GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

# Unit 4. ACCESS USING COMPONENTS

## Part 2. JavaBeans Advanced

### Acceso a Datos (ADA) (a distancia en inglés)

### CFGS Desarrollo de Aplicaciones Multiplataforma (DAM)

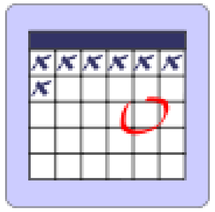
**Abelardo Martínez**

**Year 2023-2024**

# Credits

- Notes made by Abelardo Martínez.
- Based and modified from Sergio Badal (www.sergiobadal.com).
- The images and icons used are protected by the LGPL licence and have been obtained from:
  - https://commons.wikimedia.org/wiki/Crystal_Clear
  - https://www.openclipart.org

# Unit progress



| | UNIT 4: ACCESS TO DATABASES USING COMPONENTS | | | |
|---|---|---|---|---|
| 22/01/24 | UNIT 4 | WEEK 1 | JAVABEANS BASIC | |
| 29/01/24 | UNIT 4 | WEEK 2 | JAVABEANS ADVANCED | AT4.PRESENTATION |
| 05/02/24 | UNIT 4 | WEEK 3 | UNIT 4 REVIEW | |
| 12/02/24 | UNIT 4 | WEEK 4 | UNIT 3 AND UNIT 4 REVIEW | AT4.SUBMISSION |
| 19/02/24 | AT4 Oral Interview | | | |
| 26/02/24 | CONTENTS REVIEW | | | |
| 04/03/24 | ORDINARY EXAM | | | |
| ... | | | | |
| 06/05/24 | EXTRAORDINARY EXAM | | | |
| 20/05/24 | | | | |

# Contents



# 1. ADDING DATABASE ACCESS

# Auxiliary Beans

When working with JavaBeans, we can add several beans to implement access to different types or data, services and resources. We'll create a new bean to connect MySQL database using the old fashion way (JDBC), and we'll be packaging this additional bean with the other ones in a single JAR file.

```
static final String CONN_URL = "jdbc:mysql://localhost:3306/" + DB_NAME
        + "?useSSL=false&useTimezone=true&serverTimezone=UTC&allowPublicKeyRetrieval=true";
private Connection connDB = null;

public DBBean() {
    try {
        // This will load the MySQL driver, each DB has its own driver
        Class.forName("com.mysql.cj.jdbc.Driver");
        // Setup the connection with the DB
        connDB = DriverManager.getConnection(CONN_URL,DB_USER,DB_PASSWORD);

        System.out.println("Connected to the database.");
    } catch (ClassNotFoundException | SQLException sqle) {
        System.out.println("Got an exception (connecting)!");
        System.out.println(sqle.getMessage());
    }
}
```

For further information:
https://www.javatpoint.com/example-to-connect-to-the-mysql-database

# Accessing the database

Once the new bean is working properly, the new package will be imported to the main class so the beans can not only interact, but to connect with a running database.

```
public static void insertOrder(ProductBean objProductBean, int iAmount) {
    System.out.println("Inserting...");
    try {
        //create a new connection to the DB
        DBBean objDBBean = new DBBean();

        // the MySQL insert statement
        String stSQLquery = " INSERT INTO orders (idp, amount)" + " VALUES (?, ?)";
        // create the MySQL insert prepared statement
        PreparedStatement preparedStmt = objDBBean.connDB.prepareStatement(stSQLquery);
        preparedStmt.setInt(1, objProductBean.getiProductid());
        preparedStmt.setInt(2, iAmount);

        // execute the prepared statement
        preparedStmt.execute();

        //close DB connection
        objDBBean.closeDBConnection();
        //show information about the new created order
        System.out.println("Order inserted (idp=" + objProductBean.getiProductid()
                + " amount=" + iAmount + ")");
    } catch (SQLException sqle) {
        System.out.println("Got an exception (inserting)!");
        System.out.println(sqle.getMessage());
    }
}
```

# 2. ADVANCED EXAMPLE

# Changes needed

The best way to understand what do we need to do is to see first what we want to achieve. Have a look to the main class and to the desired output.

