Universidad Nacional de Luján
Departamento de Ciencias Básicas
Programación III

Introducción al uso de JDBC, Swing y Firebird 2.5 en Java 6 Standard Edition

Guillermo R. Cherencio
gcherencio@unlu.edu.ar

"Implementación del patrón de diseño Model View Controller Pattern"

Objetivo:

Implementaciónde patrón de diseño movel-view-controller pattern a partir de interfaz gráfica swing que interactúa con base de datos firebird 2.5.

Requisitos:

Conocimientos básicos de java 1.6 o superior standard edition.

Contar con el SGBD Firebird SQL 2.5.

Contar con driver JDBC para Firebird SQL JayBird 2.1.6 descargado de http://www.firebirdsql.org/en/jdbc-driver/ descomprimir archivo .zip (1)

Crear en servidor local la base de datos test y dentro de la misma crear la tabla test(id,descr) en donde id es dato de tipo integer, primary key y descr es un dato de tipo varchar(50) not null.

Contar con JDK (java 2 development kit, se (standard edition) 1.6 o superior) instalado http://www.oracle.com/technetwork/java/javase/downloads/index.html (2)

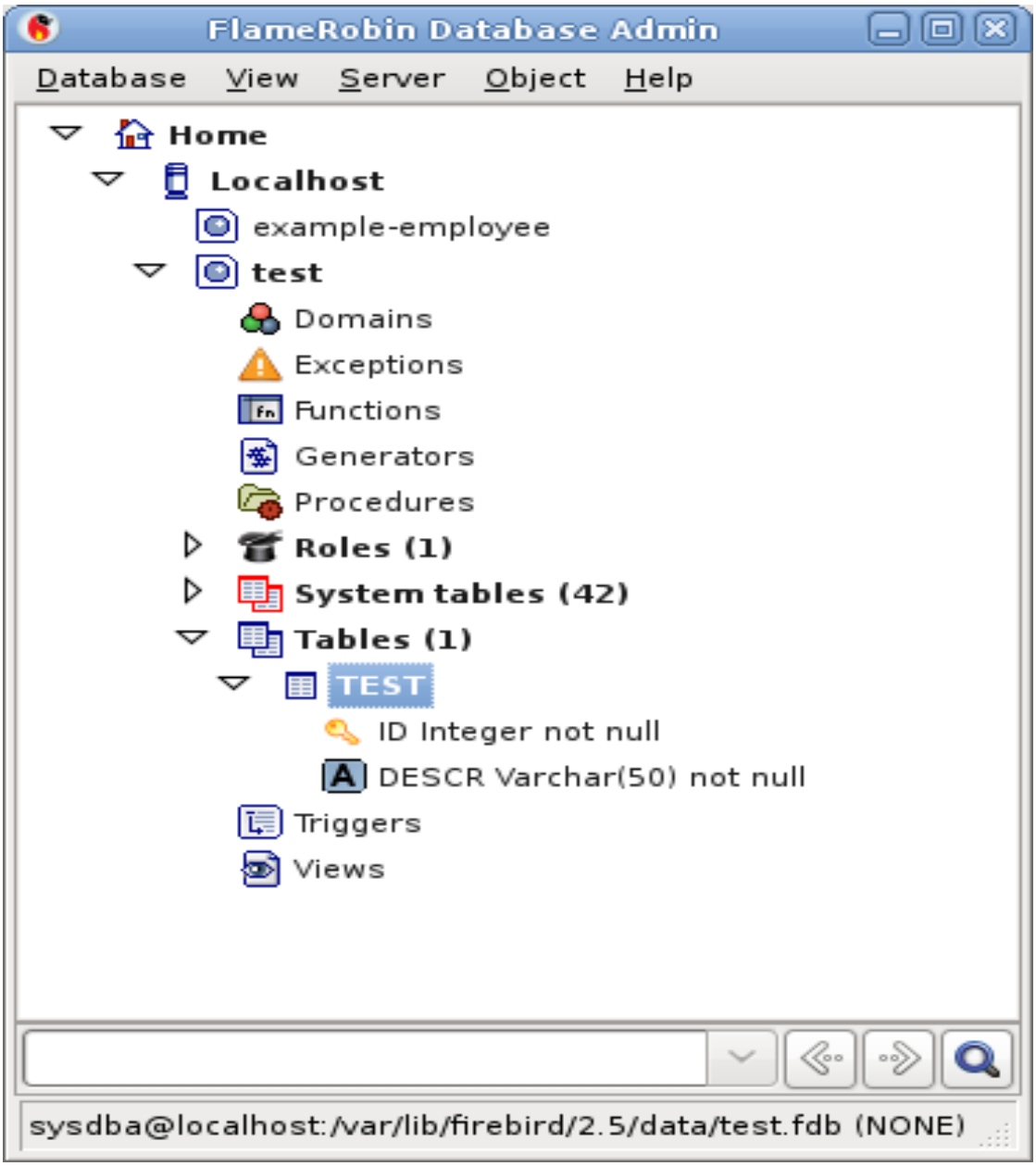
Contar con IDE BlueJ instalado y configurado para el JDK indicado en (2), agregar en BlueJ en opción Tools / Preferences, tab Libraries, agregar librería JDBC jaybird-full-<version>.jar descargado en (1).

