

**Instructor Notes:**

Add instructor notes here.



**Instructor Notes:**

Explain the lesson coverage

## Lesson Objectives



In this lesson, we will learn:

- Configuring a Database for Web Applications

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

### Lesson Objectives:

This lesson introduces database configuration in WildFly.

The lesson contents are as follows:

## Lesson 06: Configurinag Database in WildFly

## 6.1: Database configuration in WildFly

**Instructor Notes:**

Explain different ways data can be persisted

## 6.1: Database Configuration in WildFly Persistence Options in WildFly



Persistence tier is also sometimes referred to as database tier.

Data can be stored in either of the following:

- Flat files
- XML files
- Databases

Databases are the most trustworthy option for persisting data.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

3

### Persistence Options in WildFly:

Till now we have focused on designing simple web application, the focus was on the web tier. However, for any web application, the data storage is an integral part.

Now, let us move to the persistence tier, which is also sometimes referred to as database tier. This is where application data is stored for a long term / persistently.

While working with JEE applications, data can be stored in Flat files, XML files, or Database. Databases are the most trustworthy options for persisting data.

Here, we all need to accept one simple fact. While dealing with relational databases in a JEE application, all paths some way or the other eventually lead to JDBC.

We are already familiar with JDBC.

**Instructor Notes:**

Make the participants recall the JDBC code used in Module 2, to obtain the connection object

## 6.1: Database Configuration in WildFly

### Working with Typical JDBC



#### Code Snippet

```
Connection dbCon;  
String url = "jdbc:oracle:thin:@192.168.67.177:1521:trgdb";  
public void init() throws ServletException {  
    try {  
        DriverManager.registerDriver (new  
            oracle.jdbc.OracleDriver());  
        dbCon =  
            DriverManager.getConnection(url,"user1","user1");  
    } catch (Exception e) {  
        throw e;  
    }  
  
    Statement s = dbCon.createStatement();  
    .....  
}
```

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

4

#### Working with Typical JDBC:

In the previous chapter (Request-Response-DBServlet), already typical JDBC code in a Servlet is shown to interact with the database. The above slide shows a snippet of the code.

While this code definitely works, it does have some drawbacks:

Every time that we connect and disconnect from the database, the physical database connection is created and destroyed. This leads to an overhead.

If there are any transactions related to the applications, then they need to be managed by developer.

If the application deals with only local transactions within the database, it is fine.

However, in case it interacts with other resources, then managing the transactions and other aspects is difficult within the application.

**Instructor Notes:**

Take the participants back to their core java and remind them about the differences between using Driver manager versus using DataSource.. Once again remind them about the advantages of using datasource over typical JDBC coding

## 6.1: Database Configuration in WildFly

### Working with Data Source



#### Code Snippet

```
try {
    InitialContext ic = new InitialContext();
    DataSource ds = (DataSource)
        ic.lookup("java:/jdbc/TestDS");
    Connection con = ds.getConnection();
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("SELECT * FROM
    emp824177");
    while (rs.next()) {
        pw.println(rs.getInt("ID") + "<br>");
        pw.println(rs.getString("NAME") + "<br>");
        pw.println(rs.getFloat("SALARY") + "<br>");
    }
} catch (NamingException | SQLException e) {
    log("Error is:::::::::" + e.getMessage());
}
```

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

5

#### Working with DataSource:

While working with an application server, gives a lot of benefits. In the previous slide, we mentioned a few drawbacks with the typical JDBC approach of programming in a web application. However, these issues can be taken care of very easily in an Application Server like WildFly.

The above slide makes use of datasource, which is the change that is required to obtain Database connections. Instead of importing java.sql.DriverManager, import javax.sql.DataSource. Furthermore, since we would be using JNDI naming for looking up the DataSource, need to import javax.naming.InitialContext and javax.naming.NamingException.

Using the DataSource has its own advantages:

When obtaining a Database connection using DataSource, no need to create a new connection. During startup WildFly creates a DB connection pool managed by DataSource. So essentially when accessing an existing connection from the pool and when the connection is closed, it is just returned back to the pool so someone else can use it.

