

FUNDAMENTAL PROGRAMMING TECHNIQUES

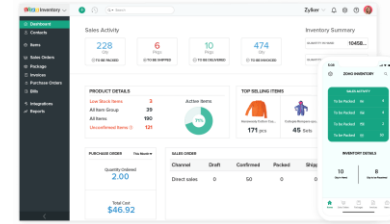
ASSIGNMENT 3 – SUPPORT PRESENTATION (PART I)

Problem and solution

PROBLEM: “Managing the products, the clients and the orders for a warehouse using hand-written registries is difficult and time consuming”



SOLUTION: order management application



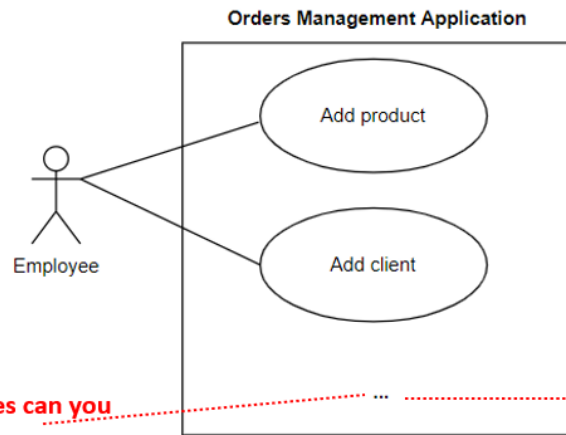
How to design and implement the solution?

1. Clearly state the main objective and the sub-objectives required to reach it.
2. Analyze the problem and define the functional and non-functional requirements.
3. Design the solution
4. Implement the solution
5. Test the solution

Objectives

- Main objective
 - Design and implement an application for managing the client orders for a warehouse
- Sub-objectives
 - Analyze the problem and identify requirements
 - Design the orders management application
 - Implement the orders management application
 - Test the orders management application

Analysis



What other use cases can you identify?

Can you identify use cases connected with "extend", "include" and "generalize" relationships?

Define requirements

Functional requirements:

- The application should allow an employee to add a new client
- The application should allow an employee to add a new product
- ... what other functional requirements can you define? ...

Use Case: add product

Primary Actor: employee

Main Success Scenario:

1. The employee selects the option to add a new product
2. The application will display a form in which the product details should be inserted
3. The employee inserts the name of the product, its price and current stock
4. The employee clicks on the "Add" button
5. The application stores the product data in the database and displays an acknowledge message

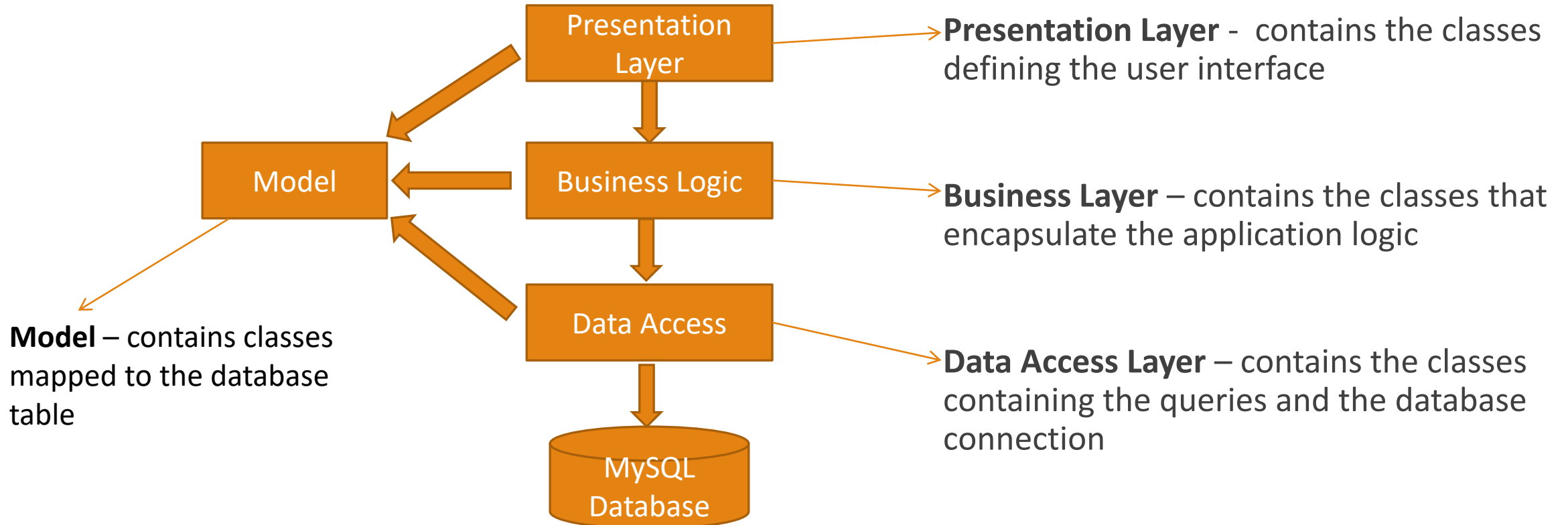
Alternative Sequence: Invalid values for the product's data

