DOCUMENTACIÓN DE PROYECTO

Universidad Nacional de Luján
REPUBLICA ARGENTINA

DEPARTAMENTO: CIENCIAS BASICAS

CARRERA: LICENCIATURA EN SISTEMAS DE INFORMACION

ASIGNATURA: PROGRAMACION ORIENTADA A OBJETOS (Progamación III)

Título	Carrito de compras

Objetivos

El objetivo del proyecto consiste en desarrollar una aplicación Java tipo carrito compras que se conecta a una base de datos relacional para efectivizar ventas a clientes.

Dicha aplicación contará con una interfaz gráfica de usuario y distintas funcionalidades como la de insertar y actualizar productos en venta y clientes al sistema, aplicar descuentos a pedidos y añadir y eliminar productos en el carrito.

La aplicación también será capaz de emitir reportes sobre las ventas llevadas a cabo en la empresa a lo largo de su actividad.

En cuanto al código fuente del software presentado cumplirá con el patrón de arquitectura modelo-vista-controlador para facilitar futuras implementaciones de cada capa. También contará con testeos de las clases del modelo utilizados durante el desarrollo.

Profesor	

Javier Blanqué (jblanque@gmail.com)

Trenque Lauquen 433, CP 1712, Castelar, Buenos Aires

011-4623-7280 / 011-15-4438-3714

Ayudante	

INTEGRANTES DEL GRUPO DE TRABAJO DEL PROYECTO		
1	APELLIDO: Cardona. NOMBRE: Juan. LEGAJO UNLu: 122124	PERFIL: TELEFONOS: 02323-15609065 EMAIL: juan_cardona@outlook.com DEDICACION HORARIA: 3hs diarias

CONSIDERACIONES SOBRE EL GRUPO DE TRABAJO

Proyecto realizado de manera individual por el alumno Juan Cardona desde el mes de agosto del actual año 2015 con el apoyo del docente de la asignatura, Javier Blanqué.

JUSTIFICACIÓN DE PERFILES ADJUDICADOS A INTEGRANTES POR PROYECTO

Proyecto realizado en su totalidad por el alumno Juan Cardona.

EXPERIENCIA DEL GRUPO EN TRABAJOS SIMILARES

No se cuenta con experiencia en trabajos similares.

INTRODUCCION – DESCRIPCION – PLAN DE TRABAJO

ALTERNATIVAS DE PROYECTO A CONSIDERAR:

- APLICACION WEB
- JUEGO DE VIDEO/PC
- INTRANET/EXTRANET
- MODELO DE SIMULACION
- SISTEMA APLICATIVO PARA EMPRESA
- OTROS

El presente proyecto consiste en un sistema de ventas tipo carrito de compras para la realización de transacciones comerciales.

Dicho software brindará al usuario la posibilidad de seleccionar de una serie de productos a la venta los ítems aquellos que se desea adquirir y de esta manera confeccionar el pedido de un cliente en particular también seleccionado por el usuario.

Asimismo, tanto los clientes del sistema como los productos en venta pueden ser actualizados cuando se lo solicite así como también pueden ser añadidos al sistema nuevos productos y clientes.

Para esto, el sistema cuenta con una base de datos relacional asociada que almacena los datos necesarios para la realización de dichas tareas.

También podrá el usuario contar con una interfaz gráfica para un uso simple y de fácil comprensión con el cual interactuar de manera eficiente y clara.

Asimismo, el usuario podrá realizar solicitudes de reportes sobre las ventas realizadas desde el comienzo de las actividades de la organización pudiendo acceder a datos estadísticos como los productos vendidos o las sumas abonadas por los clientes.

Además, el código fuente se encontrará dividido en 3 capas: modelo, vista y controlador. De esta manera se separa la lógica de negocios de su interfaz gráfica lo cual permite la reutilización de código y escalabilidad.

Por último, se proveerá de una serie de testeos realizados sobre el código fuente del proyecto que fueron implementadas durante el proceso de desarrollo del software en base a lo recomendado por la metodología de programación ágil.

El código fuente del proyecto ha sido almacenado en repositorios web a lo largo de su desarrollo y se ha llevado a cabo un control de versiones las cuales se han publicado en servicios web a medida que se han completado las distintas etapas del proyecto.

INFRAESTRUCTURA Y ARQUITECTURA TECNOLÓGICA

TEMAS A CONSIDERAR:

- ARQUITECTURA TECNOLOGICA
- HARDWARE
- SOFTWARE
- REDES Y COMUNICACIONES
- SEGURIDAD Y PRIVACIDAD
- LOGISTICA
- OBRA CIVIL
- LOCALIZACION TEMPORAL Y GEOGRAFICA

El proyecto ha sido desarrollado utilizando como dispositivo de cabecera una notebook Lenovo G480 con 4 Gb de memoria y procesador Intel I3. Pantalla de 14 pulgadas.

El Sistema Operativo utilizado durante el desarrollo ha sido Windows 8.1 de 64bits, aunque también ha sido testeado en Windows 7 y algunas distribuciones de Linux como Debian 7 y Ubuntu 11.

Se desarrolla en lenguaje JAVA utilizando la plataforma Eclipse Kepler versión 2015 con el paquete JDK SE 8, update 45.

El sistema gestor de base de datos seleccionado para el proyecto fue PostgreSQL 9.4 utilizando pgAdmin III como interfaz gráfica. La conexión a dicha base de datos se realiza mediante los drivers JDBC para PostgreSQL.

La interfaz gráfica fue desarrollada utilizando la biblioteca gráfica de Swing con el apoyo de WindowBuilder, un plug in facilitado por la plataforma Eclipse.

El diagrama de clases se confecciona con la herramienta ObjectAid directamente desde el Eclipse. Una versión alternativa y simplificada de dicho diagrama es diseñado con la aplicación Dia. El diagrama de casos de uso fue realizado mediante Microsoft Visio y los diagramas de secuencia fueron diseñados con la herramienta Dia.

El código fuente de la aplicación será almacenado en repositorios Git publicados en GitHub herramienta con la cual también se llevará el control de versiones de la aplicación.

Los reportes suministrados serán confeccionados utilizando el motor de reportes Jasper Reports y diseñados a partir de la aplicación de escritorio iReports 5.6.0. También se añaden al proyecto de Eclipse las librerías mínimas necesarias para la confección de reportes JasperReports.

Finalmente, el desarrollo de clases de testeos del modelo se llevará a cabo a través de JUnit test cases.

DEFINICION DETALLADA DEL PROYECTO

SECCIONES A DOCUMENTAR DEL PROYECTO:

- ANTECEDENTES
- OBJETIVOS O PROPÓSITO
- BENEFICIOS
- TEMAS PRINCIPALES
- ALCANCE
- REQUERIMIENTOS
- PRE Y POST-CONDICIONES
- LIMITACIONES
- PRESUPUESTO INICIAL
- COSTOS OPERATIVOS (FIJOS O MENSUALES)
- CRECIMIENTO
- LÍNEAS FUTURAS

ANTECEDENTES:

Aún existen en la actualidad organizaciones donde las actividades transaccionales o de venta se llevan a cabo de manera manual o sin el soporte de un sistema que facilite la labor de los empleados.

Si bien de un tiempo a esta parte se ha crecido mucho en cuanto a la automatización de las tareas organizacionales, aún queda mucho por desarrollar y es de importancia que se continúe por esta senda.

OBJETIVOS O PROPÓSITO:

Como propósito general se busca generar nuevas soluciones de software para la automatización de actividades llevadas a cabo en organizaciones así como obtener conocimientos sobre tecnologías utilizadas en la actualidad y sus posibles implementaciones.

En particular, lo que se busca con este proyecto es lograr sistematizar las actividades de venta de una organización mediante la implementación de un software informático orientado a las transacciones comúnmente conocido como carrito de compras.

Otros objetivos son:

- Administrar y almacenar datos del sistema de manera fiable y segura a partir de la integración al proyecto de un sistema gerencial de bases de datos (SGBD).

- Implementar una interfaz gráfica para el sistema de manera tal que el usuario pueda interactuar con ella de manera clara, concisa y eficiente otorgando todas las funcionalidades necesarias y requeridas para la realización de sus actividades.

- Realizar validaciones sobre los datos ingresados por el usuario para evitar el ingreso al sistema de datos no pertinentes.

- Realizar conexiones seguras de la aplicación cliente hacia el SGBD mediante el uso de conexiones JDBC que realicen llamadas a stored procedures o vistas almacenadas en la base de datos.

- Confeccionar test de casos de uso como JUnit para un proceso de desarrollo de software afín a una metodología de programación ágil.

- Permitir al usuario insertar y actualizar nuevos productos y clientes al sistema.

- Llevar una lista con los productos que el usuario desea adquirir permitiendo inserción, eliminación y actualización de cualquiera de los ítems agregados.

- Suministrar reportes con información detallada sobre las ventas realizadas en el curso de actividades de la organización.

- Permitir al usuario establecer descuentos a clientes sobre los productos agregados al carrito bajo parámetros establecidos por él mismo.

- Separar el código fuente de la aplicación siguiendo el patrón MVC.

BENEFICIOS:

De esta manera el software logra:

- Facilitar y agilizar las tareas de venta de productos realizada por el sector de ventas de una organización.

- Mantener en resguardo, de manera independiente y organizada los datos del sistema en una base de datos relacional.

- Generar una interacción simple y clara entre el usuario del sistema y la interfaz gráfica de la aplicación cliente.

- Evitar el ingreso al sistema de datos no válidos que puedan alterar los resultados de la transacción.

- Facilitar futuras evoluciones en el comportamiento del sistema mediante la mantención del código fuente a partir de la implementación de casos de prueba.

TEMAS PRINCIPALES:

El tema principal del proyecto consiste en la elaboración de un programa informático

orientado a transacciones a partir del cual realizar ventas de productos a clientes de manera automatizada en una organización con fines comerciales o transaccionales.

ALCANCE:

Cualquier organización que realice actividades transaccionales donde existan productos en demanda con un valor determinado y clientes dispuestos a su adquisición.

Estas organizaciones pueden ser un negocio particular con atención al cliente o en general cualquier empresa pequeña o mediana con objetivos comerciales.

