Tecnicatura Superior en Análisis, Desarrollo y Programación de Aplicaciones

Programación Orientada a Objetos

Trabajo Práctico "Sistema de Facturación"
Versión 1.2 Mayo 2015

Lic. Guillermo R. Cherencio

Índice

Definición del problema

Se pretende la implementación de un sistema de facturación muy simple, el cual esta formado por las siguientes entidades:

cliente(**idc**,descr,saldo)
producto(**idp**,descr,precio_base,stock)
factura(**nro**,fecha,idc,estado,monto,pagada)
idc foreign key cliente
detalle(**nro,idp**,cantidad,precio,subtotal)
nro foreign key factura
idp foreign key producto

donde factura.estado puede ser 0 (factura iniciada, aun no impresa y confirmada) o 1 (factura impresa, terminada, confirmada y, por lo tanto, no modificable); factura.pagada es 0 (no pagada) o 1 (pagada). Cuando se confirma una factura, se suma factura.monto a cliente.saldo y se resta de producto.stock la cantidad facturada de cada producto. Cuando se paga una factura, se resta factura.monto de cliente.saldo. El atributo detalle.subtotal es igual a cantidad * precio; factura.monto es igual a la sumatoria de detalle.subtotal.

Casos de Uso

Los casos de uso de la aplicación son los siguientes:

-Alta de un nuevo cliente (saldo deudor = 0)
-Baja de un cliente existente
-Cambio del nombre de un cliente existente
-Sumar/Restar saldo a un cliente
-Alta de un nuevo producto (indicando su stock inicial)
-Baja de un producto existente
-Cambio del nombre o precio base de un producto existente
-Sumar/Restar stock de un producto
-Creación de una nueva factura
-Finalización o confirmación de una factura previamente hecha
-Pago de una factura previamente hecha

El sistema deberá utilizar una interfaz gráfica Swing y un menú de opciones para los casos de uso.

Los datos serán persistidos en una base de datos Firebird 2.5 de forma muy

simple, acorde con las entidades del sistema, sin implementar ninguna lógica en el manejo de los datos[1].

En el arranque de la aplicación, el usuario proveerá todos los datos para establecer la conexión con la base de datos Firebird: servidor, puerto tcp/ip, base de datos, usuario, contraseña, rol. La contraseña no se guardará en ningún lugar, no obstante, para facilitar el ingreso de estos datos, el resto de los datos podría almacenarse y recuperarse en cada arranque de la aplicación.

Diseño del Sistema

Discusión de opciones de diseño, acorde con http://www.grch.com.ar/docs/poo/Apuntes/Relaciones.pps y con http://www.grch.com.ar/docs/poo/Apuntes/Desarrollo.orientado.objetos.uml.pdf

Implementación del Sistema

Acorde con http://www.grch.com.ar/docs/poo/Apuntes/Desarrollo.orientado.objetos.uml.pdf la implementación será iterativa y evolutiva. Cabe destacar que esta implementación pueda diferir de un caso real porque se esta utilizando con fines de aprendizaje para que, de la misma forma que la aplicación va a ir evolucionando, también se espera una evolución del alumno con respecto a la programación orientada a objetos y el lenguaje de programación Java.

1. Diseño de clases entidades

Requisitos

- Manejo de arreglos y arreglos de referencias en clase Factura. (ch07)
- Manejo de fechas en Java.
- Manejo de comentarios javadoc. (ch03)
- Comparación en profundidad vs comparación superficial
- Sobre-escritura de métodos de la clase Object, contrato entre equals() y hashCode().

1 Se refiere a que la base de datos no tendrá ningún tipo de programación de triggers o stored procedures; esta tarea será realizada en año próximo en Práctica Profesional. Por ahora, la lógica de los datos será delegada a la aplicación.

- Box / Unbox y clases wrappers
- Manejo de la clase String
- Manejo básico de excepciones, ejemplo: ParseException para implementar método setFecha(String) en clase Factura

Cliente
-idc:int -descr:String -saldo:double
+Cliente() +Cliente(int) +Cliente(int,String) +Cliente(int,String,double)
+getIdc():int +getDescr():String +getSaldo():double +setIdc(int) +setDescr(String) +addSaldo(double)

Producto
-idp:int -descr:String -precio:double -stock:int
+Producto() +Producto(int) +Producto(int,String) +Producto(int,String,double) +Producto(int,String,double,int)
+getIdp():int +getDescr():String +getPrecio():double +getStock():int +setIdp(int) +setDescr(String) +setPrecio(double)

+addStock(int)

Detalle
-producto:Producto -cantidad:int -precio:double
+Detalle() +Detalle(Producto) +Detalle(Producto,int) +Detalle(Producto,int,double)
+getProducto():Producto +getCantidad():int +getPrecio():double +getSubtotal():double +setProducto(Producto) +setCantidad(int) +setPrecio(double)

El método getSubtotal() devolverá cantidad * precio.