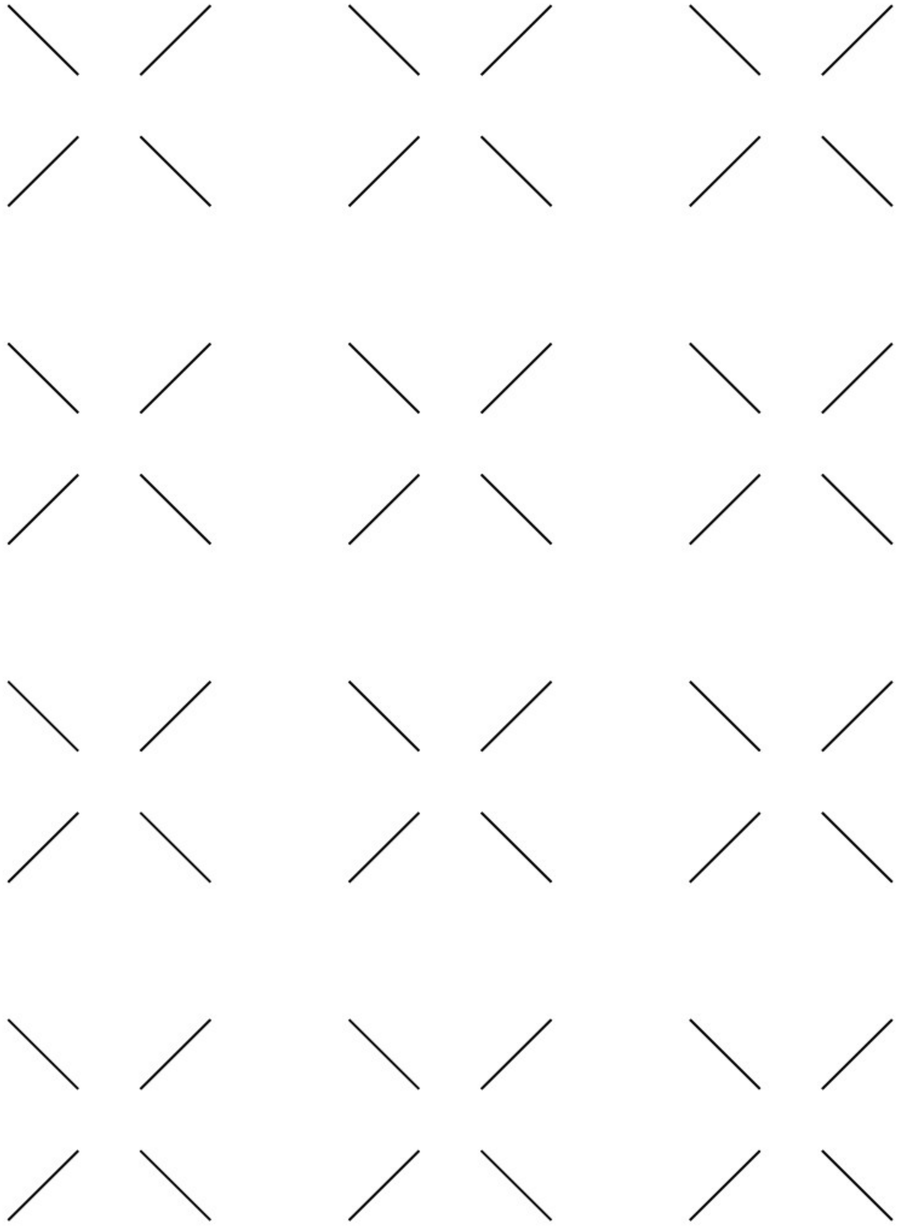
## Main class (defined later)

```
public class TestADAbeans {

    public static void main(String[] stArgs) {
    // ProductBean(int iProductid, String stDescription, float fPrice, int iCurrentstock, int iMinstock)
    // Setting currentStock to 101 units and minimumStock to 100 units
        ProductBean objProductBean = new ProductBean(1, "Robot hoover", 399, 101, 100);
        OrderBean objOrderBean = new OrderBean();
        objOrderBean.setobjProductBean(objProductBean);

        objProductBean.addPropertyChangeListener(objOrderBean);
    // Setting currentStock to 40 (below the minimum advisable)
        System.out.println("****** product.setCurrentStock(40):");
        objProductBean.setiCurrentStock(40);
    // Setting minimumStock to 50 (over the current stock)
        System.out.println("****** product.setMinStock(50):");
        objProductBean.setiMinStock(50);
    }
}
```