- The user inserts a negative value for the stock of the product
- The application displays an error message and requests the user to insert a valid stock
- The scenario returns to step 3

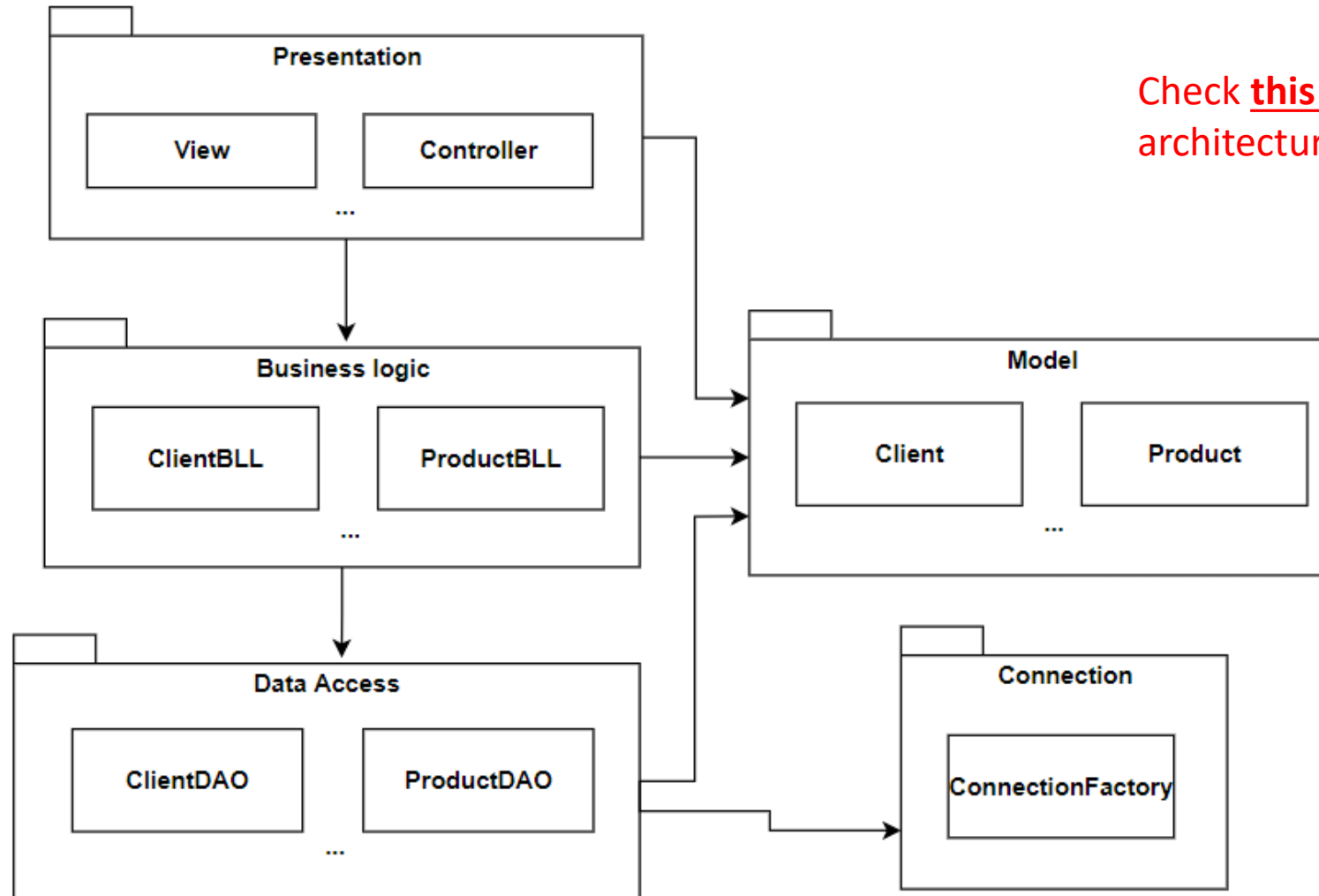
Non-Functional requirements:

- The application should be intuitive and easy to use by the user
- ... what other non-functional requirements can you define? ...

Design – Conceptual Architecture



Design – Detailed Architecture



Check [this example](#) of a layered architecture implementation!!!

JDBC Basics – Processing SQL Statements

- Steps
 - Establish a connection with the data source
 - Create a statement
 - Execute the query
 - Process the ResultSet object
 - Close the connection

JDBC Basics – Establishing a Connection

- This class contains the name of the driver (initialized through reflection), the database location (DBURL), and the user and the password for accessing the MySQL Server
- The connection to the DB will be placed in a *Singleton** object
- The class contains methods for creating a connection, getting an active connection and closing a connection, a Statement or a ResultSet

```
public class ConnectionFactory {  
    private static final Logger LOGGER = Logger.getLogger(ConnectionFactory.class.getName());  
    private static final String DRIVER = "com.mysql.cj.jdbc.Driver";  
    private static final String DBURL = "jdbc:mysql://localhost:3306/schooldb";  
    private static final String USER = "root";  
    private static final String PASS = "root";  
  
    private static ConnectionFactory singleInstance = new ConnectionFactory();  
  
    private ConnectionFactory() {  
        try {  
            Class.forName(DRIVER);  
        } catch (ClassNotFoundException e) {  
            e.printStackTrace();  
        }  
    }  
  
    private Connection createConnection() {  
    }  
  
    public static Connection getConnection() {  
    }  
  
    public static void close(Connection connection) {  
    }  
  
    public static void close(Statement statement) {  
    }  
  
    public static void close(ResultSet resultSet) {  
    }  
}
```