When we use Container Managed DataSource (In this case JBoss Server Managed), all database access for a particular Transaction commits or rollback automatically. No need to manage the transaction anymore.

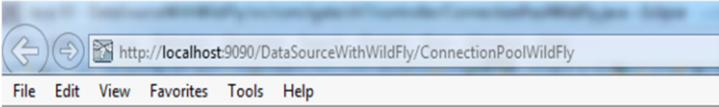
Let us now take some time and understand the code.

Instructor Notes:

Some points to mention:  
Make sure the JNDI names are exactly the same.  
The slash has to be a forward slash in the Application code of the data source context that is used in the application

Demo

Configuring Database in WildFly.  
Execute the ConnectionPoolWildFly.java class



Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

6

Note:  
Steps to configure the database for your application in WildFly  
Database Configuration:

Refer to Lab Book: Steps to configure Database connection pool in WildFly in Lab 4:

**Instructor Notes:**

This is partial listing of standalone.xml file specifying the connection Pool "TestDS" being created

Refer to the Connection Pooling with WildFly 8 in lab book file to configure the Connection Pool

**Demo****Partial listing of standalone.xml**

```
<datasource jta="false" jndi-name="java:/jdbc/TestDS" pool-
name="TestDS" enabled="true" use-ccm="false">
  <connection-
url>jdbc:oracle:thin:@localhost:1522:XE</connection-url>
  <driver-class>oracle.jdbc.OracleDriver</driver-class>
  <driver>ojdbc6_g.jar</driver>
  <security>
    <user-name>system</user-name>
    <password>igate</password>
  </security>
  <validation>
    <validate-on-match>false</validate-on-match>
    <background-validation>false</background-
validation>
  </validation>
  <statement>
    <share-prepared-statements>false</share-prepared-
statements>
  </statement>
</datasource>
```

**Instructor Notes:**

Instead of using JNDI name with InitialContext , we could also use @Resource annotation

## 6.1: Database Configuration in WildFly

### Working with Data Source - @Resource Annotation



The @Resource annotation is used to declare a reference to a resource, such as a data source

The @Resource annotation is specified on a class, a method, or a field

The container is responsible for injecting references to resources declared by the @Resource annotation and mapping it to the proper JNDI resources

Resources can be injected only into container-managed objects

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

8

@Resource annotation is used to inject a data source into a component that needs to make a connection to the data source, as is done when using JDBC technology to access a relational database:

The container injects this data source prior to the component's being made available to the application. The data source JNDI mapping is inferred from the field name (example: TestDS) and the type, javax.sql.DataSource.

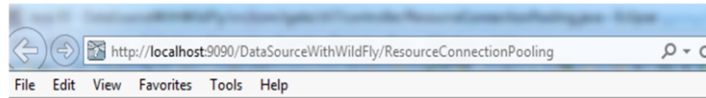
Container managed objects are the web components (Servlets / JSPs).Resources can be injected only into container-managed objects. Resource injection is convenient for the developer. Since a container must have control over the creation of a component it can perform the injection into a component. As a result, resources cannot be injected into objects such as simple JavaBeans components



**Instructor Notes:****Demo**

Configuring Database in WildFly.

Execute the ResourceConnectionPooling.java class



```
1
Rahul
50000.0
2
Anju
90000.0
Num of Records:::2
```

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

9

Execute the following servlet:

<http://localhost:9090/DataSourceWithWildFly/ResourceConnectionPoolig>

Partial listing of code is given below: which applies the field – level annotation

```
@Resource(lookup="java:/jdbc/TestDS")
DataSource ds;
```

```
try {
    Connection con = ds.getConnection();
    -----
} catch (SQLException e1) {
    log("Error is:::" + e1.getMessage());
}
```

## Summary



In this lesson, we have learnt:

- Configuring Database in WildFly

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

10

Add the notes here.

**Instructor Notes:**

Answers for the  
Review Questions:

Answer 1:  
`javax.sql.DataSource`  
e

Answer 2: True

## Review Questions



Question 1: The package to be imported for using  
DataSource is \_\_\_\_.

Question 2: The Datasource lookup happens in  
standalone.xml

- True/ False

Add the notes here.