## Output

```
****** product.setCurrentStock(40):
[OrderBean says... ]
Current stock is now less than minimum stock!
=> Old current Stock: 101
=> New current Stock: 40
It will place an order for this product: Robot hoover
****** product.setMinStock(50):
[OrderBean says... ]
Minimum stock is now greater than current stock!
Old minstock Stock: 100
New minstock Stock: 50
It will place an order for this product: Robot hoover
```

## 2.1 Step 1. Connecting to the DB



# STEP 1: Creating the DB (1 of 2)

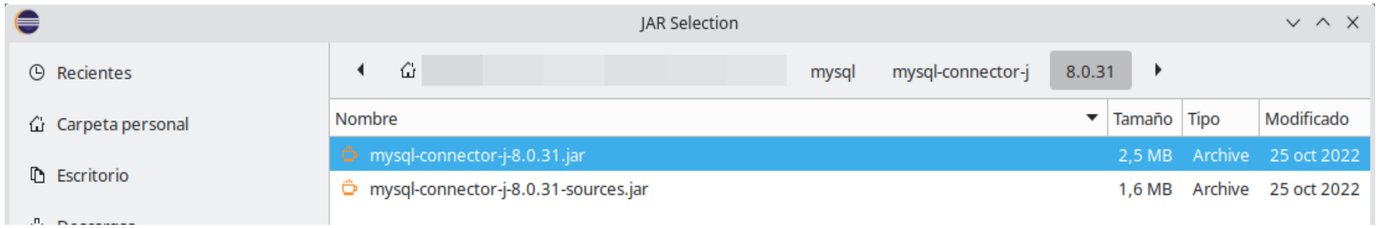
```
CREATE DATABASE DBProducts;
-- Create a user if you are using MySQL 5.7 or MySQL 8 or newer
CREATE USER 'mavenuser'@'localhost' IDENTIFIED WITH mysql_native_password BY 'ada0486';
GRANT ALL PRIVILEGES ON DBProducts.* TO 'mavenuser'@'localhost';
USE DBProducts;

CREATE TABLE products (
idp     INTEGER,
name    VARCHAR(20),
CONSTRAINT pro_idp_pk PRIMARY KEY (idp)
);

INSERT INTO products VALUES (1, "Duruss Cobalt");
INSERT INTO products VALUES (2, "Varlion Avant Carbon");
INSERT INTO products VALUES (3, "Star Vie Pyramid R50");
INSERT INTO products VALUES (4, "Dunlop Titan");
INSERT INTO products VALUES (5, "Vision King jm");
INSERT INTO products VALUES (6, "Slazenger Reflex Pro");

CREATE TABLE orders (
ido     INTEGER AUTO_INCREMENT,
idp     INTEGER,
amount  INTEGER,
CONSTRAINT ord_ido_pk PRIMARY KEY (ido),
CONSTRAINT ord_idp_fk FOREIGN KEY (idp) REFERENCES products(idp)
);
```

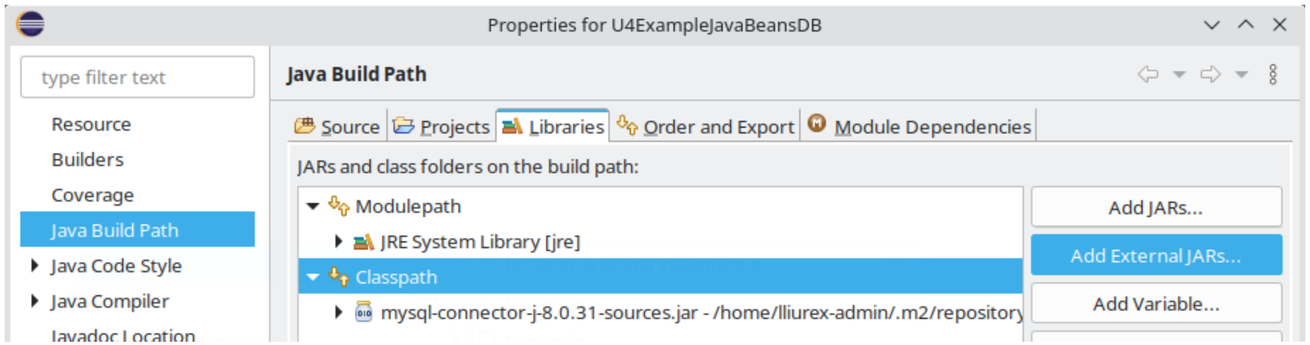
This is the database we’re using for this example.