REQUERIMIENTOS:

Para la implementación del proyecto se requerirá un computador encargado de almacenar la base de datos y un conjunto de computadores clientes que se conectarán a dicho computador servidor y realizaran las consultas que requieran, ya sea agregar nuevas ventas confirmadas, actualizar clientes y demás.

Las especificaciones de los computadores, tanto de servidor como cliente, quedan sujetas a la organización y lo que estime apropiado.

Dicha conexión entre computadores requerirá de la existencia de una red LAN con velocidad de conexión sujeta a las necesidades de la organización.

LIMITACIONES:

El sistema no permitirá:

- Editar datos de ventas ya confirmadas.

- Confirmar pedidos a clientes que no se encuentren en la base de datos de la organización.

- Confirmar pedidos que contengan productos que no se encuentren en la base de datos de la organización.

- Editar el código unívoco de identificación establecido para un cliente o producto.

- Confeccionar pedidos a más de un cliente a la vez en una misma sesión.

- Ingresar datos que no sean válidos según lo establecido por dominio.

PRESUPUESTO INICIAL:

Se deberá contar con un presupuesto inicial mínimo de $20.000 (veinte mil pesos) que podrá ser extendido según las necesidades y exigencias de la organización.

LÍNEAS FUTURAS:

A futuro se podrá extender las funcionalidades del sistema para que permita:

- Llevar un control de stock sobre los productos en venta.

- Obtener un informe sobre las ventas realizadas por el sistema durante un período especificado por el usuario.

- Destinar un carrito de compras independiente para cada cliente del sistema.

- Llevar una lista con los productos que cada cliente desearía solicitar a futuro.

- Agregar a cada cliente atributos como domicilio, teléfono, E-mail y código postal.

- Agregar imágenes descriptivas del producto en venta.

- Llevar un sistema de descuentos mediante el uso de cupones de descuento.

- Definir usuarios con distintos permisos sobre el uso de la aplicación.

- Implementar un sistema de logueo y registro de usuarios.

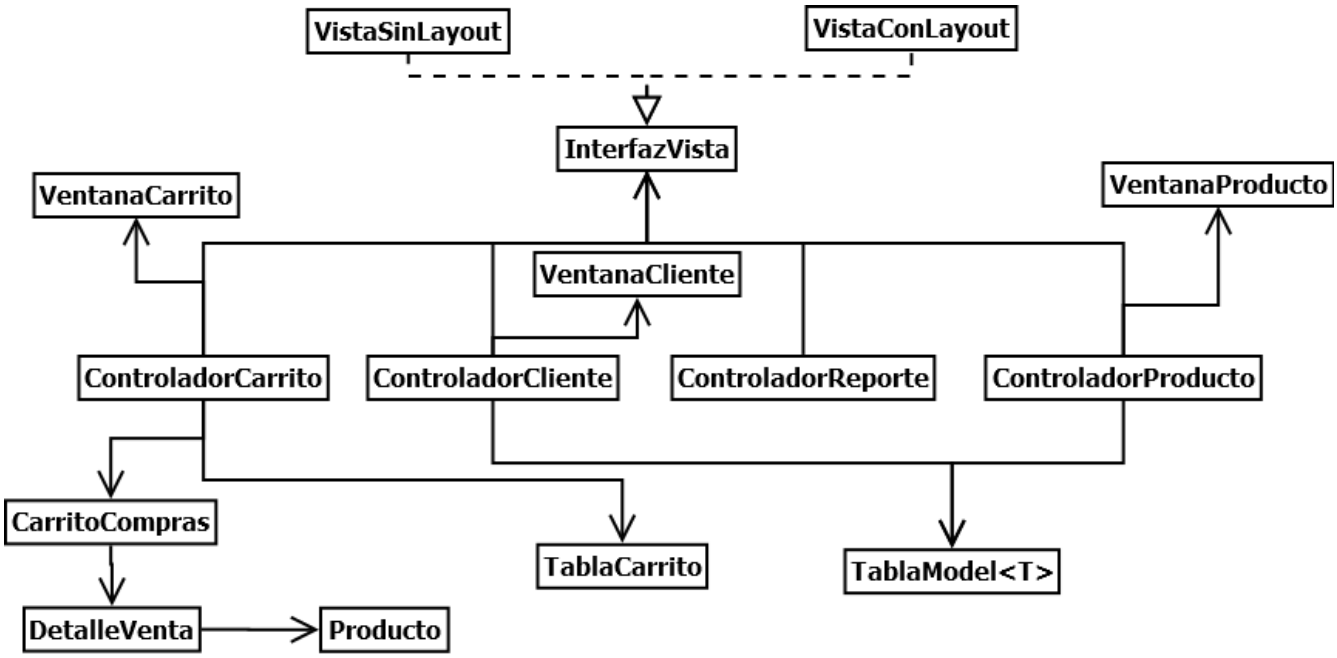
FUNCIONALIDAD DEL PROYECTO

ELEMENTOS A DOCUMENTAR:

- DOCUMENTACION UML "UNIFIED MODELING LANGUAGE"
- CASOS DE USO
- CLASES, OBJETOS Y CONCEPTOS
- ESCENARIOS Y AMBIENTE
- SECUENCIAS TEMPORALES
- EVENTOS
- MODULOS Y SUBSISTEMAS
- DESCRIPCION FUNCIONAL

UML:

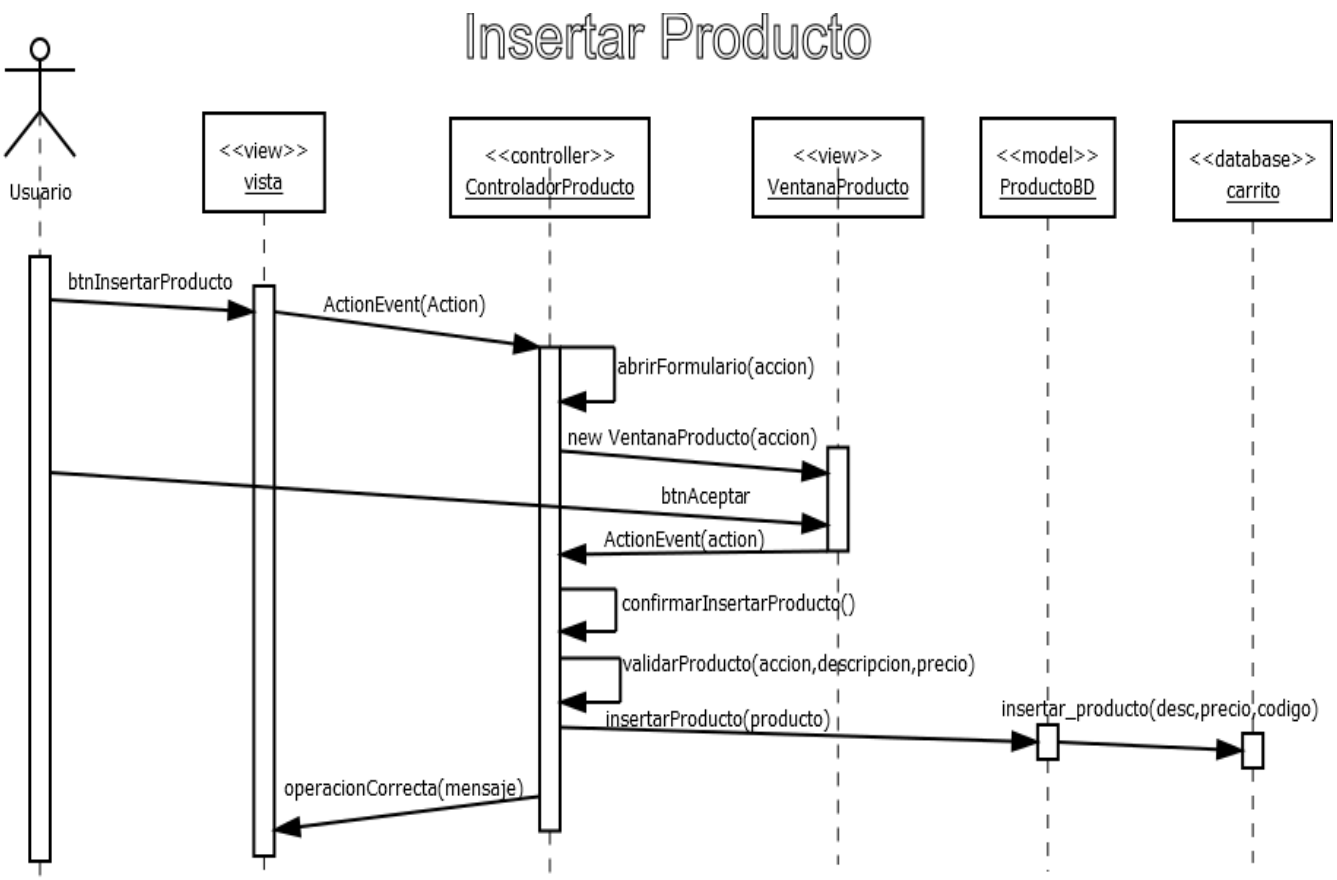
Diagrama simplificado:

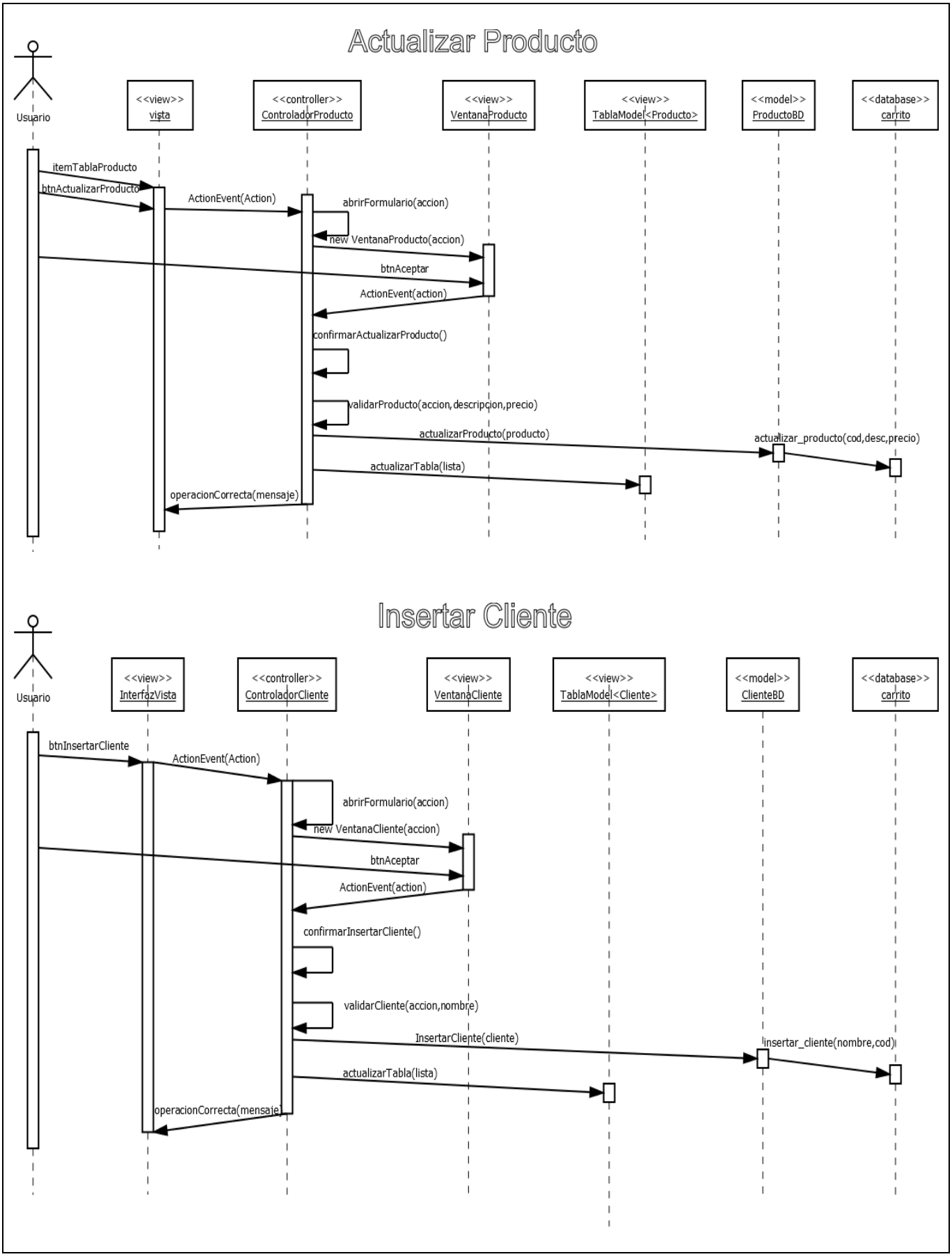


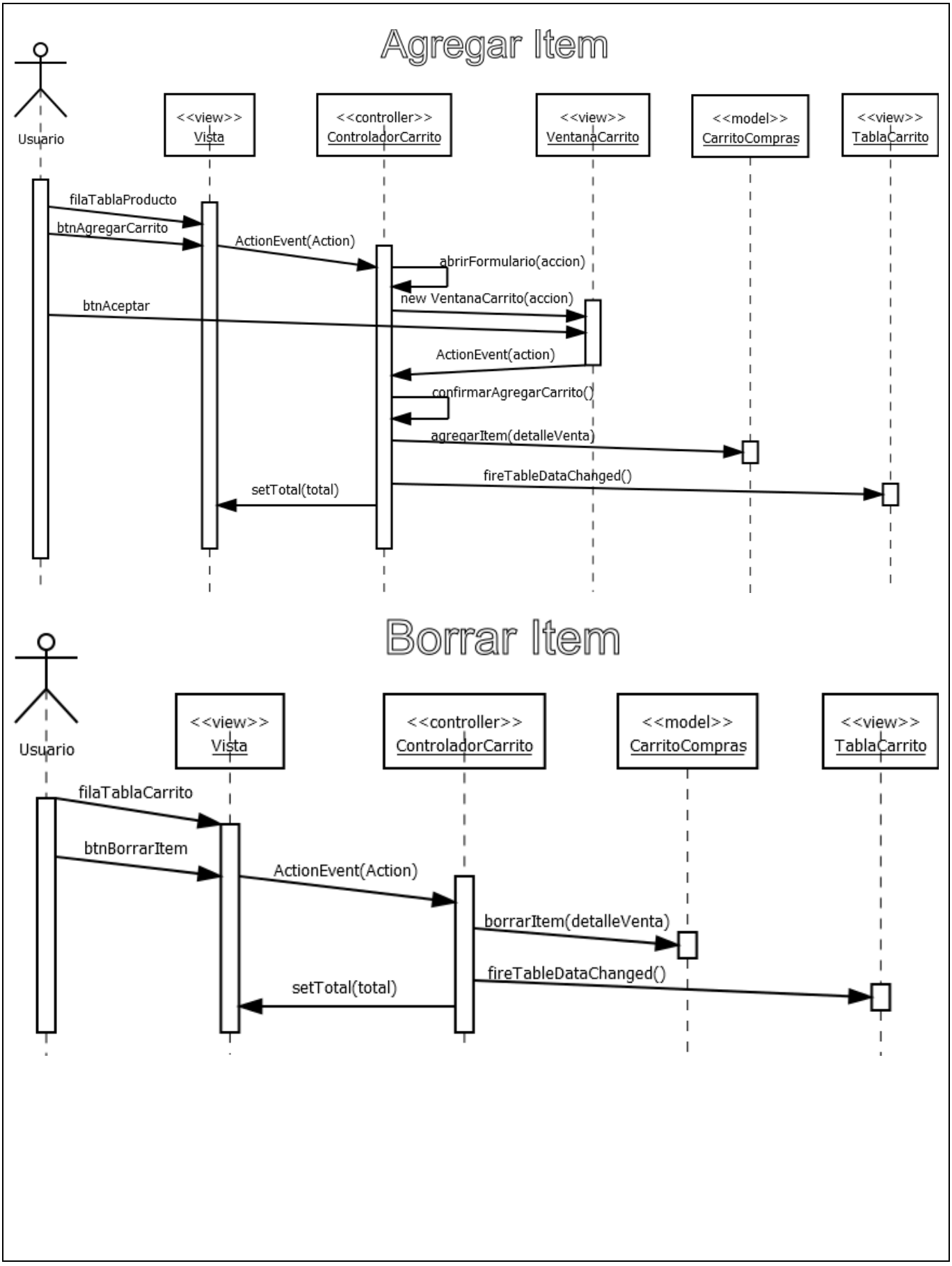
Para versión extendida ver anexo "diagramas".

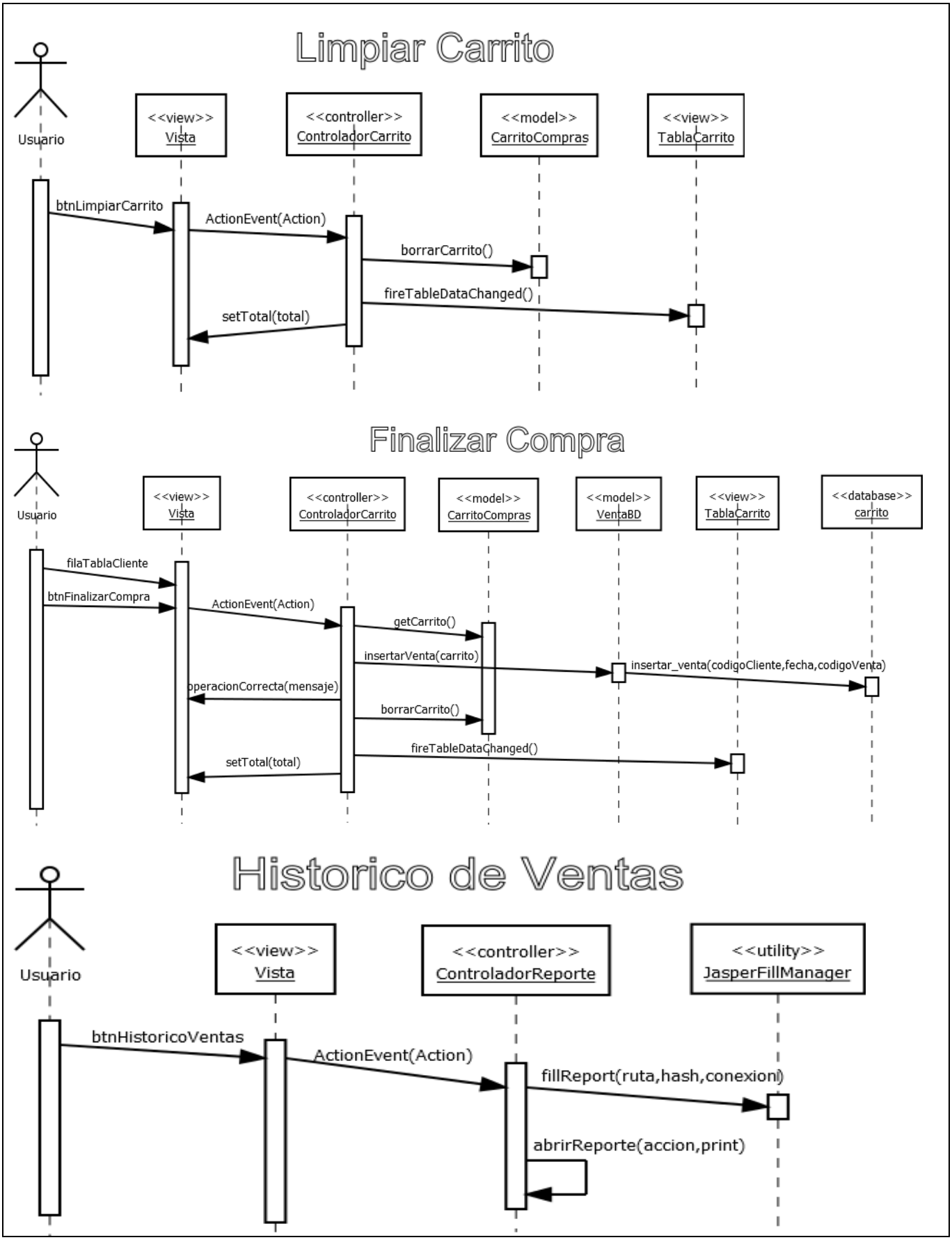
CASOS DE USO:

DIAGRAMAS DE SECUENCIA:




Actualizar Producto
Usuario
<<view>>
vista
<<controller>>
ControladorProducto
<<view>>
VentanaProducto
<<view>>
TablaModel<Producto>
<<model>>
ProductoBD
<<database>>
carrito
itemTablaProducto
btnActualizarProducto
ActionEvent(Action)
abrirFormulario(accion)
new VentanaProducto(accion)
btnAceptar
ActionEvent(action)
confirmarActualizarProducto()
validarProducto(accion,descripcion,precio)
actualizarProducto(producto)
actualizar_producto(cod,desc,precio)
actualizarTabla(lista)
operacionCorrecta(mensaje)
Insertar Cliente
Usuario
<<view>>
InterfazVista
<<controller>>
ControladorCliente
<<view>>
VentanaCliente
<<view>>
TablaModel<Cliente>
<<model>>
ClienteBD
<<database>>
carrito
btnInsertarCliente
ActionEvent(Action)
abrirFormulario(accion)
new VentanaCliente(accion)
btnAceptar
ActionEvent(action)
confirmarInsertarCliente()
validarCliente(accion,nombre)
InsertarCliente(cliente)
insertar_cliente(nombre,cod)
actualizarTabla(lista)
operacionCorrecta(mensaje)


Agregar Item
Usuario
<<view>>
Vista
<<controller>>
ControladorCarrito
<<view>>
VentanaCarrito
<<model>>
CarritoCompras
<<view>>
TablaCarrito
filaTablaProducto
btnAgregarCarrito
ActionEvent(Action)
abrirFormulario(accion)
new VentanaCarrito(accion)
btnAceptar
ActionEvent(action)
confirmarAgregarCarrito()
agregarItem(detalleVenta)
fireTableDataChanged()
setTotal(total)
Borrar Item
Usuario
<<view>>
Vista
<<controller>>
ControladorCarrito
<<model>>
CarritoCompras
<<view>>
TablaCarrito
filaTablaCarrito
btnBorrarItem
ActionEvent(Action)
borrarItem(detalleVenta)
fireTableDataChanged()
setTotal(total)


Limpiar Carrito
Usuario
<<view>>
Vista
<<controller>>
ControladorCarrito
<<model>>
CarritoCompras
<<view>>
TablaCarrito
btnLimpiarCarrito
ActionEvent(Action)
borrarCarrito()
fireTableDataChanged()
setTotal(total)
Finalizar Compra
Usuario
<<view>>
Vista
<<controller>>
ControladorCarrito
<<model>>
CarritoCompras
<<model>>
VentaBD
<<view>>
TablaCarrito
<<database>>
carrito
filaTablaCliente
btnFinalizarCompra
ActionEvent(Action)
getCarrito()
insertarVenta(carrito)
insertar_venta(codigoCliente,fecha,codigoVenta)
operacionCorrecta(mensaje)
borrarCarrito()
fireTableDataChanged()
setTotal(total)
Historico de Ventas
Usuario
<<view>>
Vista
<<controller>>
ControladorReporte
<<utility>>
JasperFillManager
btnHistoricoVentas
ActionEvent(Action)
fillReport(ruta,hash,conexion)
abrirReporte(accion,print)

DURACIÓN

ELEMENTOS A DOCUMENTAR:

- DETALLE DE TIEMPOS ASIGNADOS, TAREAS Y ROLES (POR PARTICIPANTE)
- GANTT O PERT
- DOCUMENTO PROJECT O SIMILAR

Se trabaja sobre proyecto a horario completo en días libres y aprovechando tiempo disponible por la mañana y noche para días de cursada universitaria.

Para comenzar se instala el software necesario del cual no se disponía previamente, luego se trabaja sobre las tablas y stored procedures de la base de datos comenzando por tareas simples de inserción y actualización para luego aumentar progresivamente la complejidad.

Una vez realizado lo anterior se comienza con el desarrollo en JAVA mediante Eclipse creando una clase por cada tabla generada y sus respectivos testeos JUnit.

Más tarde se desarrollan las clases necesarias para la conexión de la aplicación con la base de datos. Para esto fue necesario documentarse sobre su implementación e instalar los drivers necesarios.

Posteriormente, se desarrolla una interfaz gráfica de usuario que contiene los botones y tablas necesarios para proveer las funcionalidades mínimas implementadas del sistema. Para esto fue necesario documentarse sobre su desarrollo e implementación ya que no se contaban con conocimientos previos de la materia.

Finalizada dicha vista se procede a trabajar sobre el código ya escrito para mejorar la eficiencia de la aplicación. Para esto se dividió lo desarrollado en tres capas: modelo, vista y controlador.

Una vez limpio el código comienza la implementación de reportes Java a través de JasperReports lo cual requirió un breve lapso previo de aprendizaje sobre la herramienta y su puesta a punto en el proyecto.

Periódicamente se realizan back-ups tanto del código JAVA como de la base de datos del sistema en sistemas de repositorios git como GitHub, donde también se lleva a cabo el control de versiones del proyecto.

Diariamente se lleva una bitácora con información actualizada del estado del proyecto, tareas realizadas, problemáticas encontradas, soluciones a dichas problemáticas y tareas pendientes de revisión o a realizar a futuro.

RESULTADOS ESPERADOS A LA CULMINACION DEL PROYECTO

ESTIMACIONES A PRESENTAR:

- MEDICION PERSONAL DEL EXITO ALCANZADO
- VALORIZACION DE LOS BENEFICIOS DEL PROYECTO

MEDICION PERSONAL DEL EXITO ALCANZADO:

A nivel personal se ha conseguido:

- Consolidar conceptos, metodologías y conocimientos obtenidos a lo largo de la carrera universitaria a partir de una puesta en práctica íntegra.

- Formar una primera experiencia en puesta a punto de proyectos de software.

- Utilizar herramientas, arquitecturas, técnicas y metodologías actuales de análisis, diseño, desarrollo e implementación de software.

- Realizar un desarrollo de código centrado esencialmente en una lógica de programación orientada a objetos.

- Poner en práctica de manera informal técnicas de programación extrema como casos de prueba y desarrollo incremental.

VALORIZACION DE LOS BENEFICIOS DEL PROYECTO:

A partir de la realización de este proyecto se ha logrado:

- Sistematizar actividades o tareas llevadas a cabo en el mundo empresarial mediante un programa informático de manera tal de proporcionar nuevas soluciones de software para problemáticas existentes en dicho contexto.

- Llevar a cabo conexiones entre una aplicación y su base de datos gestionada por un SGBD lo cual permite una administración eficiente y segura de los datos almacenados.

- Desarrollar una interfaz gráfica de usuario que logre proveer las funcionalidades esperadas del proyecto de una manera interactiva y de fácil comprensión para el usuario de la aplicación.

- Disponer de casos de prueba lo cual proporciona mayor seguridad sobre futuras ediciones o adiciones de código a la aplicación.

- Implementar solicitudes de reportes del usuario a la aplicación para proveer de información estadística y detallada de las actividades de la organización.

PROBLEMAS HALLADOS A LO LARGO DEL PROYECTO Y SUS SOLUCIONES

RELATAR:

- PROBLEMAS Y SOLUCIONES HALLADAS
- LECCIONES APRENDIDAS
- ANEXO DE BITACORA DETALLADA POR SESIÓN

BITACORA:

```
PROYECTO: CARRITO DE COMPRAS

- 29/08/2015:

      - Comienzo del proyecto.
      - Pruebas en Eclipse desde máquina virtual Debian 7.
            - Errores en tema de permisos de carpeta Workspace.
              Solución:
                  #chmod -R ugo+rw /home/juan/workspace
                  Y reinicio del eclipse.
            - Problema con ventanas de la interface de Eclipse:
                  no se visualizan opciones de menu situadas en sector
            inferior de ventana.
              Sin solución.
      - Creación de Clases con y sin main.
      - Pruebas con JUnit Test Case
      - Instalación PostgreSQL en maquina virtual Debian 7
            Problemas para iniciar el servicio
            Sin solución (solucionado en fecha 02/09/2015)
      - Instalación PostgreSQL en Windows 8.1


- 30/08/2015:

      - Pruebas de pgAdmin3 en Windows 8.1


- 01/09/2015:

      - Comienza desarrollo de base de datos postgreSQL en Windows 8.1
      - Se crea la base de datos CarritoDB
      - Se agregan tablas de producto, cliente, venta y detalleVenta.
      - Se investiga sobre stored procedures en PostgreSQL.


- 02/09/2015:

      - Solucionado problema de inicio de servidor PostgreSQL en maquina
      virtual Debian.
      - Problemas en instalación de pgAdmin3 en maquina virtual Debian.
            Sin Solución.


- 03/09/2015:

      - Creado procedimiento insertar_producto(), actualizar_producto() y
```

```
            obtener_producto().
      - Creado procedimiento de prueba llamado obtener_producto2() como
      ejemplo de uso de loops para futuras implementaciones.


- 04/09/2015:

      - Creado procedimiento insertar_cliente() e insertar_detalleventa()
      - Cambio tipo de datos de codigos univocos de integer a serial
      (cada uno con su respectiva sequence).
      - Procedimientos de insert actualizados para autoincrementar codigo
      unívoco en caso de no ser ingresado por usuario.
      - Creada vista historico_ventas para obtener todas las ventas
      realizadas
      - historico_ventas utiliza las siguientes vistas:
            - Subtotal_por_producto:
                  Cada venta con cada uno de los productos solicitados por
            el cliente.
            - Subtotal calculado a partir de la multiplicacion del precio
                  del producto por la cantidad solicitada.
            - Total_por_venta:
                  Cada venta y el total calculado a partir de los
            subtotales obtenidos por vista subtotal_por_producto.
      - Creada vista historico_ventas2 con misma funcionalidad que
      historico_ventas pero sin utilizar otras vistas.
      - Creado procedimiento obtener_venta mediante lógica de loops
      extraida de obtener_producto2.
      - Eliminado procedimiento obtener_producto2.

      - Comienzo de desarrollo de la aplicación JAVA en Eclipse.
      - Se crea proyecto CarritoCompras.
      - Desarrollada clase Producto incluyendo testeos JUnit desde clase
      Test.
      - Desarrollada clase Cliente incluyendo testeos JUnit desde clase
      Test.
      - Desarrollada clase Venta incluyendo testeos JUnit desde clase
      Test.


- 05/09/2015:

      - Aprendizaje sobre generación de conexiones entre base de datos y
      aplicación JAVA (uso de Class.forName(), Connection(), etc).
      - Pruebas en Eclipse utilizando como base documentación
      proporcionada por Prof. Blanque mediante la plataforma virtual
      de Prog. III.
      - Carpeta: ..\Programación III\JAVA Listado de Clientes.
      - Se agrega al proyecto de Eclipse la libreria postgresql-9.4-
      1202.jdbc4.jar
      - Pruebas de conexión sobre algunas tablas y views.
            Pendiente: conectarse a procedimientos.
      - Desarrollada clase DetalleVenta incluyendo testeos JUnit desde
      clase Test.
      - Creada clase Conexion.
            Problemática: ¿Qué devuelven métodos? Si no se puede generar
            conexión ¿qué pasa?.
```

```
      - Agregado a cada método de conexión un Throw Exception.
            Pendiente: Investigar sobre el tema.
      - Aprendiendo sobre consultas, inserciones y demás hacia la BD a
      través de procedimientos.
      - Se edita la funcion insertar_producto a la cual se le ingresa
      descripcion y precio de producto y devuelve el código auto
      generado. De esta manera se simplifica la implementación de la
      conexión desde la aplicación sin perder su funcionalidad.
      - Desarrollo de clase ProductoBD que interactua con la tabla
      producto de la BD.
            - Generando método insertar_producto.
                  - Errores de ejecución con origen aun indeterminable.
                  Debugging sin éxito.
                  - Sugerencia: ¿Agregar try-catch para detectar origen de
                  dicho error?
                  - Dificultad: necesidad de aprender sobre el uso de
                  dichas sentencias previo a su implementación.

      Pendiente:
            1) Arreglar problema en clase ProductoBD.
                  Sugerencia:
                        llamar un procedimiento de prueba lo más simple posible
                        e ir aumentando complejidad de a poco
                  Objetivo de dicho proceder:
                        Entender el funcionamiento de las disintas clases y
                        métodos que entran en juego.
                  Dichas Clases y métodos son:
                        CallableStatement, PrepareStatement, prepareCall,
                        prepareStatement, execute, executeQuery, executeUpdate,
                        etc.
            2) Pulir procedimientos derivados de insertar_producto.
            3) Insertar_producto3 es de prueba, eliminar luego de solucionado
            1).


- 06/09/2015:

      - Desarrollo de clase ClienteBD
      - Duda:
            ¿try-catch o throw Exception?. Por el momento se elige segunda
            opción. Revisar a futuro.
      - Codeando metodo insertar_cliente.
            - Mismos problemas que en ProductoBD.
            - Bug encontrado: mal definido parametro de salida, cambiado
              por: callableStatement.registerOutParameter();
            - Mismos problemas
            - Bug encontrado: se llamaba a funcion insertar_cliente en
                  lugar de insertar_cliente2 (funcion temporal de prueba).
            - Solucionado parcialmente. Problema:
                  execute() devuelve falso pese a que la tabla
                        cliente es cargada correctamente.
            - Solucionado: el false de execute() no tiene que ver
                  con el éxito o no de la transacción.
      - Se vuelve a trabajar sobre ProductoBD y se solucionan algunos bugs.
      - Mismos problemas.
      - Bug detectado: parece que hay problemas con el parametro de tipo double
```

de la funcion.
- Solucionado: Se castea de double a bigDecimal.
- Problemática: Cuando hay un error de ejecución como los tratados arriba, el programa lanza una excepción pero la conexión no se cierra.
    - Consecuencia:
    Atributo código de tipo serial (ejemplo, codigo_cliente) no se autogenera correctamente (genera un corrimiento).

- Para la próxima:
    - Antes de proseguir con el desarrollo de nuevos métodos y clases y generar aun más errores, trabajar sobre lo ya codeado
        Objetivo:
            Generar un código más seguro con control de erorres.
        Para esto:
            Documentarse sobre Try-catch e implementarlo en el sistema previo testeos de prueba.
- Averiguar como implementar correctamente testeos de JUnit en c clases que se comunican con BDs.

- 07/09/2015:

- Se investiga sobre try-catch y sus posibles implementaciones.
    Bibliografia: Thinking in JAVA, google, etc.
- Probando ejemplos de Thinking in JAVA.

Para mañana:
- Seguir con las pruebas de Try-Catch.
- Buscar manera óptima de implementar testeos de JUnit a conexiones con BD
- Preparar el terreno para implementar ambas en las clases ya creadas.
- Realizar back-up de todo.

- 08/09/2015:

- Se genera clase ConexionConTry en base a Conexion.
    - Comentarios en cada apartado.
    - Implementación de Try-catch
- Pruebas de ConexionConTry en Clase de testeos simples TestVenta.
- Se cambia metodos de ConexionConTry a static para simplificar control de excepciones.
- Se agrega método deshacerCommit a ConexionConTry para no confirmar consulta a BD cuando hay errores.
- Probando ConexionConTry en ClienteBD
- Nuevo ClienteBD con implementación de control de errores y de clase ConexionConTry.
- Nuevo ProductoBD con implementación de control de errores y de clase ConexionConTry.
- Se elimina clase Conexion para reemplazarla definitivamente por ConexionConTry ahora renombrada como Conexion.
- Investigando sobre cómo realizar testeos JUnit óptimos con conexiones a BD.
    - Se recomienda utilizar otras herramientas como DBUnit, SQLUnit o realizar mocks.

```
            - Se deja problemática en suspenso por el momento.
      - Se desarrolla el método actualizarProducto en clase ProductoBD.
      - Problema general: executeUpdate siempre devuelve 0.
            - ¿Bug del driver?

      Pendiente:
            - Terminar métodos de productoBD.
            - Investigar sobre métodos de clase y sus implicancias.
            Back-Up.


- 09/09/2015:

      - Creado método getProductos en ProductoBD para guardar todos los
      productos en un arreglo.
      - Desarrollando método obtenerProducto de clase ProductoBD.
            - Problema: procedure obtener_producto devuelve SETOF producto,
            ¿Cómo realizar el call desde el método JAVA?
            - Dudas: ¿Es necesario que devuelva un SETOF producto siendo que
            solo devolveria una fila?
                  - ¿Cambio a SETOF record? Investigando sobre el tema.
            - Solucionado: Se utiliza PreparedStatement.

      Pendiente:
            - Proseguir con DetalleVentaBD o VentaBD
            - Investigar sobre métodos de clase y sus implicancias.
            - Back-Up.


- 10/09/2015:

      - Desarrolladas clases VentaBD y DetalleVentaBD junto con sus respetivos
      métodos de inserción.
      - ¿Cómo hacer para que cuando inserto una venta realize tambien un
      llamado a insertarDetalleVenta?
            - Investigar sobre el tema.
            - Idea: Cuando inserto una venta tener los datos pertenecientes al
            detalle de cada producto solicitado y llamar a insertarDetalleVenta
            desde allí.
                  - Duda:
                        - ¿Cómo le doy los datos a llenar en detalleVenta?
                        - ¿Qué hacer con conexión?. Investigar y realizar
                        pruebas.

            - Solucionado:
                  Paso a insertar_ventas un arreglo de detalleVentas como
                  parámetro y a su vez dentro de este método llamo a
                  insertarDetalleVenta dandole como parámetro la conexión
                  establecida.

      - Pendiente: Averiguar si esta alternativa es la mejor o si es posible
      optimizarlo.
      - Se desarrolla método para cargar todas las ventas realizadas junto con
      el detalle de cada una.
            - Para tal fin se crea una nueva vista en la BD llamada
            obtener_detalle_ventas.
```

- 11/09/2015:

    - Terminado desarrollo de modelo de datos del proyecto.
    - Se aborda el tema de interfaz gráfica.
    - Investigando qué software se utiliza para dicho desarrollo.
        - Leyendo Thinking in JAVA - Bruce Eckel, 2da edición. Capítulo 13, Creating Windows & Applets.
        - Pruebas básicas otorgadas por dicha edición.
            - Problema con ejemplo 1:
                Restricción de acceso a libreria.
            - Solución: Quitar y luego volver a agregar JRE System Library.

- 17/09/2015:

    - Instalación de WindowBuilder.
    - Pruebas simples utilizando WindowBuilder.
    - Probando botones de "Ver Productos", "Ver Clientes" y "Ver Ventas" con sus respectivas acciones.
    - Sálida en JTextArea.
        - Se agrega método getClientes a ClienteBD con su respectivo testeo.
        - Se agrega método getVentas a VentaBD con su respectivo testeo.
    - Aprendiendo a usar JTable.

- 18/09/2015:

    - Creado Boton de Insertar Producto que llama a ventana para ingresar datos.
    - Creada VentanaInsertarProducto.
        - Agregados botones Aceptar y Cancelar con sus respectivos actionListener.
        - Agregada validación de datos ingresados.
    - Creado Boton de Insertar Cliente que llama a ventana para ingresar datos.
    - Creada VentanaInsertarCliente.
        - Agregados botones Aceptar y Cancelar con sus respectivos actionListener.
        - Agregada validación de datos ingresados.
    - Trabajando en JTable TablaProductos.
        - Se crea una nueva clase llamada TablaProducto que hereda de DefaultTableModel.
        - Se sobreescriben métodos de getValueAt, getColumnCount y otros para que actuen sobre mi arreglo de productos extraidos de la BD.
        - Problema:
            Se inserta producto pero no se actualiza tabla.
            - Solución:
                Agrego a clase InterfazPrincipal método actualizarTabla la cual es llamada por VentanaInsertarProducto.
        - Problema:
            - Al pasar a VentanaInsertarProducto la instancia de InterfazPrincipal como parámetro WindowBuilder lanza error y

```
                no permite ver ventana de diseño.
                - Solución:
                      Luego de correr, el error dejó de mostrarse. Solución
                      indefinida.
      - Lo mismo realizado con VentanaInsertarProducto se realiza con
      VentanaInsertarCliente.
      - Trabajando en JTable TablaClientes.
            - Se crea una nueva clase llamada TablaCliente que hereda de
            DefaultTableModel.
            - Se sobreescriben métodos de getValueAt, getColumnCount y otros
            para que actuen sobre mi arreglo de clientes extraidos de la BD.
      - Se agrega boton Actualizar Producto
            -Se agrega VentanaActualizarProducto.
            - Agregada funcionalidad para tomar de la tabla el item que
            seleccionó el usuario.
            Agregada funcionalidad de rellenar campos de texto con la
            información del producto seleccionado.
            - Problema:
                  Luego de actualizar producto, no se actualiza tabla.
                  - Solución:
                        Faltaba campo código de producto necesario en la llamada
                        a la BD.
      - Elimino botones "ver producto", "ver cliente" y "ver ventas" creados
      ayer junto con JTextArea.
            - Razon:
            Solo se realizo como práctica y su funcionalidad fue reemplazada
            por el uso de tablas.
      - Realizado Backup tanto de Proyecto Eclipse como de BD PostgreSQL


- 19/09/2015:

      - Creado boton Agregar al carrito
            - Se codea ventanaAgregarCarrito.
            - Se agrega funcionalidad de validación de datos ingresados por
            usuario.
      - Creo clase CarritoCompras que contiene arreglo de detalleVenta.
            - Agrego métodos:
                  - addDetalleVenta: agrega detalleVenta al carrito.
                  - getCarrito: devuelve arreglo de carrito.
                  - borrarCarrito: Limpia el carrito.
                  - productoEnCarrito: true si codigoProducto ingresado se halla
                  en carrito.
                  - agregarCantidad: suma nueva cantidad al existente.
                  - getDetalleVenta: Se ingresa codigoProducto y devuelve
                  DetalleVenta;
                  agregarDescuento: suma el nuevo descuento al existente.
            - Testeos JUnit en clase TestCarritoCompras.
            - Optimizo código utilizando el método indexOf del arrayList.
                  - Cuidado: el equals da true pese a no asignar codigoVenta.

      Pendiente:
            - ¿IndexOf funciona pese a no asignar codigoVenta? Investigar.
            - Metodo borrarItem.
            - Codear TablaCarrito
            - Default para cantidad y descuento en ventanaAgregarCarrito.
```

```
            - Probar paneles con distintos Layouts.


- 20/09/2015:
      - Duda solucionada: "¿IndexOf funciona pese a no asignar codigoVenta?
      Investigar."
            - Atributos del objeto son asignados valores por defecto.
      - Agregado método borrarItem y su respectivo testeo.
      - Realizada clase JTable TablaCarrito que utiliza CarritoCompras.
      - Agregado método getSubtotal a DetealleVenta y a CarritoCompras.

      Pendiente:
            - Default: segunda vez no aparece.
            - Que no aparezca mismo producto dos veces.
            - ¿Quién calcula descuento (descuento * cantidad)?
            - ¿Por qué fireTableDataChanged actualiza sin volver a llamar a
            carrito.getCarrito?


- 21/09/2015:

      - Agregada funcionalidad para que dos productos iguales aparezcan una
      sola vez en carrito.
      - Código de dicha funcionalidad optimizado.
            - Antes lógica se encontraba en clase InterfazPrincipal, ahora se
            situa en CarritoCompras.
            - Cambios de método addDetalleVenta genera errores en testeo JUnit.
                  - Reparación pendiente.
      - Agregados botones de carrito
            - Borrar Item: Quita item seleccionado del carrito.
            - Limpiar Carrito: Borra todos los productos del carrito.
            - Finalizar Compra: Confirma el pedido.
      - Codigo de InterfazPrincipal comentado y optimizado.
            - Se soluciona siguiente problema:
                  - En actionListener no podia enviar InterfazPrincipal con this
                  por lo que instanciaba clase ventana desde fuera.
                  Solución:
                  InterfazPrincipal.this

      Pendiente:
            - Al confirmar compra otorgar al usuario el número de venta.
            - Total a pagar.
            - Informe de Ventas.


- 22/09/2015:

      - Optimizadas clases de paquete interfaz con metodos privados para
      facilitar lectura de codigo.
            - A futuro: implementar herencia en clases tabla y ventana.
      - Reparado TestCarrito.
      - Optimizo testeos JUnit
            - En lugar de una sola clase para todos los tests creo clases
            particulares para cada clase del modelo en paquete test.
      - Agregado Total del carrito (método, labels, campos, etc.)
      - BackUp realizado.
```

```
      - Actualizo Eclipse a Mars 4.3
      - Instalado el ObjectAid para genera UML

      Pendiente:
            - DetalleVenta, ¿tiene atributo Venta?
            - Clase Conexion no es utilizada segun UML, revisar llamadas a
            dicha clase.


- 23/09/2015:

      - Optimizado código de clase Conexion.
      - Generado UML.
            Dudas generadas:
                  - ¿Por qué diagrama no muestra relación entre VentaBD y
                  DetalleVentaBD?
                        - ¿Tendrá que ver con que DetalleVenta usa la misma
                        conexión que VentaBD?
                        - Solucion:
                              Aparece como dependencia.
                  - CarritoCompras se relaciona con Producto en lugar de con
                  DetalleVenta.
                        - ¿No sería más eficiente si trabajara con DetalleVenta?
                        - Solucionado:
                              Implementada dicha propuesta y actualizado testeo
                              JUnit.
                  - Venta debiera tener objeto Cliente.
                        Hecho.
                  - DetalleVentaBD parece no tener uso en diagrama.
                        ¿Lo tendría en caso de implementar Histórico de Ventas?
                  - Uno a uno entre interfaz y ventanas, ¿Sera recomendado?
      - Generado segundo UML con las nuevas implementaciones.


- 27/09/2015:

      - Investigando sobre back-ups
      - Investigando sobre Git
            Bibliografia: "Pro Git".
      - Se realizan pruebas simples.


- 28/09/2015:

      - Se realiza repositorio Git en carpeta: ..\Eclipse
      Workspace\CarritoCompras
      - Utilizo .gitignore extraido de la web para ignorar archivos
      autogenerados.
      - Commit realizado.
      - Algunos comandos:
            - git init
            - git log -1 -p -> ver cambios introducidos en el ultimo commit
            - git commit -a -m 'mensaje en commit' -> realiza commit (-a: stage
            de archivos modificados)
```

- 30/09/2015:

    - Se realizan pruebas de uso de Branch en git.
      Algunos comandos utilizados.
        - git branch branch_prueba
          crea branch llamado branch_prueba
        - git checkout branch_prueba
          cambia a branch branch_prueba
        - git checkout -b branchprueba
          integra los dos comandos anteriores en uno solo.
        - git branch
          muestra branches existentes en proyecto
        - git log --decorate
          muestra commits solo de branch actual (al que apunta HEAD)
        - git log --oneline --decorate --graph --all
          mismo que arriba pero de todos los branch.
        - git merge branch_prueba
          branch en que estoy posicionado apunta a branch_prueba
        - git branch -d branch_prueba
          eliminar branch_prueba

- 01/10/2015:

    - Se realizan pruebas de uso de servidor
        - git clone <directorio_local>
          Para guardar una copia desde un directorio del filesystem propio.
        - git remote -v
          Si se realizó un clone especifica que URLs utilizan el fetch y push
        - git fetch <origin>
          Para traer data de proyectos remotos
        - git push <origin> <branch>
          Enviar a servidor la data actualizada.
        - Subido proyecto a:
          https://github.com/Juancard/Carrito-de-compras
    - Se realizan pruebas del proyecto en PC de escritorio con Windows 7.
        - git clone https://github.com/Juancard/Carrito-de-compras
        - En PostgreSQL:
            - Creo nueva base de datos llamada "carrito"
            - Restauro backup en dicha base de datos con restore
            Pendiente: Conectarme a base de datos de manera remota
        - En Eclipse:
            - Creo un nuevo proyecto java.
            - Click derecho en src -> import -> fileSystem y selecciono carpeta src localizada en donde se realizó el git clone.
            - Se agrega libreria JUnit
              Add library -> JUnit
            - Se agrega JDBC Driver para postgreSQL
                - Bajar JDBC de página oficial
                - en Eclipse click en add external JAR y luego seleccionar el driver descargado.

