UNIVERSIDAD AUTÓNOMA METROPOLITANA

LIC. INGENIERIA EN COMPUTACIÓN

MODALIDAD: PROYECTO TECNOLÓGICO

INFORME FINAL DE PROYECTO

PROGRAMACIÓN DE UN ADMINISTRADOR DE LLAMADAS TELEFÓNICAS VÍA MODEM

INGENIERÍA EN COMPUTACIÓN
CIENCIAS BÁSICAS E INGENIERÍA

TRIMESTRE 14-I

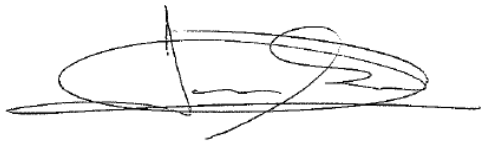
ABRIL 2014

ELABORADO POR:
JOSÉ ROBERTO ARROYO GÓMEZ.
MATRICULA: 204200272

ASESORADO POR:

OSCAR HERRERA ALCÁNTARA

Yo, OSCAR HERRERA ALCÁNTARA, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Yo, JOSÉ ROBERTO ARROYO GÓMEZ, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

ÍNDICE

1. RESUMEN

El propósito del presente proyecto es la implementación de un sistema administrador de llamadas telefónicas mediante la lectura de un módem. El sistema es capaz de realizar tareas como: detección de llamadas entrantes con notificación visual y auditiva, registro de las llamadas en bitácora, consulta de las llamadas más frecuentes, y generación de reportes de llamadas por fecha.

Durante el desarrollo del proyecto se revisaron los siguientes puntos:

- Se analizó el funcionamiento de un módem y sus comandos de programación.

En éste punto se investigaron las características de un módem con funciones de caller ID, es decir, el módem necesitaba tener la función de identificador de llamada y ser programable mediante un lenguaje de programación.

- Se utilizó un lenguaje de programación para acceder al módem.

Java es un lenguaje de programación de propósito general orientado a objetos desarrollado por Sun Microsystems (Oracle). Es una tecnología que no sólo se reduce al lenguaje sino que además provee de una máquina virtual Java que permite ejecutar código compilado Java, sea cual sea la plataforma que exista por debajo. El apoyo a esta tecnología viene dado por la gran cantidad de fabricantes que apoyan esta especificación de máquina virtual.

- Se diseñó una base de datos que incluye números telefónicos e información adicional con la finalidad de dar notificaciones visuales y auditivas en la pantalla y el sistema de audio de la computadora que hospeda al módem.

Las necesidades de almacenamiento de los sistemas de información, no pueden ser atendidas utilizando únicamente el sistema de archivos. Esto hace necesario el uso de un gestor de bases de datos, que brinde las facilidades de almacenamiento, recuperación, búsqueda optimizada, integridad y seguridad, que son necesarios para implementar el almacén de datos. El sistema gestor de bases de datos que será utilizado en este proyecto es MySQL. Este gestor cuenta con una variedad de herramientas de administración y programación que facilitan el diseño del almacén de datos, que se distribuye con licencia GLP y que facilita el desarrollo de

aplicaciones “stand alone” como el que se propone aquí, y de aplicaciones Web en donde podría revisarse la bitácora.

1.1 INTRODUCCIÓN.

En la actualidad las empresas telefónicas ofrecen cada vez un mayor número de servicios a los usuarios, sin embargo aún se omiten algunos. Los servicios que se ofrecen no cubren todas las necesidades de los usuarios, un ejemplo es cuando se desea dejar un mensaje a una persona específica en una contestadora telefónica personalizada y que no cuenta con el tiempo necesario para esperar la llamada. Otro ejemplo es el registro de la bitácora de llamadas telefónicas entrantes ya que las compañías telefónicas suelen dar el reporte de las llamadas salientes y por confidencialidad no se ofrece el registro de llamadas entrantes.

El presente proyecto pretende colaborar con los servicios telefónicos mediante un programa de cómputo que lee datos de un módem conectado a una línea telefónica de casa habitación o empresarial, los datos leídos incluyen el número telefónico de la llamada entrante y que es provista como un servicio de “identificador de llamadas” por las compañías telefónicas. En nuestro caso el número de teléfono sirve como referencia para extraer información almacenada en una base de datos de posibles personas o clientes. Esa información puede ser utilizada para responder un mensaje de voz personalizado a través del “modo voz” del módem o para desplegar un mensaje visual en la pantalla de una computadora que notifica el usuario que ha realizado la llamada y que puede incluir una notificación auditiva con síntesis de texto a voz en la computadora que hospeda al módem.

1.2 ANTECEDENTES.

ANTECEDENTES EN LA UAM.