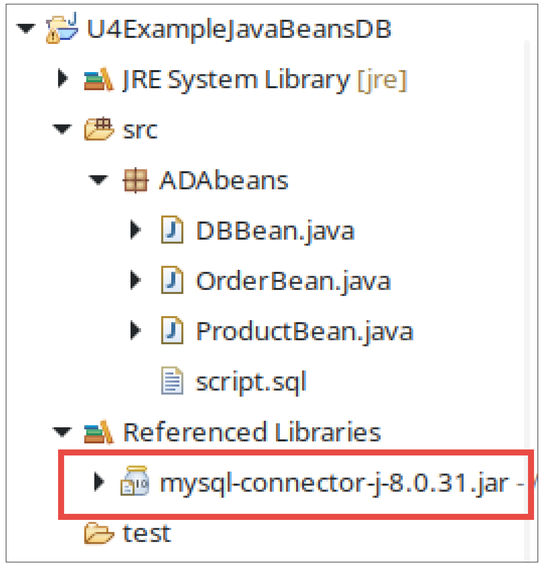
# STEP 1: Including the JDBC driver  (2 of 2)

We must include the JDBC driver into the classpath to have access to MySQL database.

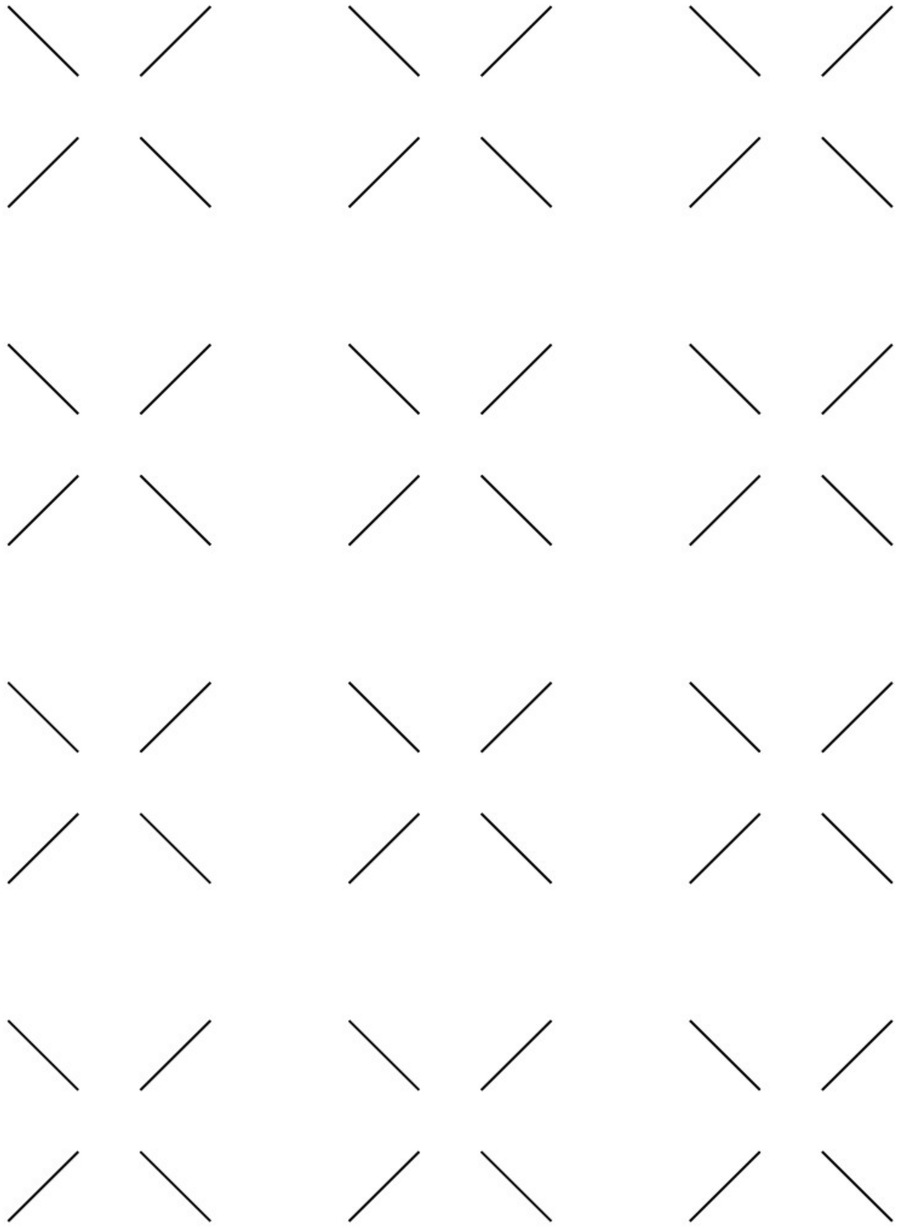
1) Download and install the MySQL JDBC driver: https://dev.mysql.com/downloads/connector/j/
2) Add the driver to the Class Path:
   - Add an external JAR file
   - Apply and close