    - Trabajando sobre conexión a base de datos de forma remota donde

```
                - Direccion de red privada: 192.168.1.0/24
                - Servidor: Mi notebook personal.
                      Ip privada: 192.168.1.105.
                - Cliente: PC de escritorio como cliente.
                      Ip privada: 192.168.1.101
          Para esto:
          - En Servidor:
                - Editar archivo pg_hba.conf y agregar direcciones de red
                aceptadas.
                      - En este caso:
                            192.168.1.0/24
                - Editar archivo postgreSQL.conf y setear
                listen_addresses = '*' (sin comillas).
                - Generar nueva regla en firewall para aceptar conexiones en
                puerto 5432.
          - En Cliente:
                Editar clase Conexion para conectarse a base de datos remota:
                - DB_URL =
                jdbc:postgresql://192.168.1.105:5432/carrito


- 06/10/2015:

     - Confección del anteproyecto.
          - Se describe brevemente el proyecto mediante una introducción.
          - Se establecen los objetivos generales y especificos del proyecto.
          - Se listan los recursos utilizados.
          - Se confecciona el cronograma de actividades del proyecto.
          - Se especifica la bibliografía de referencia.
     - Anteproyecto entregado al docente.


- 09/10/2015:
     - Objetivos del proyecto
          Duda:
                ¿Objetivos del proyecto, del estudiante o de ambas?
     - Confección de la documentación del proyecto
          - Realizada la introducción y descripción del proyecto.
                Duda:
                      ¿Plan de trabajo?
          - Especificada la arquitectura e infraestructura del proyecto.
     - Trabajando sobre la definición detallada del proyecto.
          Duda:
                ¿Temas principales?
     - Requerimientos.
          Duda: ¿Cómo se pide especificarlos?
     - ¿Pre y post condiciones?
     Pendiente:
          Consultar sobre dudas generadas a Blanqué.


- 10/10/2015:
     - Dudas del día anterior resueltas en clase.
     - Corregidos los apartados de "Objetivos", "Requerimientos" y "Temas
     principales".
     - Determinadas las limitaciones del sistema.
```

```
- 11/10/2015:
      - Esbozo de diagrama de Casos de Uso


- 17/10/2015
      - Dudas consultadas a Javier Blanqué
      - Se resuelven cuestiones sobre los diagramas a presentar en el proyecto.
      - Se pulen los diagramas de clases y de casos de uso.


- 20/10/2015:
      - Completado apartado "Duración" de la documentación.
      - Completado apartado de "Resultados esperados" de la documentación.


- 22/10/2015:
      - Se realizan mejoras sobre el formato de la Bitácora y se agrega a la
      documentación del proyecto.
      - Finalizada primera versión del proyecto.


- 18/11/2015:
      - Se revisa bibliografia de asignatura Base de datos II
      - A futuro:
            - Implementar ideas extraidas de documento:
              "Introduccion al uso de JPA 2 genérico"
              por Guillermo Cherencio


- 19/11/2015:
      - Se lleva a cabo el desarrollo de clase genérica TablaModel<T>
            - Desarrollo basado en documento
              "Introduccion al uso de JPA 2 genérico".
            - Con esto se busca reutilizar código mediante el uso de reflection
            - TablaModel<T> reemplazará las clases
               TablaProductos, TablaClientes y TablaCarrito
      - Para resguardar anterior código:
            - Creo una nueva Branch en el repositorio Git llamada
               "prueba_tabla_model"
      - Se cambia parametros de entrada de la clase tablaProductos
            - Antes recibia productoBD, ahora recibe lista de productos:
               productoBD.getProductos()
      - Mismo con clase TablaClientes y TablaCarrito.
      - Se implementa TablaModel<Producto>.
            - Problema:
                  - Class.forName("Producto") arroja excepcion.
            - Solucion:
                  - Class.forName("carrito.Producto")
                  - Es decir, se agrega al nombre el paquete en el que se
                    encuentra la clase.
      - Mismo para TablaModel<Cliente>
      - Implementando TablaModel<DetalleVenta> para el carrito
            - Problemas:
                  - Algunos atributos calculados como Subtotal o descuento entre
```

```
                  otros no aparecen en tabla ya que no son atributos de clase.
                - Ademas aparecen atributos como Producto y Venta que no se
                  Deseaban agregar a la tabla
          - Solución
                - Por ahora se vuelve a implementación anterior.
                - Revisar a futuro.
     - Transformo clase Conexion en ConexionUtility y cambio metodos a static.
     - Diagramdo UML con actualizaciones.
     - Documentación actualizada.


- 24/11/2015
     - Se crea clase "JTableCofig" en paquete "anotaciones".
     - Se agrega llamada a dicha anotación en clases Cliente y Producto.
     - Se realiza Reflection en clase TablaModel para obtener anotaciones.


- 29/11/2015:
     - Problemática:
          - Vista no es independiente del modelo.
          - Demasiado código en Vista.
          - Necesidad de implementar un nexo entre modelo y vista.
     - Investigando sobre modelo vista controlador (MVC).


- 30/11/2015:
     - Registro la version v1.0.0 del software en github.
     - Creo una nueva branch "controlador"
     - Nueva clase "IniciarCarrito" con un main que crea un objeto de la vista
que se quiera
          implementar (en este caso "InterfazPrincipal")
          - Antes el main se hallaba en la vista.
          - Esto facilita el crear nuevas vistas.
     - Creo una interfazVista con los métodos fundamentales que cualquier
vista que se vaya
          a desarrollar tenga que implementar.
     - Implementado controlador para insertar producto.
          - Problemas:
                - antes data era validad por vista ahora ¿Quién valida la
data?
                - ¿Controlador que evento escucha?
                     Si escucha en boton "aceptar" como obtiene los datos del
producto?
     - Creo otro proyecto aparte donde realizo pruebas con diferentes
implementaciones.
     - Investigando sobre distintas implementaciones de MVC en Java Swing.


- 02/12/2015:
     - Trabajando sobre Insertar Producto.
     - Cuestiones solucionadas:
               ¿Quién valida? para facilitar implementación, el controlador.
               ¿Controlador qué evento escucha? todos los eventos tanto de la
               vista principal como los de las ventanas que el mismo solicita
               abrir.
     - Problemas encontrados:
```

```
            - Tablas:
                  - ¿las table model son parte de la vista o del modelo?
                        - Para facilitar implementación dejo como estaba, es
                          decir, los modelos de la tabla son parte de la vista.
                  - ¿Cómo actualizar esas tablas?
                        - Para facilitar implementación dejo como estaba,
                          es decir, cada table model tiene un metodo actualizar
                          tabla que recibe del controlador el arreglo de objetos
                          que tiene que mostrar.
                        - Inconvenientes:
                              - Debo importar clases Producto, Cliente y Carrito
                                a la vista para que realice reflection.
                              - ¿Sera eso permitido en el patrón MVC?
            - ¿Demasiados métodos en Interface InterfaceVista?
      - Trabajando sobre Actualizar Producto
      - En desarrollo:
            - Utilizar una sola ventana que reemplace "VentanaInsertarProducto"
              y "VentanaActualizarProducto".
            - La nueva VentanaProducto tendrá atributos con privacidad
              protected que
              seran seteados por la vista principal si fuera necesario.

            - Desarrollo finalizado

      Para la proxima:
            - Pensar en la posibilidad de utilizar mas de un controlador.
            - Queda por codear la implementacion de controlador para las
acciones:
                  - Insertar Cliente
                  - Agregar Carrito
                  - Quitar Item Carrito
                  - Limpiar Carrito
                  - Confirmar Compra


- 03/12/2015:
      - Desarrollo de VentanaInsertarCliente.
      - Se cambia nombre a VentanaCliente permitiendo utilizar la misma ventana
            para proporcionar diferentes funcionalidades trabajando sobre los
            datos del cliente.
      - Desarrollo de funcionalidad Agregar al carrito.
      - VentanaAgregarCarrito renombrada a VentanaCarrito
      - ¿Ventanas podrian heredar de algo asi como ventanaFormulario?
            - Tal vez reduciria codigo de abrirFormulario.
            - Revisar a futuro.
      - Trabajando sobre Modelos de tablas
            - Dudas:
                  - ¿El controlador mantiene un arreglo con los clientes,
                    productos y los items del carrito o los va a buscar a la BD
                    cada vez que lo requiere?
                  - En cada caso ¿Cómo actualizaria las tablas de la vista?
            - Decision:
                  - Se decide buscar en la BD los productos y clientes cada vez
                    que se requieran.
                  - Las tablas se actualizan dando el nuevo arreglo de manera
completa.
```

```
      - Finalizada implementación de Controlador.
      - UML actualizado.
      - Se hace un merge de la branch master hacia la branch "controlador"
          en la cual se realizó el desarrollo.
      - Se publica versión v1.1.0 del software.

      A futuro:
            - Ventanas de formularios que hereden de VentanaFormulario
            - Dividir Controlador en ControladorCliente, ControladorCarrito y
                ControladorProducto
            - Revisar implementacion Table Model (¿vista o modelo?, etc)
            - ¿Demasiados métodos en InterfazVista?
            - Controlador y modelo ¿Son dependientes?
            - Implementar vista utilizando BorderLayOut
            - Estudiar nuevo UML
            - Actualizar documentación con el MVC implementado.


- 04/12/2015:
      - Objetivo:
            - Dividir la clase Controlador en
                  - ControladorProducto
                  - ControladorCliente
                  - ControladorCarrito
      - Se genera una nueva branch "controladores"
      - Desarrollo de ControladorProducto.
      - Se rompieron las annotations. Revisar!
            - Arreglado.
            - Mal asignado en la clase reflection de TablaModel
                  el paquete donde se ubican.
      - Desarrollo de ControladorCliente.
      - Desarrollo de ControladorCarrito.
      - UML actualizado.
      - Cambio priacidad de atributos de VentanaProducto de protected a
        privados y armo getters y setters de botones y campos de texto.
      - Objetivo cumplido.
      - Se hace un merge de lo desarrollado en el repositorio git.
      - Nuevo Objetivo:
            - Que las clases VentanaProducto, VentanaCliente y VentanaCarrito
                  sean administradas por el controlador y no por la vista.
      - Armo nueva branch "ventana"
      - Trabajo sobre VentanaCliente.
      - Trabajo sobre VentanaProducto.
      - Trabajo sobre VentanaCarrito.
      - Objetivo Cumplido.
            - InterfazVista ya no tiene tanta cantidad de métodos ya que la
administracion
                  de los formularios queda a mano de los controladores.
      - Se hace un merge de lo desarrollado en el repositorio git.
      - Nuevo Objetivo:
            - Que las tableModel sean manejadas por los controladores.
      - Desarrollo de TablaCarrito.
      - Desarrollo de TablaModel<Cliente>.
      - Desarrollo de TablaModel<Producto>.
      - Objetivo cumplido.
      - Trabajando sobre Ventanas utilizando BorderLayout
```

```
- 05/12/2015
      - Aplico nuevo diseño de ventanas con BorderLayout en:
            - VentanaCliente.
            - VentanaCarrito.
            - VentanaProducto.
      - UML actualizado
      - Desarrollando nueva vista "OtraVista" utilizando BorderLayout.
            - Problema:
                  - Al agrandar la pantalla panel sur come el panel centro.
                        ¿No debiera ser al reves?
                  - Hallado verdadero causante:
                        En Swing cuando se setea un JPanelScroll en el CENTER
                        del BorderLayout este panel nunca se redimensiona.

      Para mañana:
            - Seguir trabajando sobre dicho error.
            - Agregar BorderTitled a paneles carrito, cliente y producto.
            - Agregar funcion setActionCommand(String) en InterfazVista
            - Probar vista en PCs con diferentes tamaños de pantalla
            - Arreglar formularios (descripcion aparece por debajo de precio)
            - Probar FlowLayout en botones de la vista.


- 06/12/2015:
      - Error solucionado (parcialmente)
            - Defini un setPreferredSize en la tabla que causaba el error.
            - Puede que esta solucion represente una limitación.
            - Probar en pantallas de PCs de otros tamaños.
      - Se prueba FlowLayout en algunos botones, sin éxito.
      - Agrego bordes con títulos a paneles.
      - Nueva vista testeada en pantalla 1080p, sin problemas.
      - Se agrega función setBotonActionCommand() en InterfazVista
      - Se arregla bug de formularios.
      - Terminada nueva vista ahora llamada "VistaConLayout"
      - La anterior vista se revautiza "VistaSinLayout".
      - VentanaCarrito le agrego código y descripción del producto
seleccionado.

      A futuro:
            - Implementar funcion que permita conocer el historico de ventas.
            - Investigar sobre reportes.
            - Investigar sobre Triggers.


- 07/12/2015:
      - Investigando sobre generación de reportes en Java
      - Instalando el iReports.
      - Genero un nuevo proyecto y selecciono como diseño un template
            predefinido.
      - Cargo Datasource y la query historico_ventas_2
      - Comienzo el diseño del reporte.
      - Investigando como mostrar reporte en aplicación Java
      - Trabajando en proyecto de pruebas "carrito_pruebas"
      -Se prueba ejemplo de:
```

```
          -http://community.jaspersoft.com/wiki/deploying-reports
     - Problema:
          -No encuentra el archivo en el path especificado.
     - Solución:
          - Busco path actual con System.getProperty("user.dir")
               y le agrego /src/reporte/historicoVentas.jasper
          - Atención: Funciona pero hay Warn.
     - Se prueba mismo ejemplo pero esta vez dandole mi DataSource.
     - Muestro pdf en ventana Swing.
     - Jars necesarios de JasperReports:
          - commons-beanutils-x.x.x.jar
          - commons-collections-x.x.x.jar
          - commons-digester-x.x.jar
          - commons-logging-x.x.jar
          - groovy-all-x.x.x.jar(if the report language is deafult Groovy)
          - iText-x.x.x.jar(For PDF Exports)
          - jasperreports-x.x.x.jar
          - poi-x.x.jar(for excel exports)
          - jfreechart-x.x.x.jar
     - Comienzo a trabajar directamente sobre el proyecto.
     - Implementado.
     - Prueba en otras PC.
          - Error:
               - Al solicitar reporte la aplicación se cierra.
          - Causante:
               - Jasper Reports genera path absolutos, no variables.
          - Solución:
               - Ademas del JAR ejecutable sera necesario pasar la carpeta
                 con los archivos necesarios del
                 JasperReport(.jrxml,.jasper,.img).
          - Dicha carpeta es: "C://Jasper_Reports/"
     - UML actualizado.

     A futuro:
          - Publicar la v2.0.0
          - Actualizar la documentación.
          - Probar con Branchs de git una implementación de
            "ControladorPrincipal" que abstraiga a la vista de los distintos
             controladores.
          - DetalleVenta tiene atributo venta. Revisar.
          - Investigar sobre Triggers.


- 08/12/2015:
     - Se lleva a cabo la actualización de la documentación del proyecto.
     - Se actualiza sección que describe el proyecto.
     - Especificada las nuevas tecnologías utilizadas (JasperReports, etc.)
     - Detallados los nuevos objetivos.
     - Confeccionando nuevo diagrama de casos de uso.
     - Confeccionando UML adicional con herramienta Dia.
     - Actualizada sección Duración detallando el flujo de trabajo
       llevado a cabo.
     - Transcripción de bitacora a documentación.

     A futuro:
          - pulir secciones Dudas, Opcional y Lecciones aprendidas.
```