En la Universidad Autónoma Metropolitana no se cuenta con algún trabajo con estas características, de ahí que se proponga como proyecto terminal en donde se apliquen conocimientos adquiridos en varias materias como Bases de Datos, Programación, Sistemas Digitales, Metodología, Análisis y Diseño de Sistemas de Información e Interacción entre ser humano y computadora.

ANTECEDENTES EXTERNOS A LA UAM.

Existen varios programas comerciales como son: que ofrecen este tipo de servicios pero que tienen ciertas restricciones de licencia, de opciones para manipular la base de datos, para generar reportes específicos o que funcionen en diferentes Sistemas Operativos

1.3 JUSTIFICACIÓN.

Uno de los problemas a los que se enfrentan los usuarios de servicios telefónicos es que éstos son limitados, tienen un costo adicional o simplemente no se proveen en nuestro país así que representan áreas de oportunidad para desarrollar aplicaciones de cómputo y dispositivos tecnológicos acordes a las necesidades de diferentes usuarios. Actualmente en nuestro país existen alrededor de 18 millones 73 mil usuarios de telefonía fija de los cuales la mayoría han expresado por lo menos una queja o sugerencia en algún servicio. Un caso común es que los usuarios no pueden imprimir una bitácora de las llamadas que reciben e que se ven limitadas a un número máximo típicamente cuarenta [1]. Con ello se generan muchos conflictos para el usuario ya que es cotidiano recibir un gran número de llamadas no deseadas que va desde el exnovio(a) que está enojado(a), empresas de telemarketing e inclusive porque existen personas que se dedican a la extorsión telefónica.
Este último caso es de especial interés y podría contrarrestarse si se hacen públicos y se distribuye vía modem o por Internet una base de datos con números telefónicos previamente identificados por una autoridad competente. Lo mencionado anteriormente sugiere la programación de un administrador de llamadas telefónicas mediante un módem que ofrezca el servicio de registro o de bloqueo y notificaciones que sirve además como control interno de las llamadas que se registran en periodos amplios por días, meses, semanas o años en una casa o empresa. Un Ingeniero en Computación posee los conocimientos necesarios para realizar un proyecto como el aquí presentado ya que incluye programación de diferentes tipos de dispositivos sin requerir la modificación o construcción de hardware. Los fundamentos de estos conocimientos se adquieren a lo largo de la trayectoria académica de la carrera de Ingeniería en Computación de la UAM Azcapotzalco.

2. OBJETIVO

Durante el desarrollo del proyecto se logró el siguiente objetivo:

Diseñar e Implementar un administrador de llamadas telefónicas a través de un módem programable que permita identificar el número de la llamada entrante provista por una compañía telefónica, así como la consulta de información a una base de datos con información asociada a diferentes números telefónicos para desplegar un mensaje visual y auditivo.

3. MARCO TEÓRICO.

La parte de software del sistema de información requiere la realización de una serie de etapas conocidas como ciclo de vida del software [Stu05]. Es posible adaptarlo como mejor convenga a un proyecto determinado; el ciclo de vida consta de las siguientes etapas:

* Análisis.- Esta etapa requiere estudiar y documentar el proceso de recepción de llamadas telefónicas mediante un módem. Se definirán todos los objetivos funcionales y no funcionales del sistema, en términos de los actores (o usuarios), los eventos que se producen y la respuesta del sistema a tales eventos. El análisis concluye con un documento de especificación de requerimientos, formado por un conjunto de diagramas y representaciones gráficas detallando los elementos citados.

* Diseño.- En esta etapa se diseñará una solución conceptual al problema. En términos generales, se identifican los objetos reales del dominio de la aplicación (así como los algoritmos necesarios para manipularlos) y se transformarán en entidades de software. Esta etapa también incluye el diseño de los mecanismos de persistencia o almacenamiento para los datos utilizados y el diseño de la interfaz del sistema con el usuario
* Implementación de los artefactos de diseño.- En la etapa de implementación se traduce la descripción del sistema producida en la etapa de diseño, a instrucciones de un lenguaje de programación de manera que se pueda generar un programa ejecutable. Dicho programa contendrá la lógica suficiente para comunicarse y controlar adecuadamente a todas las partes del sistema, así como mostrar la funcionalidad establecida en la etapa de análisis.

* Validación y verificación de la implementación.- En esta etapa se verifica que los elementos individuales del programa producido, cumpla con los mínimos requisitos de calidad, tratando de eliminar la mayor cantidad de defectos del código que hayan quedado de la etapa de implementación.

3.1 DESCRIPCIÓN DEL ENTORNO.

El sistema en su mayor parte funciona en forma automática, es decir no necesita la interacción del usuario para registrar las llamadas entrantes, pero sí para realizar reportes o consultas específicas, éste proceso se lleva a cabo de la siguiente forma:

El módem se conecta a una PC mediante el puerto serial (denominado COM), al generarse el tono de una llamada entrante se detecta utilizando uno de los comandos del módem programado mediante un lenguaje de programación. Una vez identificado el tono de llamada entrante (“RING” “RING” “Numero que Llama” “RING”...) se desglosa en pantalla los datos de la persona perteneciente al número del usuario que llama, al mismo tiempo se registra en la bitácora los datos que se mostraron en pantalla, además de la fecha y hora en la que entró la llamada. Como se puede apreciar en la figura 2 que contiene el diagrama de bloques de detalle.