Factura
-nro:int -fecha:Date -cliente:Cliente -confirmada:boolean -monto:double -pagada:boolean -detalle[20]:Detalle -cnt:int //contador para arreglo de detalle
+Factura() +Factura(int) +Factura(int,Date) +Factura(int,Date,Cliente)
+getNro():int +getFecha():Date +getFechaStr():String // fecha en formato dd/mm/yyyy +getCliente():Cliente +getCount():int

+isConfirmada():boolean
+isPagada():boolean
+getMonto():double
+setNro(int)
+setFecha(Date)
+setFecha(String)
+setCliente(Cliente)
+confirmar()
+pagar()
-calcular() // realiza el calculo de monto, acorde con el Detalle actual
+addDetalle(Detalle)
+addDetalle(Detalle,int) // agrega detalle en la posición indicada
+delDetalle(Detalle)
+delDetalle(int) // borra detalle en la posición indicada

El método setFecha() es sobrecargado y el String que recibe se supone que es una fecha con formato “dd/mm/yyyy”. En caso de que la fecha sea incorrecta, el atributo fecha deberá quedar como null y enviar mensaje de error a stderr.

El método getCount() devuelve la cantidad de objetos Detalle asociados con esta factura.

El método confirmar() es para confirmar la factura y ello implica recalcular el monto de la factura, sumar al saldo deudor del cliente y descontar el stock de los productos asociados a la factura. Se supone que para ejecutar este método, se debe contar con un cliente asociado y tener 1 o más objetos Detalle asociados. Luego de que una factura es confirmada no se puede modificar, es decir, no se pueden agregar ni quitar ni modificar objetos Detalle a la misma, ni la fecha, ni el cliente, etc.

El método pagar() puede ejecutarse si la factura esta previamente confirmada. No se puede pagar dos o más veces una factura. Tampoco es posible hacer un pago parcial de la factura. Una vez pagada la factura, se debe descontar del saldo deudor del cliente de la factura.

1.1 Sobre-escritura de métodos de la clase Object

Los métodos delDetalle() deberán implementarse realizando una comparación en profundidad para determinar si el objeto Detalle esta asociado a la Factura (borrar todas las ocurrencias de igualdad de objeto que se encuentren). Borrar todos los objetos en donde la comparación en profundidad sea verdadera utilizando equals(). Sobre-escribir los métodos equals() y hashCode() en todas las clases del sistema. El método hashCode() de Cliente debe retornar el atributo idc; en Producto es idp; en Detalle es un entero formado por la combinación del id del producto más la cantidad; en Factura es el atributo nro.

Se debe sobre-escribir el método toString() de la clase Object en todas las clases, la representación String de cada objeto será "[<prop>:<valor>,<prop>:<valor>,....]" ejemplo para un objeto de clase Producto: [idp:1,descr:motosierra,precio:456.78,stock:100] .

2. Prueba de clases entidades

Crear clase App, implementar metodo main(), crear objetos de clases entidades y probar el correcto funcionamiento de todos los métodos.

Requisitos

- Manejo de arreglos y arreglos de referencias en clase Factura. (ch07)
- Manejo de clase Date en Factura.
- Manejo de comentarios javadoc.(ch03)

3. Creación de clase estática Consola

Clase Consola posee sólo métodos estáticos para el manejo de la consola/terminal, ingreso de datos, validaciones, salidas por pantalla, etc.

Requisitos

- Manejo de clases Wrappers
- Boxing / UnBoxing
  https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html
- Manejo de excepciones derivadas de Exception y RuntimeException (ch12)
- Manejo de import (ch09)
- Package java.io (ch13)
- Manejo de streams, en especial InputStream, BufferedReader, InputStreamReader, BufferedInputStream, System.in (ch06, ch13)
- Overloading (ch05)

Consola
+static print(Object) +static println(Object) +static readLine():String +static readLine(String):String

```
+static readLine(String,String[ ]):String  // arreglo de valores posibles a ingresar
+static readInt():Integer
+static readInt(String):Integer
+static readDouble():Double
+static readDouble(String):Double
```

Los argumentos String que reciben estos métodos se refiere al prompt que espera ver el usuario durante el ingreso de los datos, ejemplo:

int codigo = Consola.readInt("Ingrese Codigo:")

en la pantalla del usuario seria:

Ingrese Codigo: 1234

el valor 1234 quedaría asignado a la variable codigo.

Los métodos readInt(), readDouble(), etc. devolverán objetos wrappers que serán null en caso de que ocurra una IOException mientras se ingresan por teclado.

Los métodos print() y println() reciben argumentos de tipo Object para permitir imprimir objetos de la misma forma que podemos hacer con System.out.println(), obsérvese que este método de la clase PrintStream esta sobrecargado.

4. Modificación de la clase App

Con el objetivo de realizar pruebas interactivas y hacer reuso de la clase Consola.

Requisitos

- Conocimiento, uso y aplicación de clase Consola

4.1 Refinamiento de entidades

Eliminación de arreglo en clase Factura, uso de colección de objetos Detalle. Pruebas, cambio de interfase de Factura, etc.

