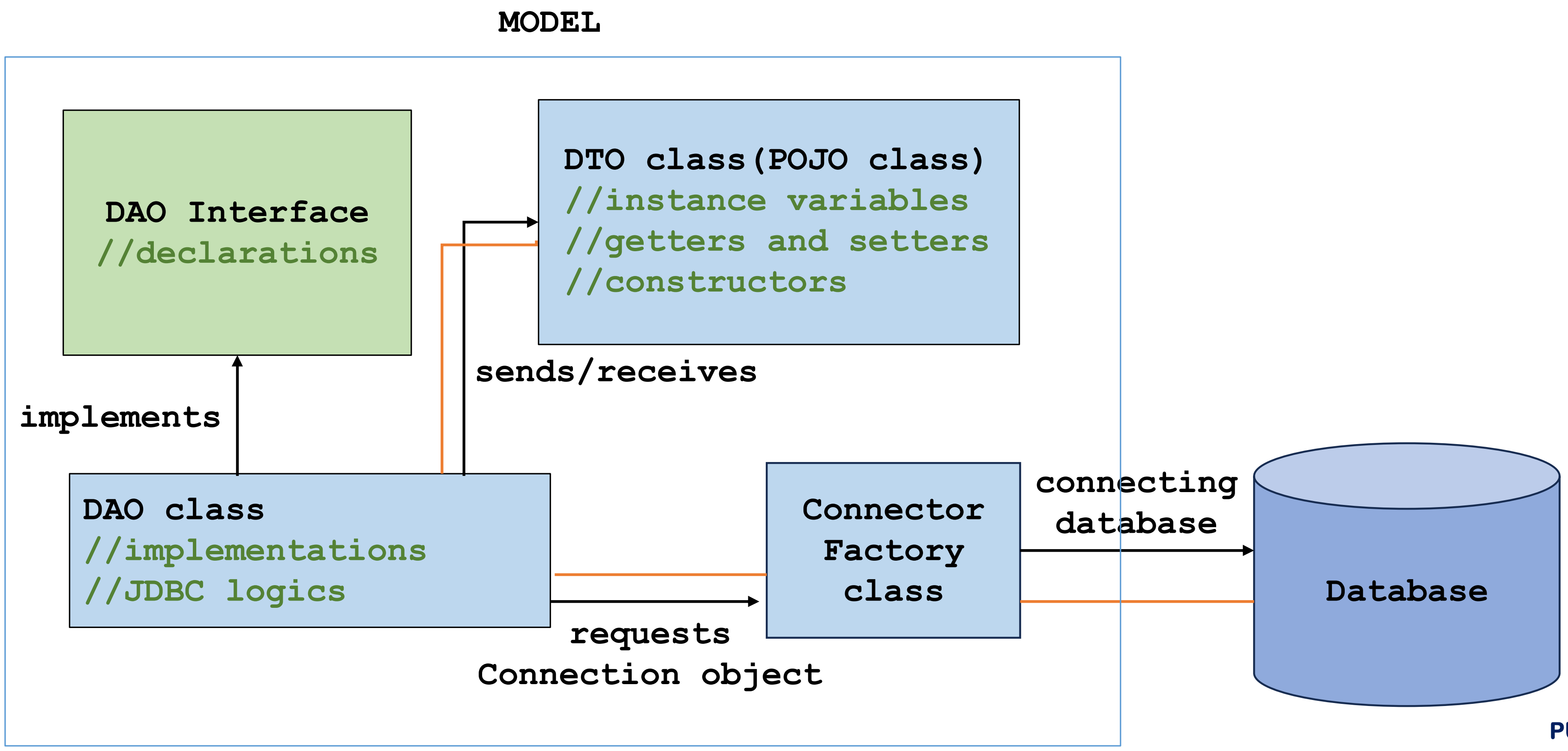
Savepoint

This interface is used to maintain and manage the savepoints.
To set the savepoint,

```
Savepoint setSavepoint(String name) throws SQLException;
String getSavepointName() throws SQLException;
```

DAO PATTERN

- DAO patterns are basically used to access the data in the form of object.
- It consists of DTO class (POJO class), DAO class (JDBC implementation).



PUNITH B