

## **Carrito de Compras JSF y MySQL**

Bien este proyecto es un rework del ejemplo del carrito de compras creado por: Henry Joe Wong Urquiza en la antigua página: "programandoconcafe"

El ejemplo original está muy bien explicado, fue realizado en jsp, manejando todo con HttpServletRequest y HttpServletResponse, separando bien las capas y con un código limpio, organizado y bien explicado, con buena documentación para novatos. Sin embargo tiene algunas cosas por mejorar, puesto que no hace validaciones de ninguna clase, permite ingresar precio en cero, y no valida que haya agregado una cantidad de productos mayor de cero al carrito, entre otras. Pero independiente de ello el ejemplo original está bien construido, su autor es muy bueno en la parte técnica y sabe bastante.

Que se hizo diferente ahora, que se trabajó con Java Server Faces, y se le aplicaron validaciones, mejorando un poco el diseño en la base de datos.

Para empezar se modificó en la BD la generación de los ID, de las estructuras producto y venta, por autoincrementales evitando estar haciendo Select Max(ID) desde el front para obtener el ID. Se modificó la fecha de venta cambiando su tipo de dato de date a timestamp, y se le colocó un default, para que tome automáticamente la fecha del día, y sea la BD quien se encargue de esas tareas. Se adiciona una columna en producto para almacenar su imagen, pues lo que no se muestra no se vende.

Desde luego esta BD se queda corta, para reflejar todo lo que demanda un sistema de Ecommerce, pues a todas luces faltan muchas más estructuras como: Clientes, Categorías, Proveedores, Ofertas, Calificaciones, Pagos, Roles y Seguridad; porque cualquiera puede modificar precios y productos, labor que solo debería realizar un usuario tipo administrativo, pero bien cada quien que lo optimice y se divierta, que de eso se trata, al fin de cuentas es un ejemplo muy pequeño.

Se redujo la cantidad de procedimientos almacenados, unificando los de consulta en uno solo, y uniendo los de inserción y actualización del producto, en uno solo, al igual que con los procedimientos que ingresan la venta y su respectivo detalle, quedando de esta forma solo 3 procedimientos almacenados. Creo que queda más optimizado, desde la perspectiva de que no estamos haciendo múltiples llamados, un mismo SP sin necesidad, pues inicialmente se invocaba el SP que ingresa el detalle de la venta, como tantos productos se hubiesen adquirido, no es que eso esté mal, de hecho así tiene que ser, pues si yo en un pedido o venta, compro 10 productos por ejemplo, necesariamente tendrán que efectuarse 10 inserciones a la estructura del detalle, para mejorar eso lo que se hizo, es que al SP que ingresa esos detalles se le pasa el contenido completo del carro, y en el mismo SP se ingresa tanto la venta, como sus detalles asociados, sin necesidad de hacer 10 viajes innecesarios a la BD, efectuamos todo en una sola ida.

Se modificó la estructura de los métodos en Java, que acceden a la BD, transformándolos para que no sean métodos "static", y en algunos casos se cambió el retorno de dichos métodos de boolean a int.

Se creó una capa adicional llamada DAL, donde alojamos todos los métodos que apuntan a la BD (Conexion, VentaBD y ProductoBD)

Se excluyeron de los modelos los constructores con parámetros, pues no se necesitan.

Se guardan los datos del carrito en un arraylist en vez de guardarlas en la sesión como lo hacen en el ejemplo original, que igual también se podrían seguir guardando en sesión, ya que en jsf incluso es más fácil trabajar las sesiones.

Se creó una plantilla o template para evitar crear los links en cada página.

Se corrigió un pequeño bug en VentaBD. donde estaban asignando el descuento al parcial, lo que impedía visualizar bien las ventas.