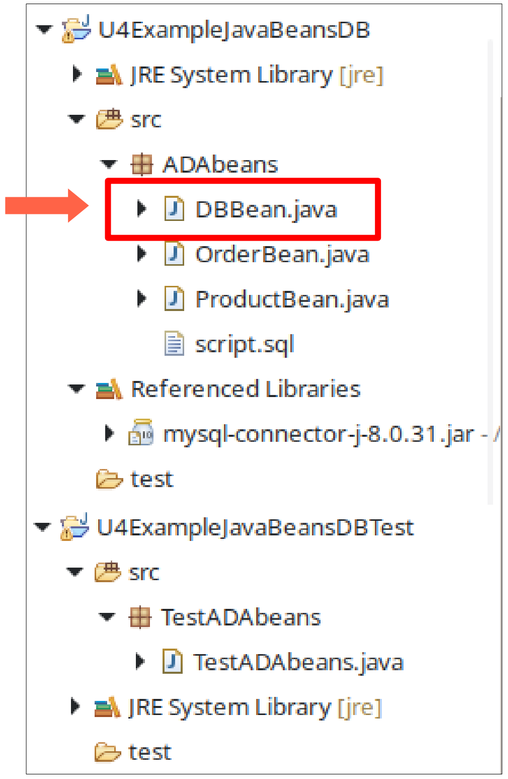
## 2.2 Step 2. Creating new DB class



# STEP 2: DBBean class (1 of 2)

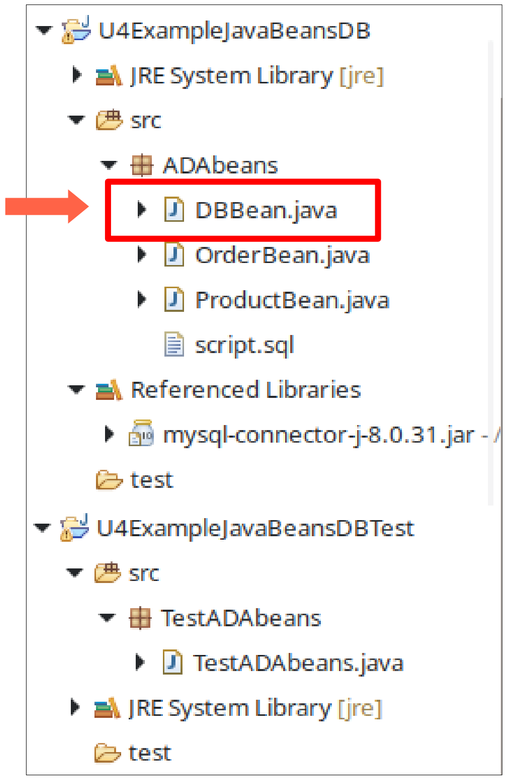
- Create a **new JavaBean** with this code.
- Pay attention on the type of the class we are creating.
- It's just an auxiliary bean, so there's no need to implement from any interface such us Serializable or PropertyChangeListener.

```
package ADAbeans;

import java.sql.*;
public class DBBean {

   static final String DBNAME = "DBProducts";
   static final String DBUSER = "mavenuser";
   static final String DBPASSWORD = "ada0486";
   static final String URL = "jdbc:mysql://localhost:3306/" + DBNAME
         + "?
useSSL=false&useTimezone=true&serverTimezone=UTC&allowPublicKeyRetrieval=true";
    private Connection connDB = null;
    public DBBean( ) {
        try {
            // This will load the MySQL driver, each DB has its own driver
            Class.forName("com.mysql.cj.jdbc.Driver");
            // Setup the connection with the DB
            connDB = DriverManager.getConnection(URL,DBUSER,DBPASSWORD);

            System.out.println("Connected to the database.");
        } catch (ClassNotFoundException | SQLException sqle) {
            System.out.println("Got an exception (connecting)!");
            System.out.println(sqle.getMessage( ));
        }
    }
```