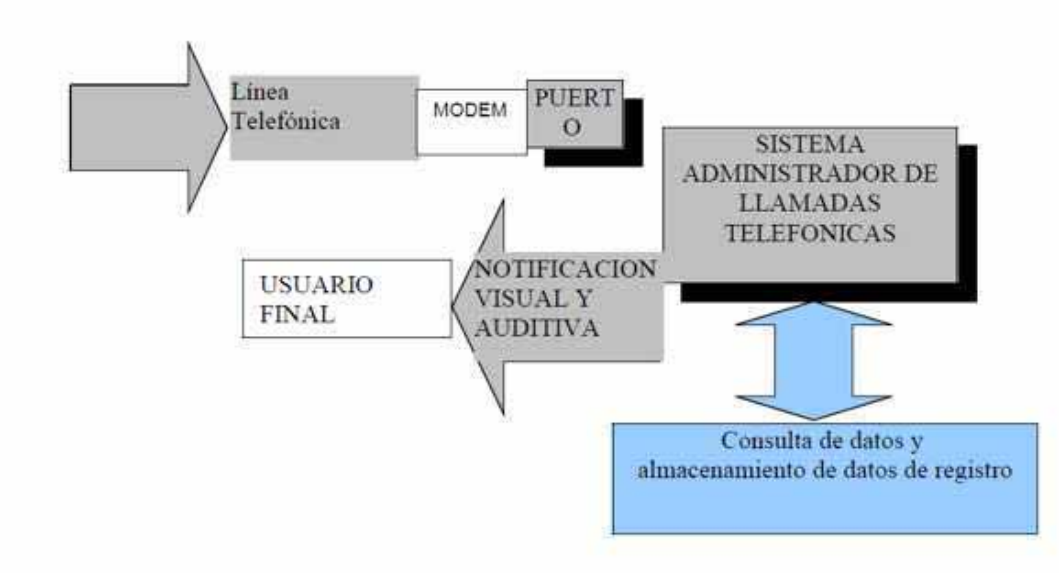


FIGURA 1. DIAGRAMA A BLOQUES.

4. DESARROLLO DEL PROYECTO

RECURSOS

Este proyecto es económico en cuanto a que requiere una pequeña cantidad de recursos de software y hardware. La siguiente lista muestra los recursos más importantes necesarios para el desarrollo, los cuales en su gran mayoría ya han sido adquiridos y no suponen ningún obstáculo para la realización del proyecto.

RECURSO	CANTIDAD	COSTO UNITARIO	DISPONIBILIDAD
Computadora portátil	1	$ 6,000	Adquirida
Módem externo	1	$300	Adquirida
Línea Telefónica	1	PAGO DE RENTA	Adquirida
MANEJADOR BD	1	0	LIBRE
LENGUAJE C	1	0	LIBRE
LENGUAJE JAVA	1	0	LIBRE

Todo el software empleado en la realización de este proyecto está liberado bajo la licencia pública general de GNU (General Public Licence), y es considerado

software libre. Debido a ello no genera ningún gasto adicional por concepto de licencias.

ACTIVIDADES

Lo más importante de éste proyecto es alcanzar los objetivos planteados al inicio, y obtener como resultado un sistema que lleve el control de las llamadas telefónicas entrantes, para ello se necesitan llevar a cabo las siguientes tareas para completar el proyecto:

1. Análisis y especificación de requerimientos.

2. Diseño de alto nivel de la capa de acceso a datos.- El diseño conceptual de la base de datos y el diseño físico de las tablas y vistas que vayan a ser utilizadas.

3. Implementación de las aplicaciones.

4. Codificación del código necesario para llevar a cabo el control de llamadas telefónicas entrantes.

5. Redacción del informe final.

5. RESULTADOS

Durante el desarrollo del presente proyecto se obtuvieron los siguientes resultados:

- Al analizar el funcionamiento de un módem y sus comandos de programación observamos que no todos los módems tienen las mismas funciones y características, esto depende del fabricante.

- JAVA y C tienen las herramientas necesarias para el acceso y manipulación de un módem con función de CallerID

- A través de una Base de Datos podemos manipular información haciendo consultas, inserciones y modificaciones de la misma.

- Una interface gráfica hace de un sistema de información una herramienta amigable para el usuario final.