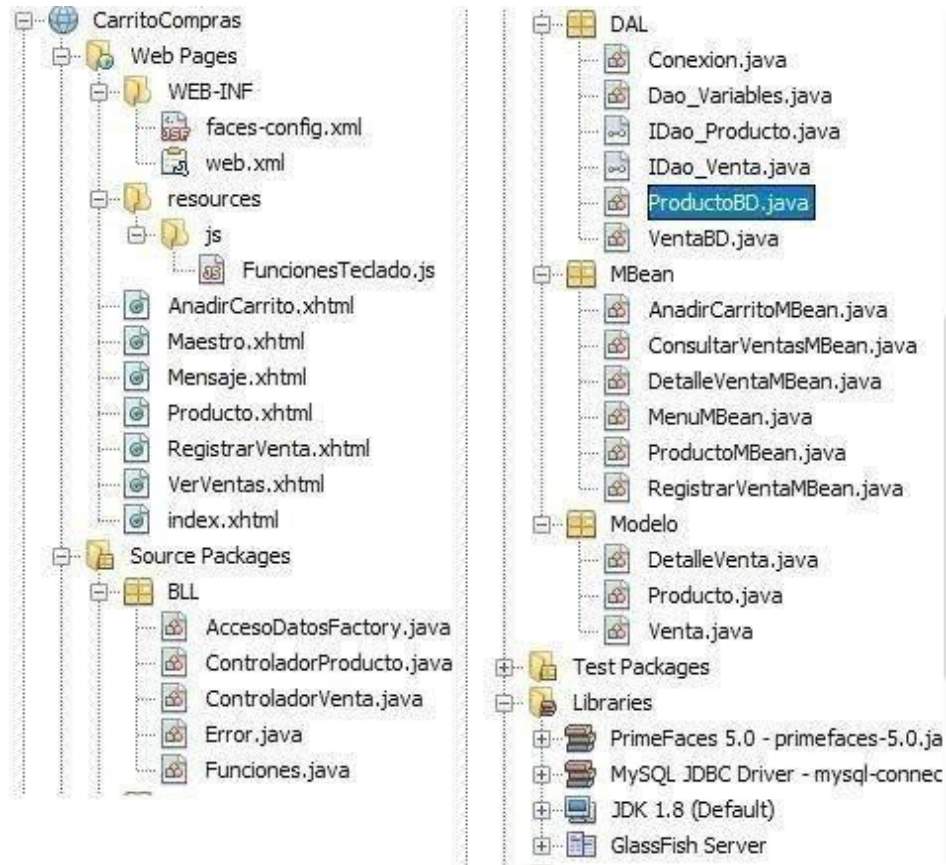
#### Entorno:

- JSF 2.2
- PrimeFaces 5.0
- IDE NetBeans 8.2
- Glassfish Server 4.1.1
- MySQL 6.0
- JDK 1.8

#### Diagrama Base de Datos



## Estructura del Proyecto en NetBeans



## ARQUITECTURA EMPLEADA

El objetivo es diseñar una aplicación multicapa, separando adecuadamente las capas de presentación, negocio y datos, construyendo e implementando clases orientadas a objetos, reusables y de fácil mantenimiento.

Estará basado en el patrón MVC (Modelo Vista Controlador), porque nos facilita el mantenimiento, ya que podemos cambiar una o más capas sin afectar las otras. El contexto es diseñar una aplicación en capas, donde cada capa expone servicios que otras aplicaciones o capas pueden consumir, y donde cada capa puede consumir servicios de otras, lo que se traduce en una simplicidad conceptual, con alta cohesión y un bajo acoplamiento, además de facilitar su reutilización.

Donde predomina una “organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.” Garlan & Shaw.

Se acabaron aquellos días de programación tipo “Espagueti”, donde teníamos variables definidas por doquier, había dificultad para escribir y recordar el código, no era posible reutilizar, había que copiarlo y pegarlo todo, se abría una conexión por cada operación, y nuestras aplicaciones quedaban expuestas al SQL injection, ya que todo el código estaba mezclado con la capa de presentación, sin orden alguno; desde luego realizar mantenimiento a una aplicación de ese tipo, era arduo, dispendioso y a menudo peligroso.

Acá manejaremos los siguientes aspectos de arquitectura:

- Aplicación Web Cliente – Servidor.
- Estilo arquitectónico Multicapa: presentación, lógica de negocio y datos.
- Uso de Patrones MVC y Factory Method.
- Orientado a Objetos y Componentes.
- Creación de objetos de negocio personalizados.
- Uso de buenas prácticas, desde luego que hay mejores.

Este documento es una visión muy particular de un entorno específico, sin querer en ningún momento estereotipar, o afirmar que esta sea la mejor y la única forma de hacer las cosas, en el desarrollo de software siempre habrá más y mejores formas de llevarlo a cabo.

"Beethoven era un buen compositor porque utilizaba ideas nuevas en combinación con ideas antiguas. Nadie, ni siquiera Beethoven, podría inventar la música desde cero. Es igual con la informática."