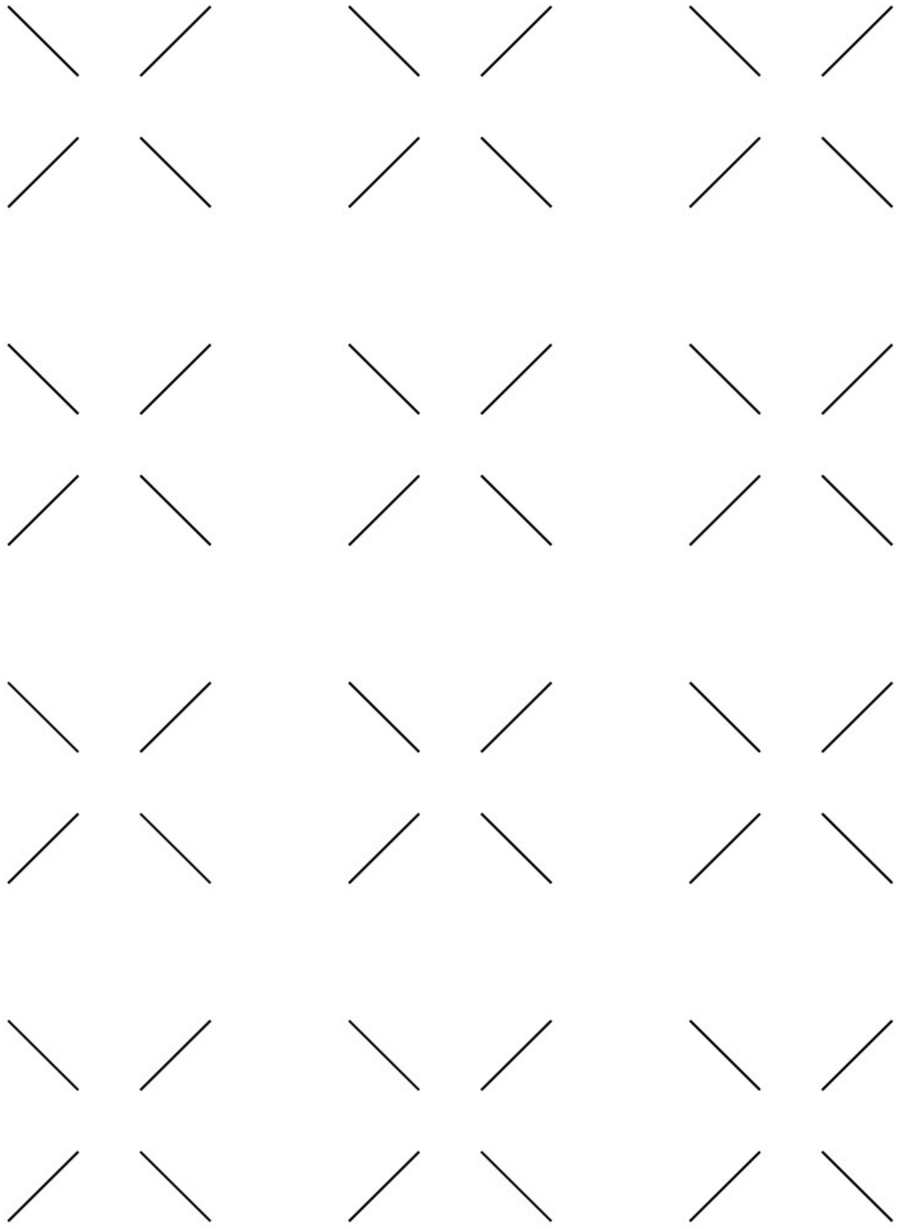


# STEP 2: DBBean class (2 of 2)

```
public static void insertOrder(ProductBean objProductBean, int iAmount) {
        System.out.println("Inserting...");
        try {
        //create a new connection to the DB
            DBBean DBProducts = new DBBean();
            // the MySQL insert statement
            String stSQLquery = " INSERT INTO orders (idp, amount)" + " VALUES (?, ?)";
            // create the MySQL insert prepared statement
            PreparedStatement preparedStmt = DBProducts.connDB.prepareStatement(stSQLquery);
            preparedStmt.setInt(1, objProductBean.getiProductid());
            preparedStmt.setInt(2, iAmount);
            // execute the prepared statement
            preparedStmt.execute();
            //close DB connection
            DB.closeDBConnection();
            //show information about the new created order
            System.out.println("Order inserted (idp=" + objProductBean.getiProductid()
