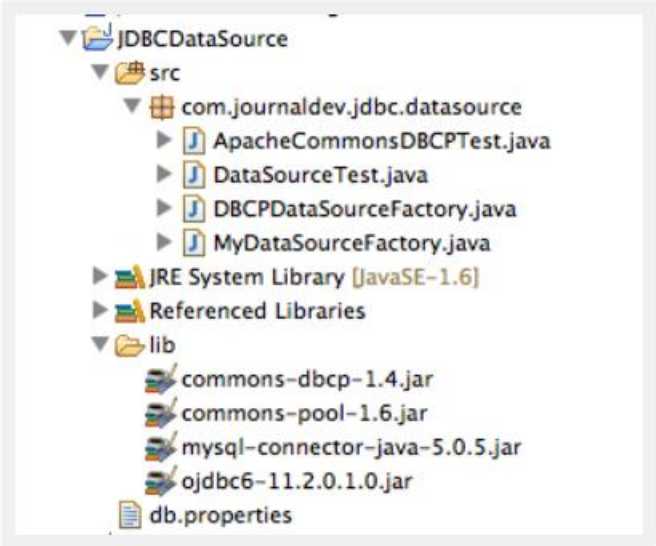
JDBC DataSource Example

Let's create a simple JDBC project and learn how to use MySQL and Oracle DataSource basic implementation classes to get the database connection.
Our final project will look like below image.



- Database Setup

Before we get into our example programs, we need some database setup with table and sample data. (Installation of MySQL and ORACLE database must be done.)

MySQLSetup.sql

```sql
--Create Employee table
CREATE TABLE `Employee` (
  `empId` int(10) unsigned NOT NULL,
  `name` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`empId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- insert some sample data
INSERT INTO `Employee` (`empId`, `name`)
VALUES
    (1, 'Pankaj'),
    (2, 'David');

commit;
```

OracleSetup.sql

```
CREATE TABLE "EMPLOYEE"
  (
    "EMPID"   NUMBER NOT NULL ENABLE,
    "NAME"    VARCHAR2(10 BYTE) DEFAULT NULL,
    PRIMARY KEY ("EMPID")
  );

Insert into EMPLOYEE (EMPID,NAME) values (10,'Pankaj');
Insert into EMPLOYEE (EMPID,NAME) values (5,'Kumar');
Insert into EMPLOYEE (EMPID,NAME) values (1,'Pankaj');
commit;
```

Now let's move on to our java programs. For having database configuration loosely coupled, I will read them from property file.

db.properties

```
#mysql DB properties
MYSQL_DB_DRIVER_CLASS=com.mysql.jdbc.Driver
MYSQL_DB_URL=jdbc:mysql://localhost:3306/UserDB
MYSQL_DB_USERNAME=pankaj
MYSQL_DB_PASSWORD=pankaj123

#Oracle DB Properties
ORACLE_DB_DRIVER_CLASS=oracle.jdbc.driver.OracleDriver
ORACLE_DB_URL=jdbc:oracle:thin:@localhost:1521:orcl
ORACLE_DB_USERNAME=hr
ORACLE_DB_PASSWORD=oracle
```

Make sure that above configurations match with your local setup. Also make sure you have MySQL and Oracle DB JDBC jars included in the build path of the project.

➢ JDBC MySQL and Oracle DataSource Example

Let's write a factory class that we can use to get MySQL or Oracle DataSource.

```
import java.io.FileInputStream;
import java.io.IOException;
import java.sql.SQLException;
import java.util.Properties;

import javax.sql.DataSource;

import oracle.jdbc.pool.OracleDataSource;

import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;
```

```
14  public class MyDataSourceFactory {
15
16      public static DataSource getMySQLDataSource() {
17          Properties props = new Properties();
18          FileInputStream fis = null;
19          MysqlDataSource mysqlDS = null;
20          try {
21              fis = new FileInputStream("db.properties");
22              props.load(fis);
23              mysqlDS = new MysqlDataSource();
24              mysqlDS.setURL(props.getProperty("MYSQL_DB_URL"));
25              mysqlDS.setUser(props.getProperty("MYSQL_DB_USERNAME"));
26              mysqlDS.setPassword(props.getProperty("MYSQL_DB_PASSWORD"));
27          } catch (IOException e) {
28              e.printStackTrace();
29          }
30          return mysqlDS;
31      }
33      public static DataSource getOracleDataSource(){
34          Properties props = new Properties();
35          FileInputStream fis = null;
36          OracleDataSource oracleDS = null;
37          try {
38              fis = new FileInputStream("db.properties");
39              props.load(fis);
40              oracleDS = new OracleDataSource();
41              oracleDS.setURL(props.getProperty("ORACLE_DB_URL"));
42              oracleDS.setUser(props.getProperty("ORACLE_DB_USERNAME"));
43              oracleDS.setPassword(props.getProperty("ORACLE_DB_PASSWORD"));
44          } catch (IOException e) {
45              e.printStackTrace();
46          } catch (SQLException e) {
47              e.printStackTrace();
48          }
49          return oracleDS;
50      }
51
52  }
```