Puede obtener driver jaybird y versión empaquetada (.jar) del proyecto BlueJ vinculado a este documento, junto con toda su documentación javadoc desde http://www.grch.com.ar/docs/unlu.poo/java.jdbc/ .Puede descomprimir el archivo chivilcoy-<version>.jar y abrir proyecto utilizando BlueJ. Puede ver toda la documentación a partir de la carpeta doc y leer el archivo readme.txt.

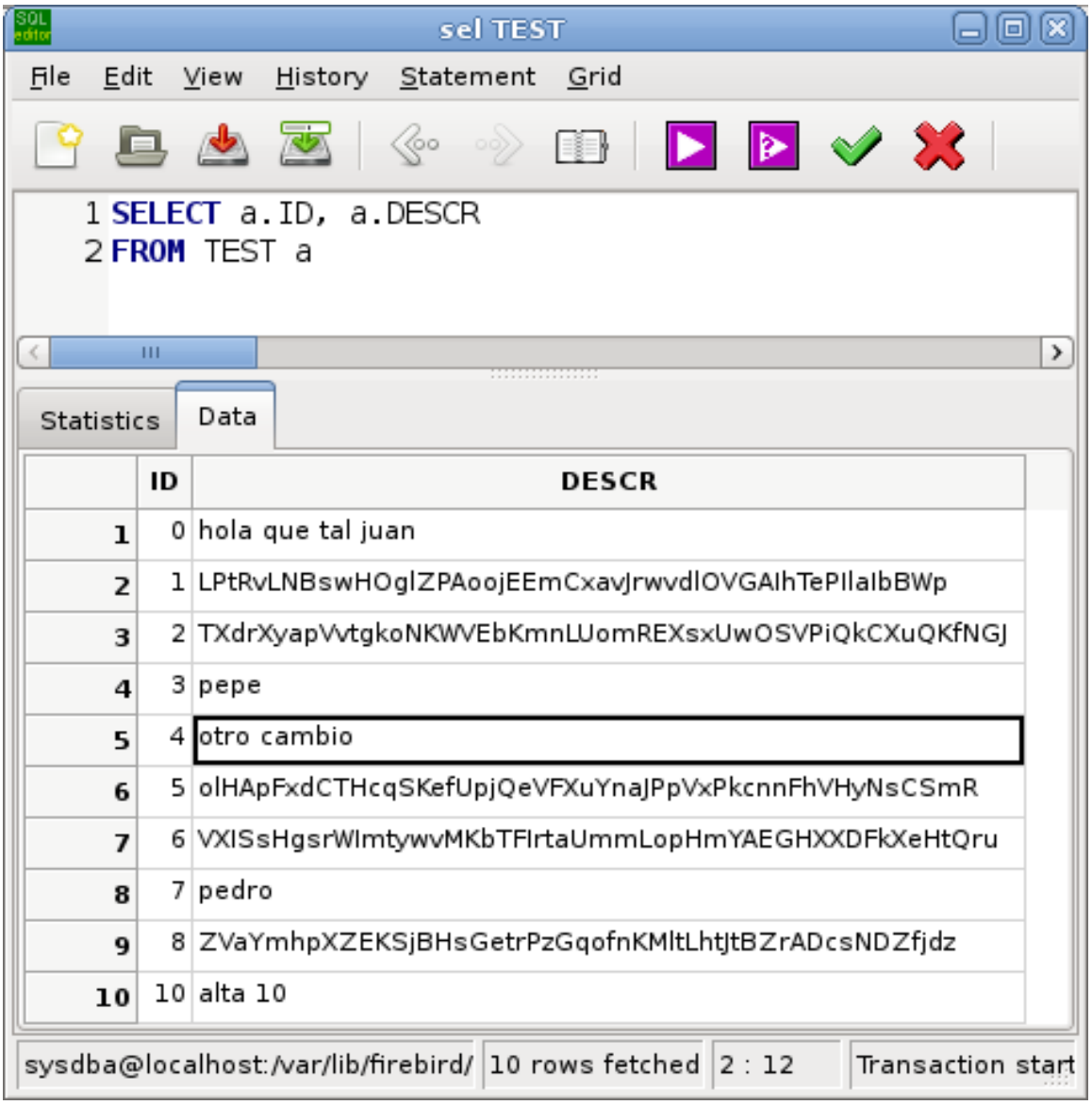
Introducción

Se pretende implementar una grilla que despliegue los datos de una consulta sql y permita realizar altas, bajas y modificaciones sobre la misma. Se propone una interfaz gráfica formada por una ventana principal (Jframe) la cual en el centro posee un JscrollPane (scroll horizontal y vertical) dentro del cual se encuentra una Jtable (grilla), al pie de la ventana una serie de botones (Jbutton) para Agregar, Cambiar y Borrar filas de la grilla:

para comenzar debemos crear la base de datos sobre el servidor firebird local previamente instalado, para ello podemos utilizar flamerobin o bien isql-fb:

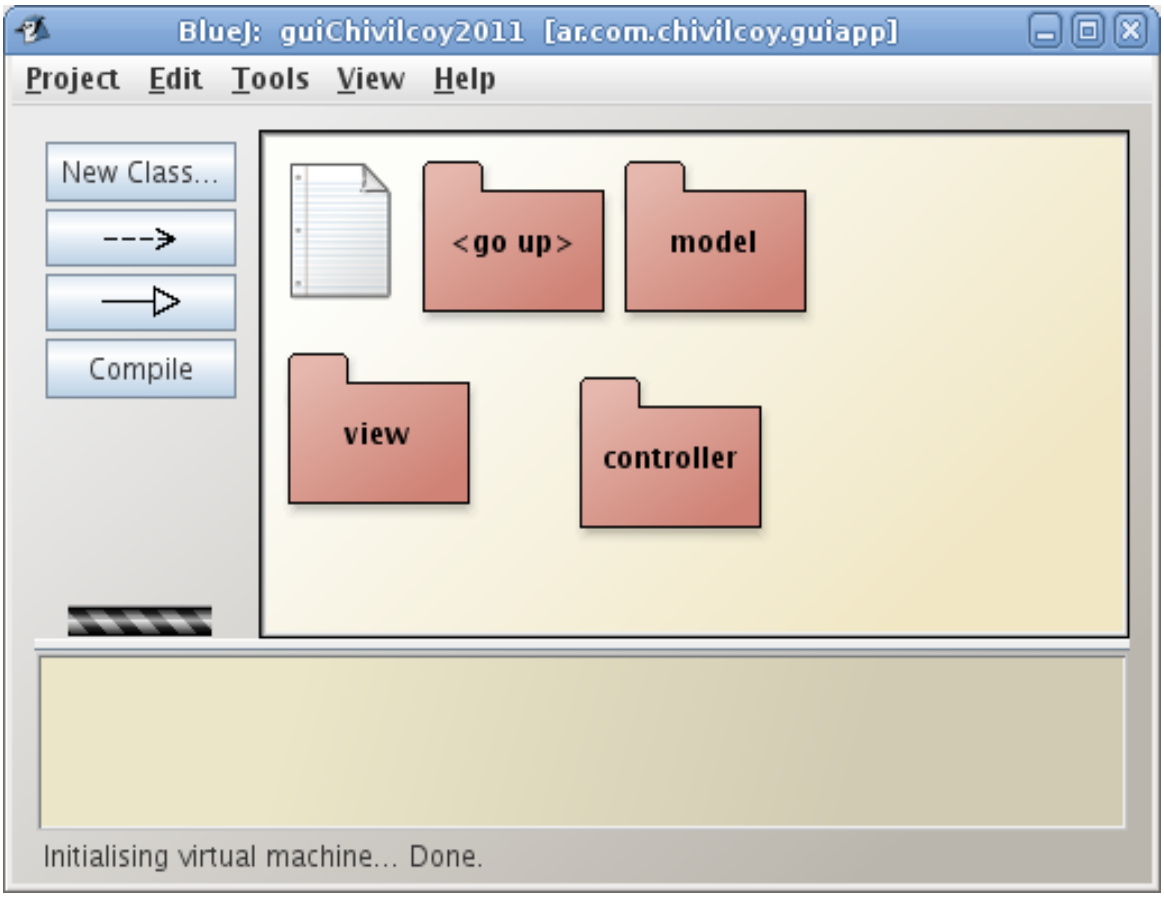


una vez creada la tabla, teniéndola seleccionada en flamerobin, podemos utilizar la opción Database / Advanced / Test Data Generator … para generar registros dentro de la tabla recién creada y luego consultar dichos datos:



Creación de proyecto y organización de clases

Las clases java deben organizarse de forma tal que sus nombres sean unívocos al momento de ser distribuidas al cliente, una forma de lograrlo podría ser a partir de un nombre de dominio, supongamos que contamos el dominio www.chivilcoy.com.ar y que nuestra aplicación se llamará “guiapp”, a su vez, esta aplicación que implementa el patrón de diseño mvc podría dividirse en “model”, “view”, “controller” para agrupar allí dentro las clases que implementan el patrón de diseño. En java esto puede lograrse a través de la agrupación de clases en packages (paquetes), BlueJ nos dá soporte para la creación de packages: crear un nuevo proyecto y dentro del mismo utilizar la opción Edit / New Package … y tipear el nombre completo o parcial de un package. Hay una cierta analogía entre package y directorio, crear el package ar.com.chivilcoy implica crear la carpeta ar, ar/com y ar/com/chivilcoy. Puedo ir creando progresivamente el package a partir de “ar” luego “com” luego “chivilcoy” utilizando BlueJ, resultado el proyecto en algo como esto:

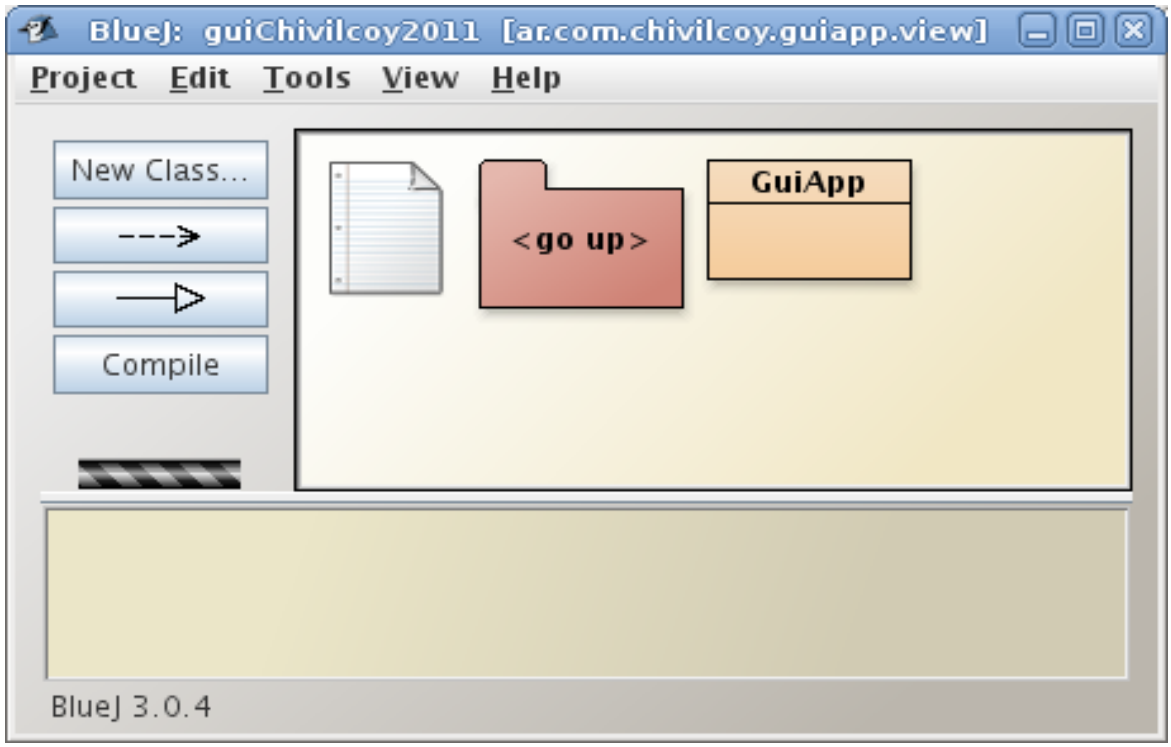


esta imagen muestra el proyecto BlueJ guiChivilcoy2011, en este caso, mostrando el contenido del package ar.com.chivilcoy.guiapp dentro del cual tenemos otras carpetas en donde guardar las clases del proyecto **según su funcionalidad**:

Clase	**Package**	**Funcionalidad**
GuiApp	ar.com.chivilcoy.guiapp.view	Clase view. Punto inicial de ejecución de la aplicación y creación de interfaz gráfica. Permite vincularse -al menos- con la clase controller.
GuiController	ar.com.chivilcoy.guiapp.controller	Clase controller. Permite implementar los eventos Action (click) sobre los botones de la view (agregar, cambiar, borrar), utiliza los servicios de la clase model para la manipulación de los datos ante la ocurrencia de los eventos.
Entidad	ar.com.chivilcoy.guiapp.model	Interfase que identifica a todo objeto que debe ser persistido en una base de datos. Obliga a la implementación de una serie de métodos que serán invocados por otros objetos para

		poder persistir un objeto de este tipo en la b.d.
DataIO	ar.com.chivilcoy.guiapp.model	Clase que contiene métodos estáticos (static, de clase), de la cual no se pretende crear objetos (al menos en esta versión del sistema) y que maneja todas las operaciones de I/O sobre la base de datos utilizando JDBC para leer y grabar de la misma. Provee servicios a otras clases del sistema. GuiApp invoca su método init() para conectar con la base de datos y al salir invoca al método finish() para terminar la conexión.
Test	ar.com.chivilcoy.guiapp.model	Clase que implementa la interfaz Entidad y por lo tanto, los objetos Test representan tuplas a persistir dentro de la tabla test.
GuiTableModel	ar.com.chivilcoy.guiapp.model	Clase Model (de datos) de la cual se alimenta la JTable visualizada en GuiApp. Mantiene en memoria una lista de objetos de tipo Entidad (en la cual almacenaremos todo tipo de objeto que implemente dicha interfase). Clase que deriva de AbstractTableModel.