- 09/12/2015:
    - Realizando curso online SQL en Stanford University:
        Link:
        https://lagunita.stanford.edu/courses/DB/2014/SelfPaced/about
    - Schemas:
        - ver path:
            - show search_path
        - agregar schema "myschema" al path:
            - SET search_path TO myschema,public;

- 12/12/2015:
    - Se agrega a documentación sección Lecciones aprendidas.
    - Finalizado curso de SQL.

- 13/12/2015:
    - Comienzo de curso "Constraints and Triggers"
    - Problemática:
        - Si se cambia precio del producto, cambio el precio en todas las ventas realizadas.
    - Solución:
        - Agregar atributo "precioUnitario" a cada producto.
    - Objetivo:
        - Agregar atributo "precioUnitario" a cada producto.
    - Se crea una nueva branch en git llamada "atributo_precio_venta"
    - En BD se agrega dicho atributo en tabla detalle_venta con el nombre de "precio_unitario".
    - Para las ventas que ya existían en las tablas se les asigna el precio actual del producto con la siguiente query:
        - UPDATE detalleventa d SET precio_unitario=p.precio FROM detalleventa dv,producto p WHERE d.codigo_producto=p.codigo_producto AND d.codigo_producto in (SELECT pp.codigo_producto FROM producto pp)
    - A atributo de tabla se le agregan la constraint NOT NULL y se le asigna el mismo dominio que posee el atributo precio de tabla producto, esto es: numeric(10,2)
    - Actualizadas vistas y procedures afectadas por el cambio y se eliminan otras que ya no son necesarias.
    - Se agrega nuevo atributo precioUnitario a clase DetalleVenta junto con sus getters y setters.
    - Actualizado método confirmarAgregarCarrito de ControladorCarrito.
    - Actualizada query del JasperReport.
    - Objetivo Cumplido.

    A futuro:
    - Chequear utilidad de mantener Producto p en detalleVenta
    - Cuando cambio el precio del producto no cambia el del carrito.
    - Investigar sobre otros diagramas UML que no han sido agregados aun a la documentación como diagrama de secuencia.

- 14/12/2015:
    - Leyendo "Ingenieria de software" de Sommerville,capítulo 5 sobre diagramas UML.

```
     - Trabajando sobre diagramas de secuencia de casos de uso especificados
       que son:
       - Insertar Producto.
       - Actualizar Producto.
       - Agregar Cliente.
       - Agregar Item.
       - Quitar Item.
       - Limpiar Carrito.
       - Finalizar Compra.
     - Diagramando caso de uso Insertar Producto.
     - Diagramando caso de uso Actualizar Producto.
     - Diagramando caso de uso Insertar Cliente.
     - Diagramando caso de uso Agregar Item.
     - Diagramando caso de uso Borrar Item.
     - Diagramando caso de uso Limpiar Carrito.
     - Diagramando caso de uso Finalizar Compra.
     - Diagramando caso de uso Historico de Ventas.
     - Diagramas agregados a la documentación.


- 15/12/2015:
     - Publicado Carrito de Compras v2.0.0 con las siguientes mejoras:

       - Una nueva interfaz gráfica con Swing utilizando BorderLayout llamada
         "vistaConLayout", la anterior ha sido renombrada a "vistaSinLayout".
       - Se ha añadido una nueva funcionalidad para la solicitud de reportes a
         traves JasperReports. El usuario podrá acceder a un histórico de las
         ventas realizadas por su organización.
       - Nuevo controlador controladorReporte para el manejo de reportes como
         el anteriormente descripto.
       - Se agrega un nuevo atributo precioUnitario a detalleVenta con el valor
         en el que fue vendido el producto el cual no varía cuando se actualiza
         el precio de venta corriente del mismo producto.
       - Las abstractTableModel son ahora administradas por los controladores
         correspondientes.

     - Se organiza el material a entregar al profesor Javier Blanqué en el
       final de programación IV.
     - Se testea aplicación en Debian 7.
       - Hubo que actualizar el jdk 1.6 a jdk1.7
       - No pudo testearse la BD ya que no pudo instalarse PostgreSQL con
         éxito.
     - Se agrega a la entrega del proyecto un archivo léeme.txt que detalla los
       pasos a seguir para la instalación de la aplicación.


- 16/12/2015:
     - Se realizan testeos en Ubuntu 11.
       - Se debió actualizar jdk a jdk1.7
       - Todo en perfectas condiciones.
     - Se realizan testeos en Debian 7.
       - Se repara BD.
       - Todo funciona con normalidad.
     - Se completa el léeme.txt con la descripción de lo entregado e
       instrucciones detalladas de su instalación.
```

A FUTURO:

- Realizar tablas de Producto, Cliente, Venta y detalleVenta. HECHO.
- Realizar clases JAVA correspondientes a cada tabla. HECHO.
- Procedimientos y vistas mínimos para manejos de tablas desde Java. HECHO.
- Clases para crear conexión desde aplicacion JAVA a BD. PARCIAL
  Problemática: ¿Qué devuelven métodos? Si no se puede generar conexión ¿qué pasa?.
- Clases Producto, Cliente, Venta y detalleVenta para realizar inserciones y demas en BD. HECHO
- Interfaz gráfica. HECHO!!
- Reflection/annotations/generic en tablas. HECHO.
- Utilizar patrón MVC. HECHO
- Agregar reportes. HECHO.
- Implementar triggers en BD.

OPCIONAL:

- Poder conectarme a BD desde otros equipos externos al servidor de BD. HECHO.
- Historial de ventas. HECHO
- Carrito para cada cliente.
- Lista favoritos por cliente.
- log de usuarios.
- Agregar más atributos a tablas
- Manejo de Stock.
- Uso de Triggers.
- Procedimientos y vistas extras.
- Clase y tabla Descuento.
- Agregar imágenes a producto.
- Migrar a JPA

DUDAS:

- Funciones de clases que se conectan a la BD devuelven booleano según si la comunicación fue correcta. El problema es que dicho booleano es devuelto según variables que podrían no haber sido bien interpretadas.
- Controlador y modelo podrían depender de cada uno ya que controlador instancia directamente clases que pertenecen al modelo.
- Las clases que implementan AbstractTableModel son consideradas en el presente proyecto como parte de la vista.
- Cuando se genera un jar ejecutable JasperReports no funciona ya que requiere que los documentos de los cuales hace uso (.jrxml,.jasper,.img) se encuentren en algún directorio especificado. Por esta causa y como decisión de implementación cuando se transfiera el jar ejecutable al usuario también se le pedirá que agregue una carpeta con dichos documentos a un path también especificado.

LECCIONES APRENDIDAS:

- Stored procedures y herramienta PostgreSQL en general.
- Desarrollo de conexiones a bases de datos a traves de API JDBC.
- Manejo de errores a partir de try-catch.
- Implementación de Interfaz gráfica a partir de Swing con herramienta WindowBuilder.
- Manejo de repositorios git y control de versiones.
- Diagramación de UMLs mediante ObjectAid.
- Diagramación de Casos de uso.
- Conexion a BD a traves de dispositivos externos a esta dentro de una red local.
- Uso de annotations y clases genericas para desarrollar TableModel
- Desarrollo siguiendo el patron de arquitectura MVC
- Uso de Layouts en Swing.
- Generación de reportes con JasperReports