### **Explicación de las Capas con sus respectivas clases**

No copiaré el contenido del código de cada uno de los módulos, pues esto se encuentra en el repositorio de Github, solo me limitaré a comentar que hace cada una de las capas.

#### **• Capa BLL: Business Logic Layer - Lógica de Negocio**

La capa de lógica de negocio, es donde estarán alojados nuestros controladores, es aquella capa donde manejamos las reglas de negocio particulares del escenario que modelamos. En realidad es una capa que sirve de enlace entre las vistas y los modelos (BO), respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo. En esta capa tendremos alojadas las siguientes clases:

**AccesoDatosFactory:** Aquí implementamos este patrón que nos facilitará el acceso a los DAO, sacando provecho de su constructor para pasar como argumento, las interfaces que requieren ya sea IDao\_Producto o IDao\_Venta para acceder a Producto y Venta respectivamente, que se encuentran alojados en la capa DAL.

**ControladorProducto:** Este es el controlador de todo lo que tiene que ver con el objeto Producto, agregar, actualizar y consultar productos tendrá que pasar por acá.

**ControladorVenta:** Este es el controlador de todo lo que tiene que ver con Ventas, como registrar y consultar ventas tendrá que pasar por acá.

**Error:** Es una clase que maneja todo lo que tiene que ver con los mensajes informativos del sistema.

**Funciones:** Es una clase transversal donde están alojados los mensajes que recibe el usuario.

- **Capa DAL: Data Access Layer - Acceso Datos**

Es la capa de acceso a la base de datos, donde se trabaja con los datos, por tanto contendrá mecanismos para acceder a la información y también para actualizar su estado, para este ejemplo solo tenemos dos DAO que son ProductoBD y VentaBD que acceden a Producto y Venta respectivamente. En esta capa tendremos alojadas las siguientes clases:

**Conexion:** Es donde definimos nuestra conexión a la base de datos con el respectivo proveedor, en este caso MySql.

**Dao\_Variables:** Este es un DAO Genérico que contiene métodos transversales que usan los demás DAO, como el método de consultar y de liberar recursos que son llamados desde ProductoBD y Venta BD, entonces para evitar estar repitiendo lo mismo, simplemente los otros DAO extienden de esta clase, y así reutilizamos código aprovechando la herencia.

**IDao\_Producto:** Interfaz que expone los métodos que implementa el DAO para productos.

**IDao\_Venta:** Interfaz que expone los métodos que implementa el DAO para Ventas y Detalle de Venta.

**ProductoBD:** Es el Dao que administra todo el acceso a datos para el control de productos.

**VentaBD:** Es el Dao que administra todo el acceso a datos para el control de ventas.

- **Managed Beans - Gestionadores de Clase**

Los managed bean son clases java que se registran en el framework de JSF, lo que permite que puedan ser consumidos desde las páginas, cuyo objetivo es servir de soporte a cada una de las páginas o formularios, en este ejemplo detrás de cada objeto de negocio (BO) hay un ManagedBean que será accedido por la página implicada. En esta capa tendremos alojadas las siguientes clases:

**AnadirCarritoMBean:** Gestiona y controla todo lo referente a adiciones de productos al carrito de compras

**ConsultarVentasMBean:** Gestiona y controla todo lo referente a consulta de ventas

**DetalleVentaMBean:** Gestiona y controla todo lo referente al detalle de las ventas

**MenuMBean:** Gestiona y controla todo lo referente al menú principal de la aplicación

**ProductoMBean:** Gestiona y controla todo lo referente a productos.

**RegistrarVentaMBean:** Gestiona y controla todo lo referente al registro de una venta

- **Capa BO: Business Objects - Objetos de Negocio**

Son los objetos de negocio inherentes al escenario que se modela, es una réplica de las estructuras de la base de datos, con sus respectivos get y set. Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, envía a la vista aquella parte de la información que en cada momento se le solicita para que sea mostrada al usuario. En esta capa tendremos alojadas las siguientes clases:

**DetalleVenta:** Es la clase u objeto de negocio que modela el detalle de la venta, que son todos esos ítems que hacen parte de la venta.