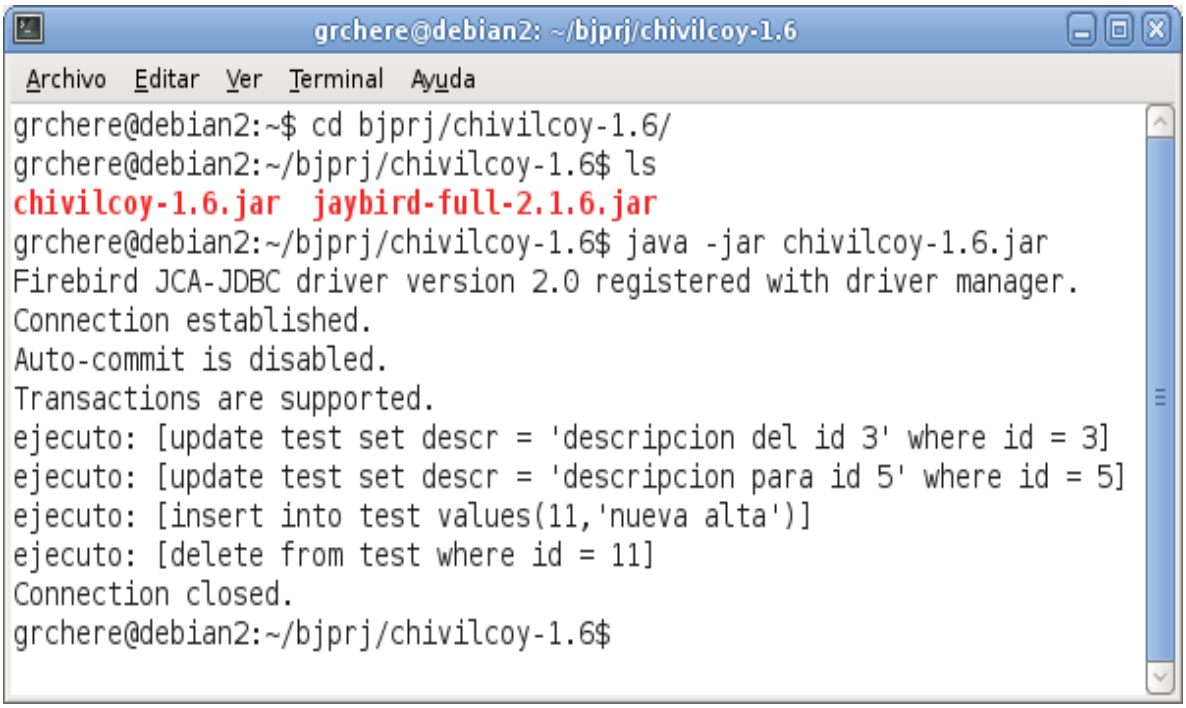
Generalmente comenzamos por la creación de la vista (la interfaz gráfica) de la aplicación, contenida dentro del package ar.com.chivilcoy.guiapp.view, la clase GuiApp la cual será responsable de la visualización:



esta clase creará un objeto de tipo GuiController al cual le pasará su referencia para que el objeto GuiController pueda acceder a los botones y métodos que exhibe -hacia el exterior- la clase GuiApp. La clase GuiController creará un objeto de tipo GuiTableModel y le indicará a ésta un objeto de tipo Entidad (Test). GuiTableModel cargará desde la base de datos, ejecutando un query de consulta indicado por el método getQuerySql(); por cada tupla recuperada de la base de datos se creará un objeto de tipo Entidad y será cargado en una lista que será manejada por GuiTableModel. Cada vez que se pretende cambiar un dato en una celda de la JTable, se ubicará al objeto de tipo Entidad dentro de la lista que mantiene GuiTableModel y se modificará dicho objeto. Cuando se presione el botón “Cambiar”, el objeto GuiController recorrerá todos los objetos de la lista y todos aquellos que hayan sido modificados los grabará en la b.d. . Cuando se presione el boton “Borrar”, se eliminará de la lista y de la b.d. El objeto de tipo Entidad seleccionado en la JTable. Cuando se presione el boton “Agregar” se ejecutará el método onPreInsert() de la interfase Entidad para que éste devuelva un nuevo objeto de tipo Entidad que será agregado a lista de GuiTableModel e insertado en b.d.

Como extender este diseño

Antes que nada, lo invito a que ejecute este proyecto en su computador y luego siga atentamente cada una de las lineas de código de estas clases para comprender cómo funciona lo descripto. La idea detrás de este diseño es poder crear nuevas clases que implementen la interfase Entidad (de la misma forma que lo hizo Test) y luego crear otros componentes swing para permitir altas, bajas y cambios sobre estos objetos en la base de datos test. Ud. puede modificar el driver jdbc a utilizar, el usuario, contraseña, server y base de datos a utilizar desde el interior de la clase DataIO la cual encapsula esta funcionalidad. Podemos ver en la siguente pantalla un ejemplo de ejecución de la aplicación desde la la linea de comando:



Atte. Guillermo Cherencio.
UNLu Programación III
UNLu Base de Datos.