6. ANÁLISIS Y DISCUCIÓN DE RESULTADOS.

Los objetivos planteados en la propuesta del proyecto fueron alcanzados obteniendo:

- Funcionamiento de un módem y sus comandos de programación:

Un **módem** es un dispositivo que sirve para modular y de modular (en amplitud, frecuencia, fase u otro sistema) una señal llamada *portadora* mediante otra señal de entrada llamada *moduladora*.

Los comandos AT son instrucciones codificadas que conforman un lenguaje de comunicación entre el hombre y un Terminal MODEM.

Algunos comandos no funcionan con todos los módems, esto depende de las funciones y características del mismo y de los proveedores

- Utilización de un lenguaje de programación para acceder al modem:

A través de JAVA y C pudimos leer una cadena enviada hacia el puerto serial y hacer la manipulación para obtener datos de una llamada entrante.

- Diseñó una base de datos

Se diseñó una base de datos que nos permitiera realizar consultas y registros de llamadas telefónicas

- Programación de una Interface

A través de NetBeans se implementó una interface gráfica que diera al sistema la característica de amigable para el usuario final

INTERFAZ GRÁFICA DE USUARIO DE LA APLICACIÓN DE ADMINISTRADOR DE LLAMADAS

Algunas de las tareas que realiza el sistema se enlistan y detallan a continuación, ilustradas con capturas de pantallas de la interfaz gráfica:

- Almacenamiento de datos de llamadas frecuentes (ver Figura 2).

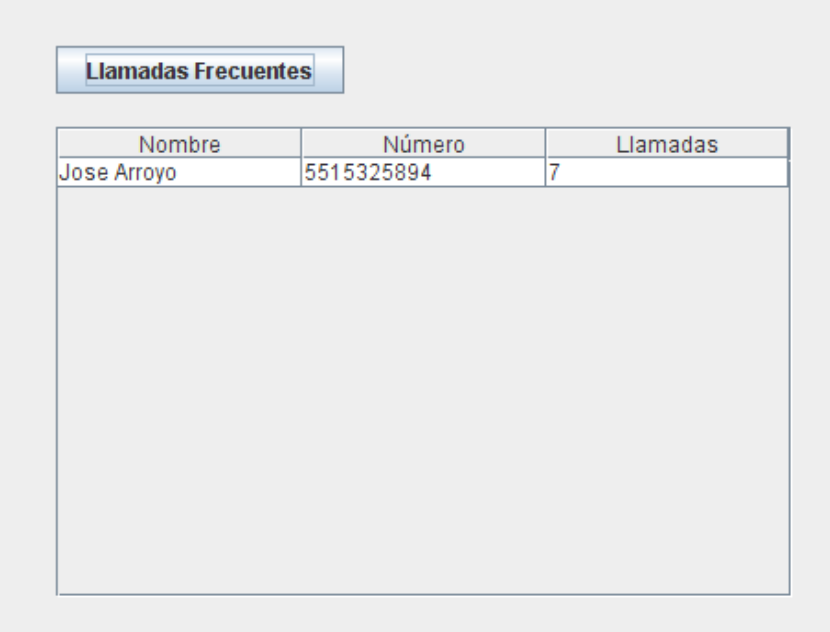


Figura 2 Pantalla que ilustra las llamadas más frecuentes.

- Identificación de números y datos de las llamadas entrantes (ver Figura 3).

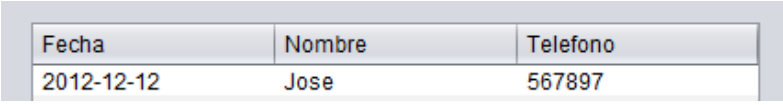


Figura. 3

- Bitácora de las llamadas entrantes (ver Figura 4).

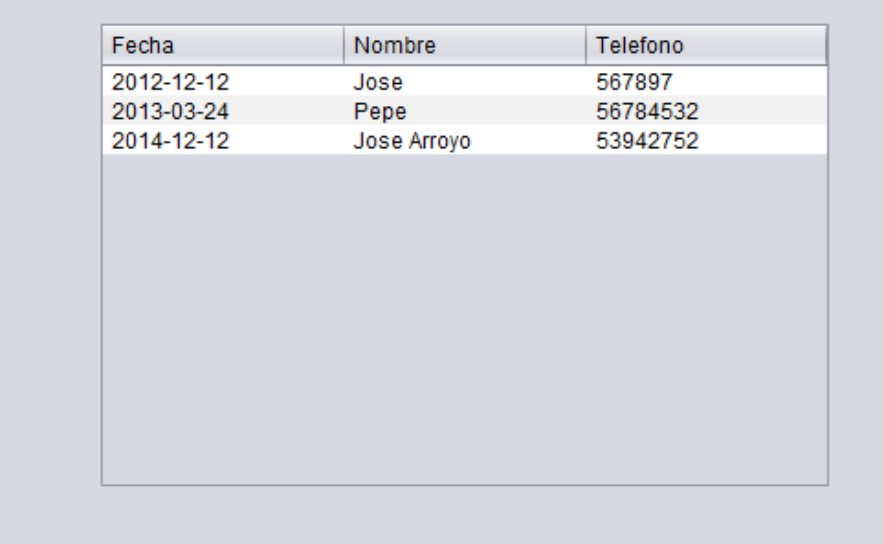


Figura. 4

- Consultas de las llamadas entrantes de periodos especificados por el usuario (ver Figura 5).