                    + " amount=" + iAmount + ")");
        } catch (SQLException sqle) {
            System.out.println("Got an exception (inserting)!");
            System.out.println(sqle.getMessage());
        }
    }

    private void closeDBConnection() {
        try {
            if (connDB != null) {
              connDB.close();
            }
        } catch (Exception exe) {
        System.out.println("Exception while closing" + exe.getMessage());
        }
    }
```



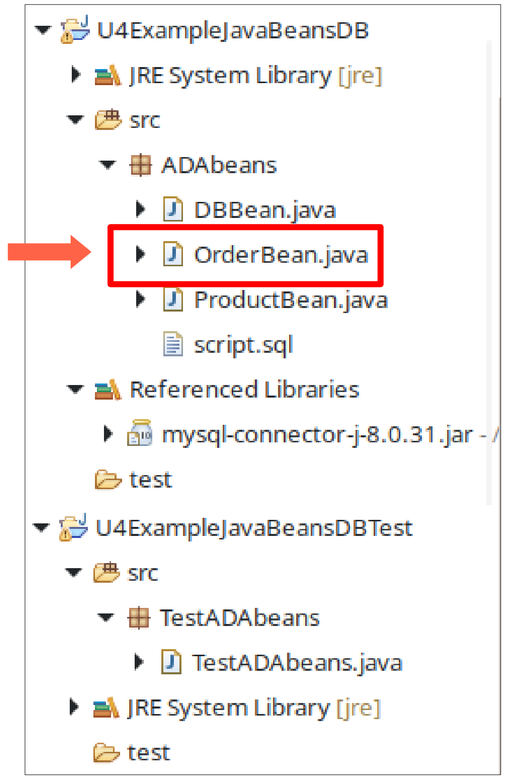
## 2.3 Step 3. Modifying listeners



# STEP 3: Changes to OrderBean class

- We only need to change the propertyChange method, adding more messages and a call to the method insertOrder as you can see below.
- In sum, we're inserting an order when the stock changes below the minimum stock or when the minimum stock raises above the current stock.
- We're making an STATIC call to InsertProduct method but there are many other ways to get the same goal.

