

JAVA SE 8 PROGRAMMING LANGUAGE

Training Assignments


Document Code	25e-BM/HR/HDCV/FSOFT
Version	1.1
Effective Date	20/11/2012

RECORD OF CHANGES

No	Effective Date	Change Description	Reason	Reviewer	Approver
1	08/Oct/2018	Create a new Assignment	Create new	DieuNT1	VinhNV
2	01/Mar/2019	Apply fsoft template	Update	DieuNT1	VinhNV

Contents

Long Assignment 3 – Option 1: Building Database Application with JDBC	4
Objectives:.....	4
Product Architecture:	4
Assignment Specifications:	5
Technical Requirements:	5
Functional Requirements	6
User Interface Requirements	6

	CODE:	JPL.L.A301
	TYPE:	LONG
	LOC:	300
	DURATION:	180 MINUTES

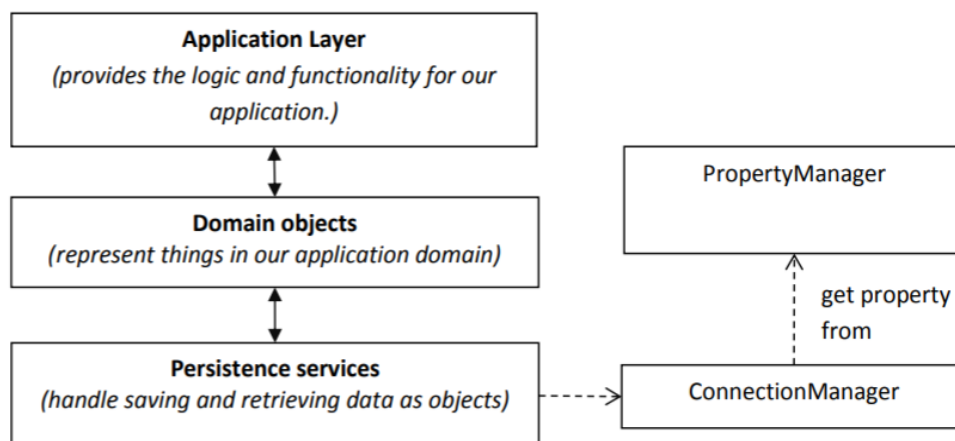
Long Assignment 3 – Option 1: Building Database Application with JDBC

Objectives:

- » Understand how to connect to database server and all components of JDBC APIs.
- » How to work with Statement, ResultSet, PreparedStatement, CallableStatement, Stored procedures using INPUT/OUTPUT parameters
- » How to call and execute stored procedures with java program.

Product Architecture:

This application will be developed using following architecture:



- » The domain layer contains objects and logic related to our application.
- » The persistence layer contains data access objects (DAO) that provide services for accessing persistent data. DAO provide 4 basic services referred to as CRUD:
 - **Create**: save new object data to the database.
 - **Retrieve**: find object data in the database and recreate objects. There may be several methods for this service to enable different forms of object lookup.
 - **Update**: update data for an object already saved to the database.
 - **Delete**: delete object data from the database.

Specifications:

Write a program that simulates the functions of the sales system.

Create a database named **SMS** for the sales system that has the following data tables:

Customer (*customer_id*, customer_name)

Employee (*employee_id*, employee_name, salary, *supervisor_id*)

Product (*product_id*, product_name, list_price)

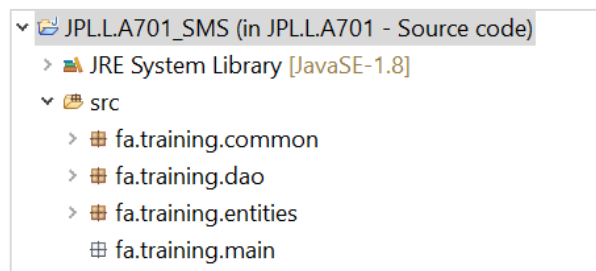
Orders (*order_id*, order_date, *customer_id*, *employee_id*, total)

LineItem (*order_id*, *product_id*, quantity, price)

Notice that, all of the tables belong the same "dbo" schema.

Technical Requirements:

Create a project named **JPL.L.A301_SMS** will have the following packages:



Inside package **fa.training.entities** contains the following classes:

The **LineItem** class:

- Four private instance variables: *orderId* (int), *productId* (int), *quantity* (int), *price* (double)
- Default constructor and the constructor has 4 parameters to initialize value of attributes.
- Getter and setter methods.

The **Customer** class:

- Two private instance variables: *customerId* (int), *customerName* (String)
- Default constructor and the constructor has 2 parameters to initialize value of attributes.
- Getter and setter methods.

The **Employee** class:

- Four private instance variables: *employeeId* (int), *employeeName* (String), *salary* (double), *spvId* (int)
- Default constructor and the constructor has 4 parameters to initialize value of attributes.
- Getter and setter methods.

The **Product** class:

- Three private instance variables: *productId* (int), *productName* (String), *listPrice* (double)
- Default constructor and the constructor has 3 parameters to initialize value of attributes.
- Getter and setter methods.

The **Order** class:

- Five private instance variables: *orderId* (int), *orderDate* (Date), *customerId* (int), *employeeId* (int), *total* (double)

- Default constructor and the constructor has 5 parameters to initialize value of attributes.
- Getter and setter methods.

Each entities will have its own interfaces and classe which are allocated inside *fa.training.dao*, follow the following pattern: for interfaces (i.e **OrderDAO**, **ProductDAO**), for impl class (**OrderDAOImpl**, **ProductDAOImpl**).

Functional Requirements

- 1) List all customers consist of *customer id*, *customer name* in the database, returns a list with all customers in the order table (`List<Customer> getAllCustomer()` method).
- 2) List all orders consist of order id, order date, customer id, employee id, total for a customer, returns a list with all the orders for a given customer id (`List<Order> getAllOrdersByCustomerId(int customerId)` method).
- 3) List all lineitems for an order, returns a list with all line items for a given order id (`List<LineItem> getAllItemsByOrderId(int orderId)` method).
- 4) Compute order total, returns a list with a row containing the computed order total from the line items (named as **total_price**) for a given order id. You must use an UDF (`Double computeOrderTotal(int orderId)` method).
- 5) Add a customer into the database, you must use a Stored Procedure (`boolean addCustomer(Customer customer)` method).
- 6) Delete a customer from the database, make sure to also delete Orders and OrderedProducts for the deleted customer. You must use a Stored Procedure (`boolean deleteCustomer(int customerId)` method).
- 7) Update a customer in the database, you must use a Stored Procedure (`boolean updateCustomer(Customer customer)` method).
- 8) Create an order into the database (`boolean addOrder(Order order)` method).
- 9) Create a lineitem into the database (`boolean addLineItem(LineItem item)` method).
- 10) Update an order total into the database (`boolean updateOrderTotal(int orderId)` method).

User Interface Requirements

Create a **SaleManagement** class that contains a `main()` method to test the above functional methods.

-- THE END --