*Singleton Design Pattern: https://en.wikipedia.org/wiki/Singleton_pattern

JDBC Basics – Table Mapping

- In order to extract elements from the DB table, a special class (named entity) must be created.
- This class **MUST** have the fields exactly the same type as the columns from the corresponding table.
- The class must have also constructors, getters and setters.

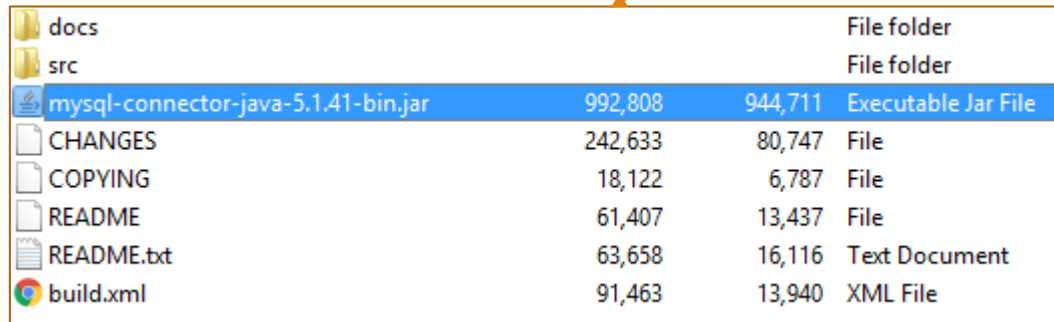
```
public class Student {  
    private int id;  
    private String name;  
    private String address;  
    private String email;  
    private int age;  
}
```



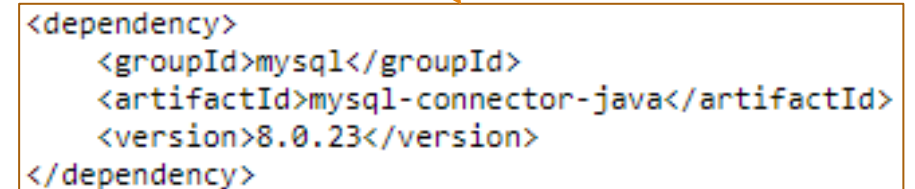
	id	name	address	email	age
	1	Ion	Baritiu	ion@cs.utcluj.ro	22
	2	Maria	Observator	maria@cs.utcluj.ro	22
▶*		NULL	NULL	NULL	NULL

JDBC Basics - Dependencies

- In order for the Java application to interact with the DB, a special **.jar** library must be added to the application
- It can be added either as an **external jar file dependency** or as a **maven dependency**, in case of a Maven project



docs			File folder
src			File folder
mysql-connector-java-5.1.41-bin.jar	992,808	944,711	Executable Jar File
CHANGES	242,633	80,747	File
COPYING	18,122	6,787	File
README	61,407	13,437	File
README.txt	63,658	16,116	Text Document
build.xml	91,463	13,940	XML File



```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.23</version>
</dependency>
```

- The Java application uses this external library to communicate with the MySQL server
- It sends queries to the server using **Statements** and it receives the results of the queries as **ResultSet**

JDBC Basics – Creating and Executing Statement

1. Define a string with the query

```
private final static String findStatementString = "SELECT * FROM student where id = ?";
```

```
public static Student findById(int studentId) {  
    Student toReturn = null;
```

```
    Connection dbConnection = ConnectionFactory.getConnection();
```

```
    PreparedStatement findStatement = null;
```

```
    ResultSet rs = null;
```

```
    try {
```

```
        findStatement = dbConnection.prepareStatement(findStatementString);
```

```
        findStatement.setLong(1, studentId);
```

```
        rs = findStatement.executeQuery();
```

2. Create a connection to the DB

3. Initialize the query

4. Add the parameters to the query (the ? Will be replaced with data from the application)

5. Execute the query

JDBC Basics - Process the ResultSet object

The results of the query execution are stored in a result set:

- Each element of the result set corresponds to a row from the table
- The result set can be iterated
- The properties/values from the columns can be extracted if the column name is known

```
rs = findStatement.executeQuery();  
rs.next();
```

```
String name = rs.getString("name");  
String address = rs.getString("address");  
String email = rs.getString("email");  
int age = rs.getInt("age");
```

JDBC Basics – Closing the connection

After each operation the connection must be closed:

- The result set → `ConnectionFactory.close(rs);`
- The statement → `ConnectionFactory.close(findStatement);`
- The connection → `ConnectionFactory.close(dbConnection);`