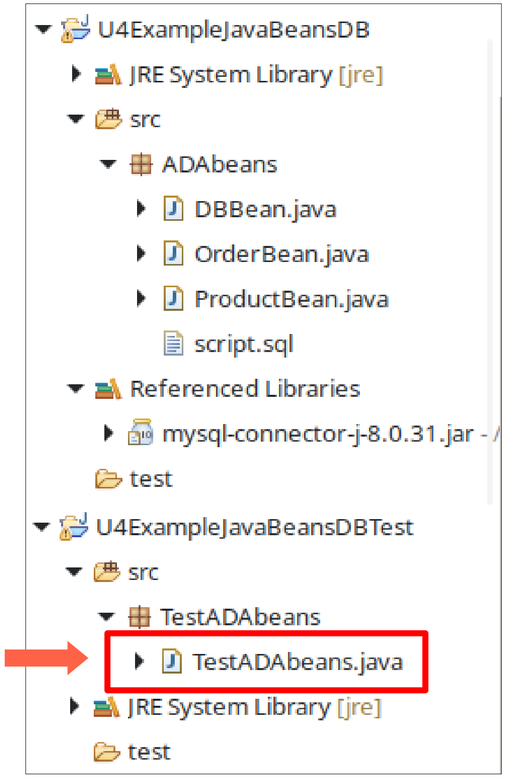
```
public void propertyChange(PropertyChangeEvent pceEvent) {
   int iAmountOrder = objProductBean.getiMinStock() - objProductBean.getiCurrentStock();

    if (pceEvent.getPropertyName().equals("currentStockBelowMinStock"))
    {
        System.out.printf("[OrderBean says... ]%n");
        System.out.printf("Current stock is now less than minimum stock!%n");
        System.out.printf("=> Old current Stock: %d%n", pceEvent.getOldValue());
        System.out.printf("=> New current Stock: %d%n", pceEvent.getNewValue());
        System.out.printf("It will place an order for this product: %s%n",
         objProductBean.getstDescription());
        //create a new order in the DB. Amount = minStock - Current stock
        DBBean.insertOrder(objProductBean, iAmountOrder);
    }
    if (pceEvent.getPropertyName().equals("minStockRaisedOverCurrentStock"))
    {
        System.out.printf("[OrderBean says... ]%n");
        System.out.printf("Minimum stock is now greater than current stock!%n");
        System.out.printf("Old minstock Stock: %d%n", pceEvent.getOldValue());
        System.out.printf("New minstock Stock: %d%n", pceEvent.getNewValue());
        System.out.printf("It will place an order for this product: %s%n",
         objProductBean.getstDescription());
      //create a new order in the DB. Amount = minStock - Current stock
        DBBean.insertOrder(objProductBean, iAmountOrder);
    }
}
```



## 2.4 Step 4. Modifying main programme

# STEP 4: Changes to TestADAbeans class

```
public class TestADAbeans {

   public static void main(String[] stArgs) {
        /*
         * Creating the objects
         */
        //Object source
        //ProductBean(int iProductid, String stDescription, float fPrice, int iCurrentstock, int iMinstock)
        //Setting currentStock to 101 units and minimumStock to 100 units
        ProductBean objProductBean = new ProductBean(1, "Robot hoover", 399, 101, 100);
        //Object listener
        OrderBean objOrderBean = new OrderBean();
        /*
         * Assign the object source to the listener
         * Start the listener object
         */
        objOrderBean.setobjProductBean(objProductBean);
        objProductBean.addPropertyChangeListener(objOrderBean);
        /*
         * Firing events
         */
        //Setting currentStock to 40 (below the minimum advisable)
        System.out.println("****** product.setCurrentStock(40):");
        objProductBean.setiCurrentStock(40);
        //Setting minimumStock to 50 (over the current stock)
        System.out.println("****** product.setMinStock(50):");
        objProductBean.setiMinStock(50);
    }
}
```



# 3. ACTIVITIES FOR NEXT WEEK

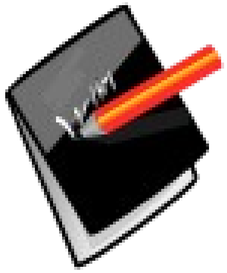
# Proposed activities

Check the suggested exercises you will find at the “Aula Virtual”. **These activities are optional and non-assessable but** understanding these non-assessable activities is essential to solve the assessable task ahead.

Shortly you will find the proposed solutions.

# 4. BIBLIOGRAPHY

# Resources




- Oracle Java Documentation. JavaBeans Component API. https://docs.oracle.com/javase/8/docs/technotes/guides/beans/index.html
- JavaBeans Tutorial - MIT - Massachusetts Institute of Technology. http://web.mit.edu/javadev/doc/tutorial/beans/index.html
- Tutorials freak. JavaBeans Class in Java: Properties, Examples, Benefits, Life Cycle. https://www.tutorialsfreak.com/java-tutorial/javabeans
- I/O Flood. Java Bean Explained: Object Encapsulation Guide. https://ioflood.com/blog/java-bean/
- Josep Cañellas Bornas, Isidre Guixà Miranda. Accés a dades. Desenvolupament d'aplicacions multiplataforma. Creative Commons. Departament d'Ensenyament, Institut Obert de Catalunya. Dipòsit legal: B. 29430-2013. https://ioc.xtec.cat/educacio/recursos