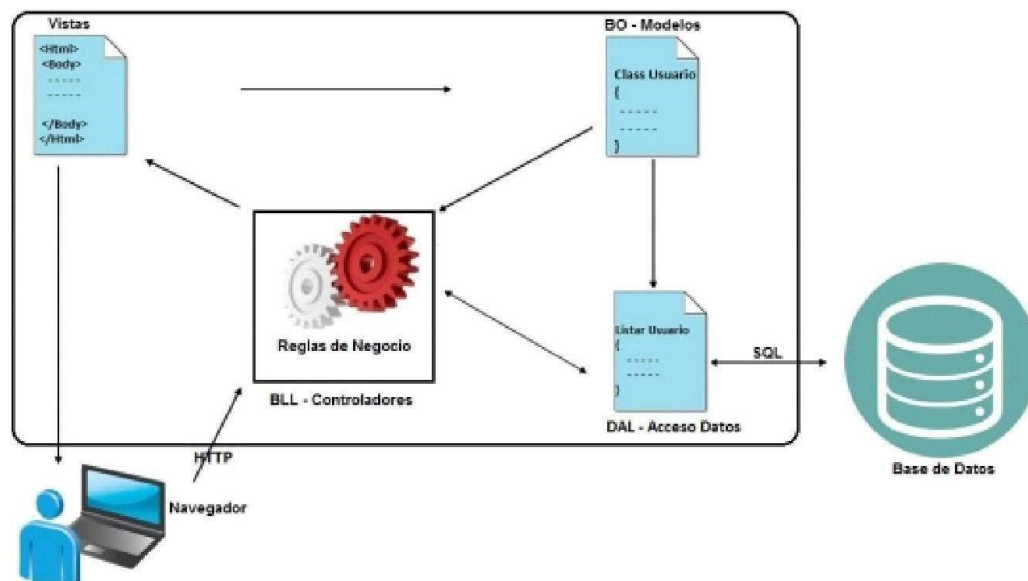
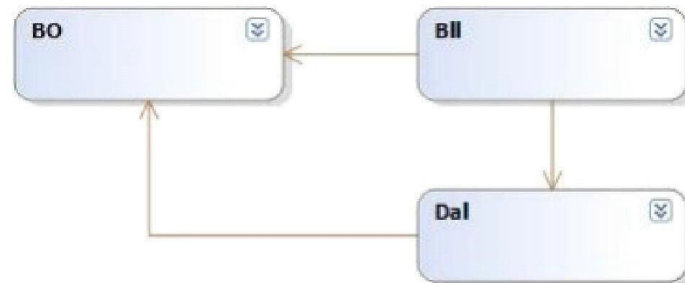
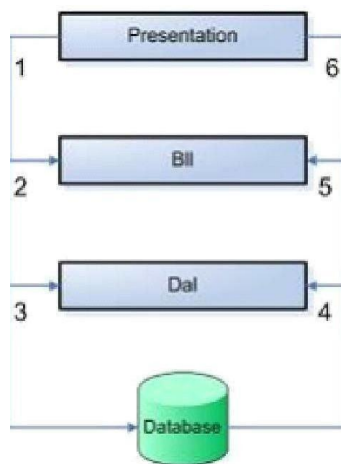
**Producto:** Es la clase u objeto de negocio que modela el producto y sus atributos.