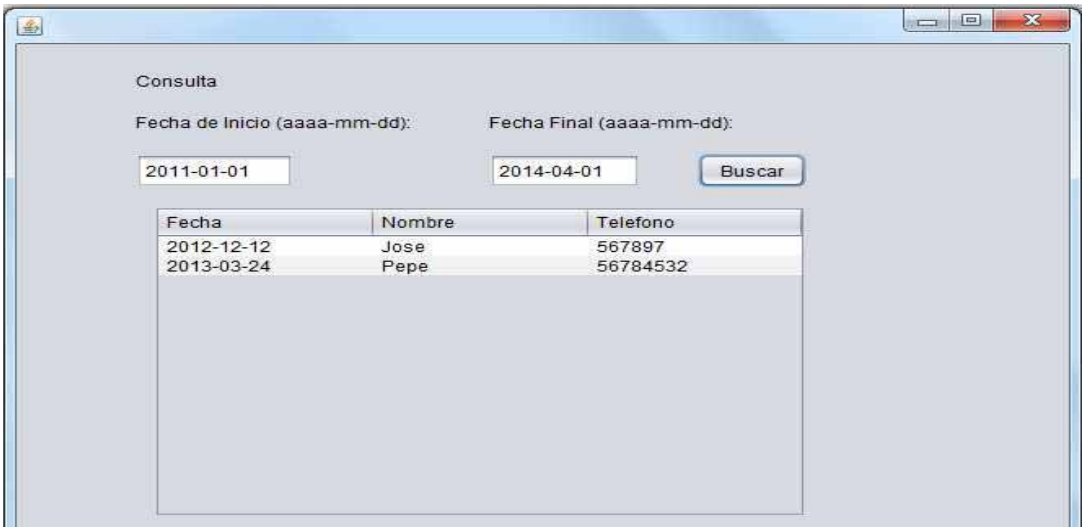


Figura 5.

ASPECTOS TÉCNICOS

La parte de software del sistema de información se realizó con una serie de etapas conocidas como ciclo de vida del software.

ARQUITECTURA.

Las necesidades de almacenamiento de los sistemas de información, no pueden ser atendidas utilizando únicamente el sistema de archivos. Esto hace necesario el uso de un gestor de bases de datos, que brinde las facilidades de almacenamiento, recuperación, búsqueda optimizada, integridad y seguridad, que son necesarios para implementar el almacén de datos. El sistema gestor de bases de datos que fue utilizado en este proyecto es MySQL. Este gestor cuenta con una variedad de herramientas de administración y programación que facilitan el diseño del almacén de datos, que se distribuye con licencia GLP y que facilita el desarrollo de aplicaciones “stand alone”.

PLATAFORMA DE PROGRAMACIÓN.

Para desarrollar un sistema de información, es necesario utilizar no sólo un lenguaje de programación, sino un conjunto de servicios que soporten correctamente la

operación de las aplicaciones. Una parte importante de los servicios lo constituye el sistema operativo, que brinda el control de bajo nivel sobre los dispositivos de almacenamiento, de visualización y de comunicación, entre otros.

Por otro lado, los lenguajes de programación normalmente se complementan con un una biblioteca de soporte, las cuales proporcionan una capa de servicios de alto nivel sobre el sistema operativo. La biblioteca de soporte permite la realización de tareas diversas, que incluyen por ejemplo, a las siguientes:

- Diseño de la interfaz de usuario.
- Programación de la interfaz de acceso a datos.
- Manipulación de los comandos del módem.

Una plataforma consolidada y con gran aceptación para la elaboración de aplicaciones es Java de Sun Microsystems (Oracle). Las aplicaciones desarrolladas con Java, son fácilmente desplegadas en casi cualquier computadora de escritorio y sobre una gran variedad de sistemas operativos para los que exista una máquina virtual.

La tecnología JDBC de Java, permite crear una interfaz de acceso a datos casi con cualquier manejador relacional, incluyendo MySQL y la biblioteca YAC la cual es programable mediante JAVA y cualquier otro lenguaje de programación que pueda escuchar sockets.

YAC, es un sistema de software basado en identificación de llamadas que hace que el consumo de la información de identificación de llamadas sea usado por JAVA y prácticamente cualquier otro lenguaje de programación a través del cual se puedan escuchar sockets. [2]

YAC fue escrito con código fuente en C++ y se distribuye bajo la Licencia Pública General GNU.