Requisitos

- Conocimiento, uso y aplicación de clase Consola
- Conocimiento del framework Collection (ch14)

5. Creación de la clase AppView, implementación GUI.

Requisitos

- Manejo del framework Swing (componentes, clases, métodos, etc.) (ch10) https://docs.oracle.com/javase/tutorial/uiswing/index.html
- Manejo de interfases (ch08)
- Clases anónimas https://docs.oracle.com/javase/tutorial/java/javaOO/anonymousclasses.html
- Patrón de diseño MVC (ch10)
- Estudio especial de la clase JTable, AbstractTableModel https://docs.oracle.com/javase/tutorial/uiswing/components/table.html
- Manejo de Layout Managers (ch10)

6. Creación de ventana de login

Creación de clase Login que implementa interfaz gráfica para conexión a base de datos, acorde con los requisitos de la aplicación y que ésta sea la primer ventana de la aplicación.

Requisitos

- Manejo del framework Swing (componentes, clases, métodos, etc.) https://docs.oracle.com/javase/tutorial/uiswing/components/table.html
- Manejo de interfases (ch08)
- Clases anónimas https://docs.oracle.com/javase/tutorial/java/javaOO/anonymousclasses.html
- Patrón de diseño MVC (ch10)
- Estudio especial de la clase JTable, AbstractTableModel https://docs.oracle.com/javase/tutorial/uiswing/components/table.html
- Manejo de Layout Managers

7. Incorporación de Driver JDBC Jaybird

Descarga, configuración e incorporación de driver JDBC Jaybird para conectar a Firebird 2.5 desde Java.

Requisitos

- Acceso a portal http://www.firebirdsql.org
- Conocimiento del driver JDBC Jaybird, programas de ejemplo https://docs.oracle.com/javase/tutorial/jdbc/index.html
- FlameRobin http://www.flamerobin.org
- Firebird 2.5

8. Implementación de clase estática DataIO

La clase DataIO facilitará la interacción con Firebird 2.5 utilizando el driver JDBC Jaybird.

Requisitos

- JDBC https://docs.oracle.com/javase/tutorial/jdbc/index.html
- Conocimiento del driver Jaybird, programas de ejemplo https://docs.oracle.com/javase/tutorial/jdbc/index.html
- FlameRobin http://www.flamerobin.org
- Firebird 2.5 http://www.firebirdsql.org
- Reuso de clase DataIO provista por docente a cargo, usada en otros proyectos

9. Creación de base de datos

Creación de base de datos Firebird 2.5 para soportar los datos de la aplicación y carga de datos de prueba.

Requisitos

- Instalación Firebird 2.5 http://www.grch.com.ar/docs/bd/tutorial/firebird/instalacion.firebird.2.5.pdf
- SQL (DML, DDL)
- FlameRobin http://www.flamerobin.org

10. Modificación de la clase Login

Modificación de la clase Login para reusar clase DataIO y permitir interacción con la base de datos de la aplicación.

Requisitos

- JDBC
  https://docs.oracle.com/javase/tutorial/jdbc/index.html
- Conocimiento del driver Jaybird, programas de ejemplo
- FlameRobin http://www.flamerobin.org
- Firebird 2.5 http://www.firebirdsql.org

11. Modificación de la clase AppView

Modificación de clase AppView para implementar la persistencia de datos en la base de datos Firebird 2.5 de la aplicación.

Requisitos

- JDBC
  https://docs.oracle.com/javase/tutorial/jdbc/index.html
- FlameRobin http://www.flamerobin.org
- Firebird 2.5 http://www.firebirdsql.org
- SQL

12. Ampliación de clase AppView

Modificación de AppView para implementar cada uno de los casos de uso, acorde con los requisitos de la aplicación. Creación de nuevas ventanas y opciones de menú. Pruebas individuales de cada nueva ventana. N iteraciones evolutivas hasta lograr la completitud de la aplicación.

Requisitos

- JDBC
  https://docs.oracle.com/javase/tutorial/jdbc/index.html
- FlameRobin http://www.flamerobin.org
- Firebird 2.5 http://www.firebirdsql.org
- SQL

13. Refinamiento del código, separación de clases.

Separación en distintos paquetes: view, model, entity, etc. Recompilación. Pruebas.

Requisitos

- Conocimiento sobre packaging, jar, distribution and deployment
- Modificadores de acceso vs package
- Instrucciones package e import
- BlueJ

14. Empaquetamiento de la aplicación

Creación de archivo .jar, scripts, etc. Pruebas desde versión compilada, desde linea de comandos/terminal/icono/etc. Desarrollo de instalador. Generación de documentación. Distribución. Pruebas.

Requisitos

- Conocimiento sobre packaging, jar, distribution and deployment (ch09)
- Uso de jar, javadoc (ch03)
- BlueJ http://www.bluej.org
- Conocimiento de scripting del sistema operativo anfitrión

Habiendo llegado a esta instancia, espero que haya disfrutado el viaje y comprendido cómo fue evolucionando nuestra aplicación y nuestro conocimiento sobre la programación orientada a objetos y el lenguaje Java.

Atte. Guillermo Cherencio
Programación Orientada a Objetos
ISFT N° 189