Notice that both Oracle and MySQL DataSource implementation classes are very similar, let's write a simple test program to use these methods and run some test.

```
3  import java.sql.Connection;
4  import java.sql.ResultSet;
5  import java.sql.SQLException;
6  import java.sql.Statement;
7
8  import javax.sql.DataSource;
9
10 public class DataSourceTest {
11
12     public static void main(String[] args) {
13
14         testDataSource("mysql");
15         System.out.println("**********");
16         testDataSource("oracle");
17
18     }
19
20      private static void testDataSource(String dbType) {
21          DataSource ds = null;
22          if("mysql".equals(dbType)){
23              ds = MyDataSourceFactory.getMySQLDataSource();
24          }else if("oracle".equals(dbType)){
25              ds = MyDataSourceFactory.getOracleDataSource();
26          }else{
27              System.out.println("invalid db type");
28              return;
29          }
30
31          Connection con = null;
32          Statement stmt = null;
33          ResultSet rs = null;
34          try {
35              con = ds.getConnection();
36              stmt = con.createStatement();
37              rs = stmt.executeQuery("select empid, name from Employee");
38              while(rs.next()){
                    System.out.println("Employee ID="+rs.getInt("empid")+
                                       ", Name="+rs.getString("name"));
40              }
41          } catch (SQLException e) {
42              e.printStackTrace();
43          }finally{
44                  try {
45                      if(rs != null) rs.close();
46                      if(stmt != null) stmt.close();
47                      if(con != null) con.close();
48                  } catch (SQLException e) {
49                      e.printStackTrace();
50                  }
51          }
52      }
53
54 }
```

Notice that the client class is totally independent of any Database specific classes. This helps us in hiding the underlying implementation details from client program and achieve loose coupling and abstraction benefits.
When we run above test program, we will get below output.

```
Employee ID=1, Name=Pankaj
Employee ID=2, Name=David
**********
Employee ID=10, Name=Pankaj
Employee ID=5, Name=Kumar
Employee ID=1, Name=Pankaj
```

- If you look at above DataSource factory class, there are two major issues with it.

1. The factory class methods to create the MySQL and Oracle DataSource are tightly coupled with respective driver API. If we want to remove support for Oracle database in future or want to add some other database support, it will require code change.
2. Most of the code to get the MySQL and Oracle DataSource is similar, the only different is the implementation class that we are using.

Apache Commons DBCP API helps us in getting rid of these issues by providing DataSource implementation that works as an abstraction layer between our program and different JDBC drivers.
Apache DBCP library depends on Commons Pool library, so make sure they both are in the build path as shown in the image.
Here is the DataSource factory class using BasicDataSource that is the simple implementation of DataSource.

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

import javax.sql.DataSource;

import org.apache.commons.dbcp.BasicDataSource;

```

```
11  public class DBCPDataSourceFactory {
12
13      public static DataSource getDataSource(String dbType){
14          Properties props = new Properties();
15          FileInputStream fis = null;
16          BasicDataSource ds = new BasicDataSource();
17
18          try {
19              fis = new FileInputStream("db.properties");
20              props.load(fis);
21          }catch(IOException e){
22              e.printStackTrace();
23              return null;
24          }
25          if("mysql".equals(dbType)){
26              ds.setDriverClassName(props.getProperty("MYSQL_DB_DRIVER_CLASS"))
27              ds.setUrl(props.getProperty("MYSQL_DB_URL"));
28              ds.setUsername(props.getProperty("MYSQL_DB_USERNAME"));
29              ds.setPassword(props.getProperty("MYSQL_DB_PASSWORD"));
30          }else if("oracle".equals(dbType)){
31              ds.setDriverClassName(props.getProperty("ORACLE_DB_DRIVER_CLASS")
32              ds.setUrl(props.getProperty("ORACLE_DB_URL"));
33              ds.setUsername(props.getProperty("ORACLE_DB_USERNAME"));
34              ds.setPassword(props.getProperty("ORACLE_DB_PASSWORD"));
35          }else{
36              return null;
37          }
38
39          return ds;
40      }
41  }
```

As you can see that depending on user input, either MySQL or Oracle datasource is created. If you are supporting only one database in the application then you don't even need these logic. Just change the properties and you can switch from one database server to another. The key point through which Apache DBCP provide abstraction is *setDriverClassName()* method.
Here is the client program using above factory method to get different types of connection.

```
3   import java.sql.Connection;
4   import java.sql.ResultSet;
5   import java.sql.SQLException;
6   import java.sql.Statement;
7
8   import javax.sql.DataSource;
9
10  public class ApacheCommonsDBCPTest {
11
12      public static void main(String[] args) {
13          testDBCPDataSource("mysql");
14          System.out.println("**********");
15          testDBCPDataSource("oracle");
16      }
```

```
18      private static void testDBCPDataSource(String dbType) {
19          DataSource ds = DBCPDataSourceFactory.getDataSource(dbType);
20
21          Connection con = null;
22          Statement stmt = null;
23          ResultSet rs = null;
24          try {
25              con = ds.getConnection();
26              stmt = con.createStatement();
27              rs = stmt.executeQuery("select empid, name from Employee");
28              while(rs.next()){
                        System.out.println("Employee ID="+rs.getInt("empid")+",
Name="+rs.getString("name"));
31          } catch (SQLException e) {
32              e.printStackTrace();
33          }finally{
34                  try {
35                      if(rs != null) rs.close();
36                      if(stmt != null) stmt.close();
37                      if(con != null) con.close();
38                  } catch (SQLException e) {
39                      e.printStackTrace();
40                  }
41          }
42      }
43
44  }
```

When you run above program, the output will be same as earlier program.
If you look at the DataSource and above usage, it can be done with normal DriverManager too. The major benefit of DataSource is when it's used within a Context and with JNDI.
With simple configurations we can create a Database Connection Pool that is maintained by the Container itself. Most of the servlet containers such as Tomcat and JBoss provide it's own DataSource implementation and all we need is to configure it through simple XML based configurations and then use JNDI context lookup to get the DataSource and work with it.