En el presente proyecto YAC es usado para enviar información de identificación de llamadas a un consumidor de Java y poder manipular la información tal como número telefónico, contacto, etc.

A continuación se ilustra la forma en como YAC trabaja:

- Simulación del envío de la cadena para la lectura en puerto (VER FIGURA 6)

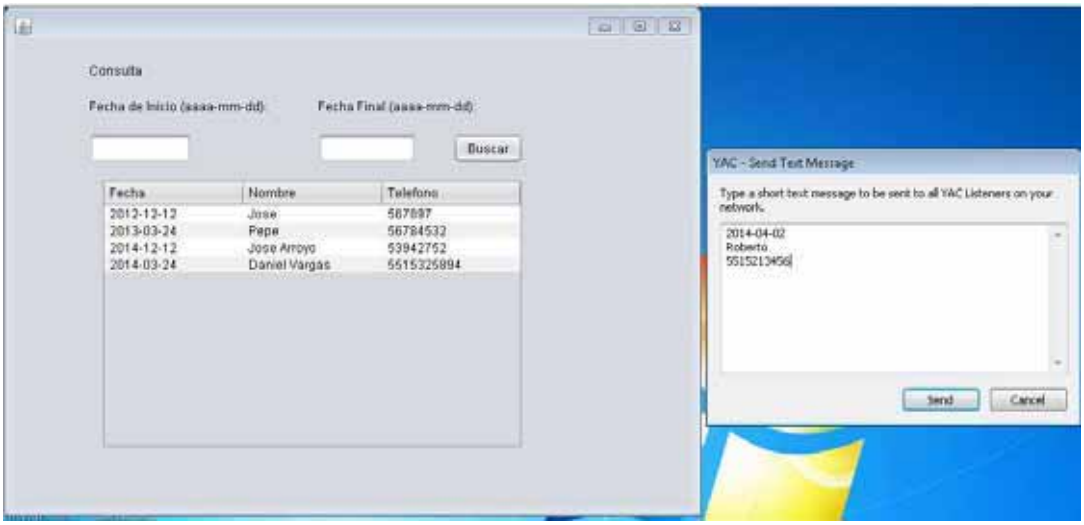


FIGURA 6 .Envío de la cadena al puerto

- Entrada de una llamada y lectura del puerto. (VER FIGURA 7)

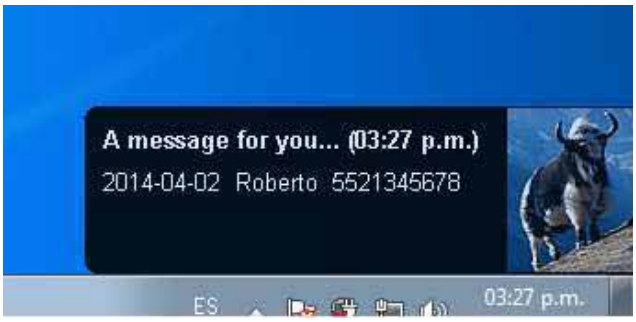


FIGURA 7. LLAMADA ENTRANTE

Los comandos AT son instrucciones codificadas que conforman un lenguaje de comunicación entre el hombre y un Terminal MODEM.

Algunos comandos no funcionan con todos los módems, esto depende de las funciones y características del mismo y de los proveedores.

A continuación están las instrucciones más usadas, estas se aplican a la mayoría de los módems. [3]

Si vas a hacer una llamada:

ATDT# Llamar con tonos al número telefónico #.

ATDP# Llamar con pulsos al número telefónico #.

(Después de marcar y contestar, se envía el protocolo).

Ejemplo: para llamar a Altair desde Torreón: ATDT165710.

Para recibir una llamada:

ATS0=n Descolgar automáticamente al llamar n veces.

Si n es 0 entonces nunca descuelga.

Ejemplo: contestar después de 4 llamadas ATS0=4.

ATA Descolgar y Contestar con protocolo.

(Sólo uno de los dos debe contestar con protocolo).

ATH1 Descolgar.

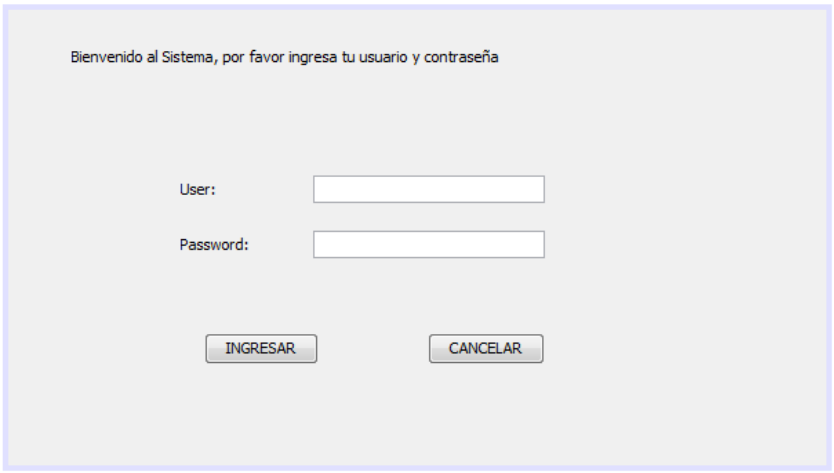
La API YAC abstrae este nivel, por lo que desde Java se puede trabajar como objetos Socket, como se usó en este proyecto.

