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);
```

PUNITH B

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();
```