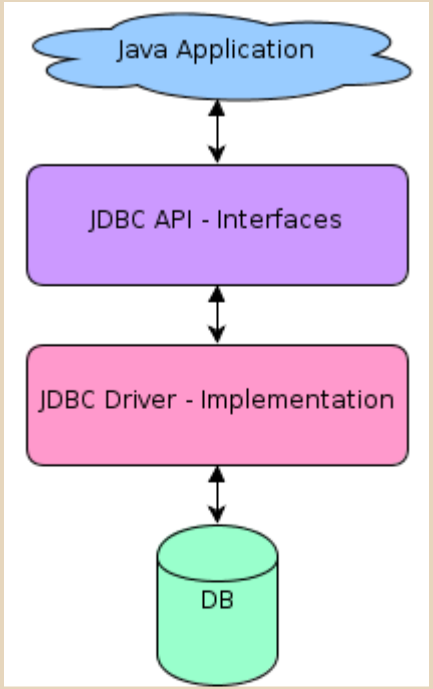
JDBC

JDBC

- Nonofficial acronym for **java database connectivity.**
- DBC helps us to connect to a database and execute SQL statements against a database.
- JDBC api provides set of interfaces and there are different implementations respective to different databases.
- JDBC was first introduced as part of JDK 1.1.
- Using JDBC in our application can be segregated into three major steps as below:
  - Obtaining a database connection.
  - Execute queries against the connected database and receive results.
  - Process the received results.

JDBC



- Java application developers program using the jdbc interfaces and database developer provide the implementation for the jdbc interfaces.
- We add the respective implementation to the application and using the corresponding database.
- A classic use of java interfaces.

Drivers

- The first thing we should know is the Driver class to use.
- Driver Class:
    - MySql
        - com.mysql.jdbc.Driver
    - Oracle
        - oracle.jdbc.driver.OracleDriver
    - You will find these driver classes in their respective driver jar files and both of these implement java.sql.Driver interface.

Drivers

- The second important part is the database connection URL string.
- Every database driver has it’s own way to configure the database URL but all of them have **host**, **port** and **Schema** details in the connection URL.
- For **MySQL** connection String format is: jdbc:mysql://<HOST>:<PORT>/<SCHEMA>
- For **Oracle** database connection string format is jdbc:oracle:thin:@<HOST>:<PORT>:<SID>
- The other important details are database **username** and **password** details to be used for connecting to the database.

Program Detail

- Load the JDBC driver and using DriverManager to create the connection.
- Code use only Java JDBC API classes and there is no way to know that it’s connecting to which type of database.
- This is also a great example of writing code for interfaces methodology.
- The important thing to notice is the **Class.forName()** method call, this is the **Java Reflection** method to create the instance of the given class.
- You might wonder why we are using Reflection and not new operator to create the object and why we are just creating the object and not using it.
- The first reason is that using reflection to create instance helps us in writing loosely-coupled code that we can’t achieve if we are using new operator.
- In that case, we could not switch to different database without making corresponding code changes.

Program Detail

- The reason for not using the object is because we are not interested in creating the object. The main motive is to load the class into memory, so that the driver class can register itself to the DriverManager.
- This is a great example where we are making our code loosely-coupled with the use of reflection API. So basically we are doing following things using Class.forName() method call.

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

- **DriverManager.getConnection()** method uses the registered JDBC drivers to create the database connection and it throws **java.sql.SQLException** if there is any problem in getting the database connection.

Statement vs Prepared Statement

- We can use **Statement** or **PreparedStatement** to execute queries.
- JDBC Statement has some major issues and should be avoided in all cases.
- Statement is easy to use but it can lead to SQL injection, that is very common way of hacking any application.
- That is why we should use PreparedStatement to avoid SQL injection attacks.
- Major benefits we get from using PreparedStatement over Statement such as caching, object oriented programming and elegant looking code.

Prepared Statement

- Preventing SQL injection attacks because it automatically escapes the special characters.
- Allows to execute dynamic queries with parameter inputs.
- Faster than Statement.
- Helps us in writing object Oriented code with setter methods whereas with Statement we have to use String Concatenation to create the query. If there are multiple parameters to set, writing Query using String concatenation looks very ugly and error prone.
- One of the limitation of PreparedStatement is that we can’t use it for SQL queries with IN clause because PreparedStatement doesn’t allow us to bind multiple values for single placeholder (?).

Batch Processing

- Sometimes we need to run **bulk queries** of similar kind for a database, for example loading data from CSV files to relational database tables.
- Apart from **Statement** and **PreparedStatement** JDBC API provides **Batch Processing** feature through which we can execute bulk of queries in one go for a database.
- JDBC API supports batch processing through Statement and PreparedStatement **addBatch**() and **executeBatch**() methods.
- CallableStatement
- Transaction Management with SavePoint