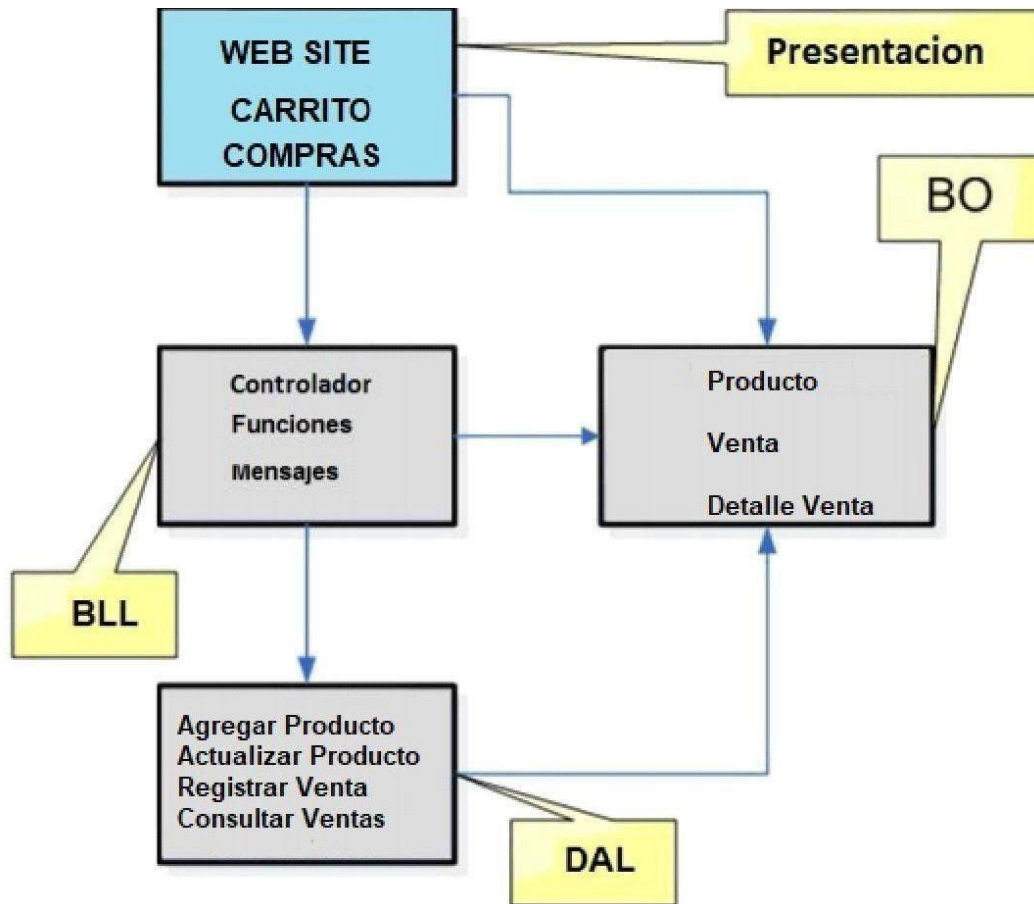
**Venta:** Es la clase u objeto de negocio que modela la venta y sus atributos.

- **Capa Web Pages: Presentación - Formularios y Vistas**

Es la capa donde agrupamos las vistas o formularios, como su nombre nos hace entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML, que es lo que usuario visualiza.

Este sistema deberá verse reflejado así:











Funcionamiento Global - Carrito de Compras



## Listado de Productos (Index)

[Inicio](#) [Registrar Producto](#) [Registrar Venta](#) [Consultar Ventas](#)

<div>Diadema</div> <div></div> <div>Precio: \$ 25000.0</div> <div>Modificar</div> <div>Añadir</div>	<div>Disco Solido</div> <div></div> <div>Precio: \$ 480000.0</div> <div>Modificar</div> <div>Añadir</div>	<div>Impresora</div> <div></div> <div>Precio: \$ 419000.0</div> <div>Modificar</div> <div>Añadir</div>
<div>Monitor</div> <div></div> <div>Precio: \$ 500000.0</div> <div>Modificar</div> <div>Añadir</div>	<div>Portatil</div> <div></div> <div>Precio: \$ 1500000.0</div> <div>Modificar</div> <div>Añadir</div>	<div>Teclado y Mouse</div> <div></div> <div>Precio: \$ 79500.0</div> <div>Modificar</div> <div>Añadir</div>

## Actualizar Productos

[Inicio](#) [Registrar Producto](#) [Registrar Venta](#) [Consultar Ventas](#)

Nombre \*

Precio \*

## Añadir Carrito

[Inicio](#) [Registrar Producto](#) [Registrar Venta](#) [Consultar Ventas](#)

Codigo: 3

Nombre: Mouse Genérico

Precio: 10000.0

Cantidad:

## Registrar Venta

[Inicio](#) [Registrar Producto](#) [Registrar Venta](#) [Consultar Ventas](#)

Cliente \*

Carrito de Compras

Codigo	Nombre	Cantidad	Precio	Descuento	SubTotal
3	Mouse Genérico	2.0	10000.0	1000.0	19000.0

## Consultar Ventas

[Inicio](#) [Registrar Producto](#) [Registrar Venta](#) [Consultar Ventas](#)

Lista de Ventas

Cod. Venta	Producto	Precio	Cantidad	Descuento	SubTotal
51	Mouse Genérico	10000.0	2.0	1000.0	19000.0

Bien esto es todo, si tienes dudas e inconvenientes no dejes de escribirme