Es por ello que estas herramientas fueron seleccionadas para el desarrollo de éste proyecto.

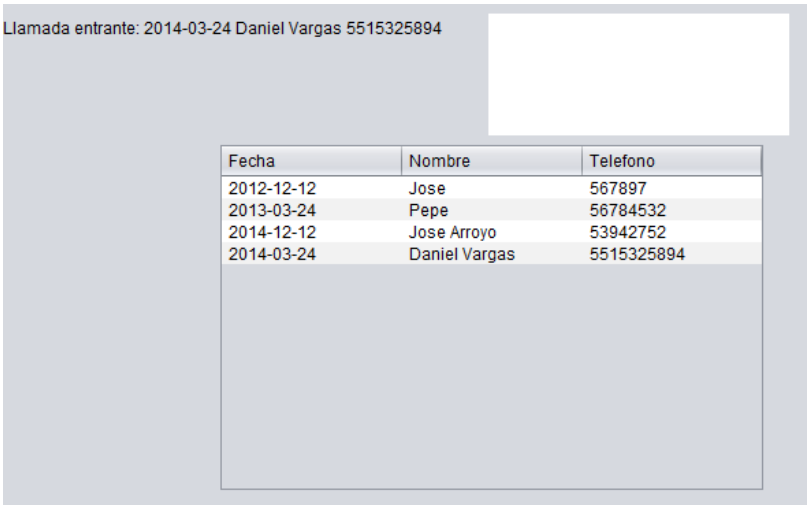
6. CAPTURAS DE PANTALLAS DE LA INTERFAZ DE USUARIO

Durante la interacción del usuario final con el sistema, las pantallas que comúnmente observará para realizar las diferentes acciones son las siguientes:

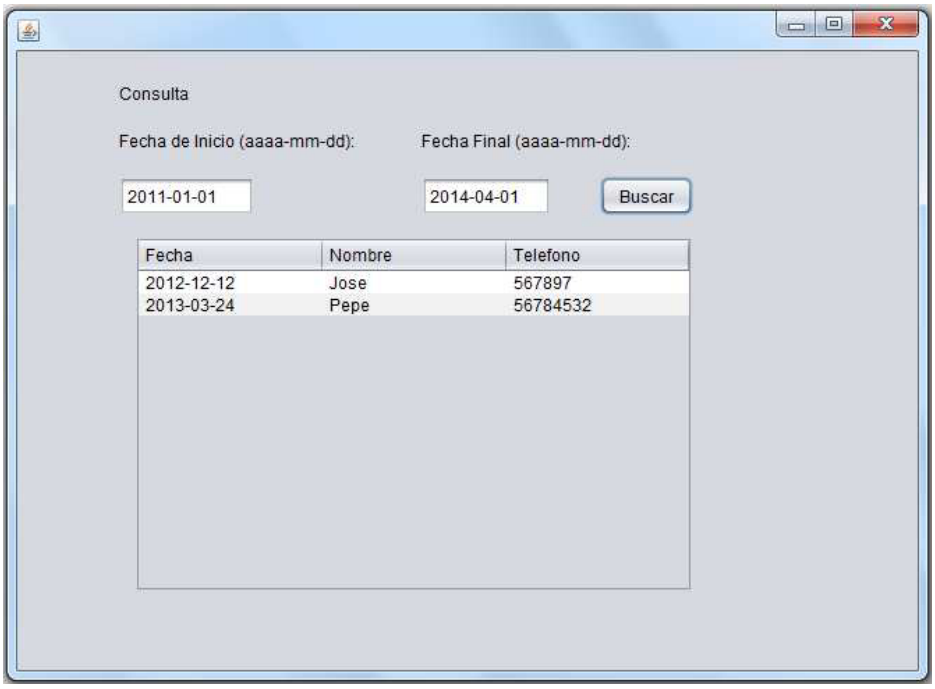
PANTALLA DE INICIO



LLAMADA ENTRANTE

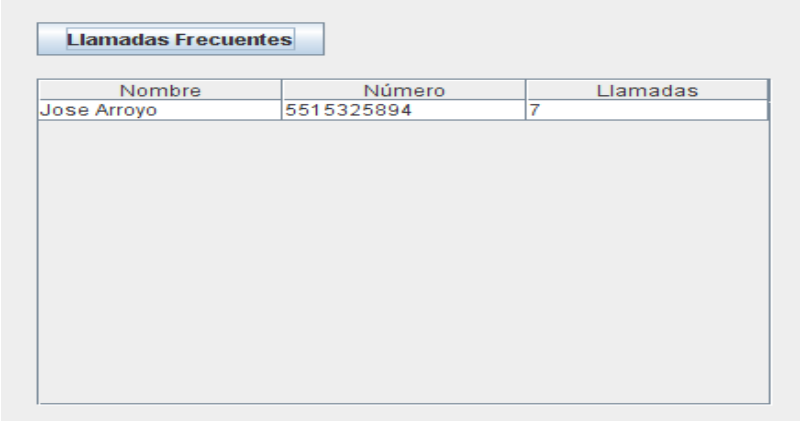


CONSULTA DE LLAMADAS POR FECHA



Para propósitos de demostración, y en caso de que la línea telefónica en un conmutador, o en una línea comercial o de casa no se cuente con el servicio de identificación de llamadas, se puede simular una llamada mediante la API YAC, a continuación una ilustración de una llamada entrante.

CONSULTA DE LLAMADAS FRECUENTES



8. CÓDIGO FUENTE.

```
00001
00005 package examples;
00006
00007 import java.awt.Toolkit;
00008 import
java.io.BufferedReader;
00009 import java.io.IOException;
00010 import
java.io.InputStreamReader;
00011 import java.net.ServerSocket;
00012 import java.net.Socket;
00013
00014 import
java.io.BufferedReader;
00015 import java.io.IOException;
00016 import
java.io.InputStreamReader;
00017 import java.net.ServerSocket;
00018 import java.net.Socket;
00019 import java.sql.ResultSet;
00020 import java.sql.SQLException;
00021 import java.util.Date;
00022 import java.util.Properties;
00023 import
javax.swing.table.DefaultTableModel
;
00024 import
javax.swing.table.TableColumn;
00025 import
javax.swing.text.TableView;
00026
00032  public class CallerID
extends javax.swing.JFrame {
00033
00037     public CallerID() {
00038         initComponents();
00039     }
00040
00046
@SuppressWarnings("unchecked")
00047     // <editor-fold
defaultstate="collapsed"
```

```
desc="Generated Code">//GEN-
BEGIN:initComponents
00048
00049
00053         private void
initComponents() {
00054
00055         jLabel1 = new
javax.swing.JLabel();
00056         jScrollPane2 = new
javax.swing.JScrollPane();
00057         jTable2 = new
javax.swing.JTable();
00058         jTextField1 = new
javax.swing.JTextField();
00059         jTextField2 = new
javax.swing.JTextField();
00060         jButton1 = new
javax.swing.JButton();
00061         jLabel2 = new
javax.swing.JLabel();
00062         jLabel3 = new
javax.swing.JLabel();
00063         jLabel4 = new
javax.swing.JLabel();
00064         jButton2 = new
javax.swing.JButton();
00065
00066
setDefaultCloseOperation(javax.swin
g.WindowConstants.EXIT_ON_CLOSE);
00067
00068         jTable2.setModel(new
javax.swing.table.DefaultTableModel
(
00069             new Object [][] {
00070                 {null, null,
null, null},
00071                 {null, null,
null, null},
00072                 {null, null,
null, null},
00073                 {null, null,
null, null}
00074             },
```

```
00075             new String [] {
00076                 "Title 1",
"Title 2", "Title 3", "Title 4"
00077             }
00078         ));
00079
jScrollPane2.setViewportView(jTable
2);
00080
00081
jButton1.setText("Buscar");
00082
jButton1.addActionListener(new
java.awt.event.ActionListener() {
00083             public void
actionPerformed(java.awt.event.Acti
onEvent evt) {
00084
jButton1ActionPerformed(evt);
00085             }
00086         });
00087
00088
jLabel2.setText("Consulta ");
00089
00090
jLabel3.setText("Fecha de Inicio
(aaaa-mm-dd):");
00091
00092
jLabel4.setText("Fecha Final (aaaa-
mm-dd):");
00093
00094
jButton2.setText("Llamadas
Frecuentes");
00095
jButton2.addActionListener(new
java.awt.event.ActionListener() {
00096             public void
actionPerformed(java.awt.event.Acti
onEvent evt) {
00097
jButton2ActionPerformed(evt);
00098             }
```

```
00099         });
00100
00101
javax.swing.GroupLayout layout =
new
javax.swing.GroupLayout(getContentP
ane());
00102
getContentPane().setLayout(layout);
00103
layout.setHorizontalGroup(
00104
layout.createParallelGroup(javax.sw
ing.GroupLayout.Alignment.LEADING)
00105
.addGroup(layout.createSequentialGr
oup()
00106                 .addGap(73,
73, 73)
00107
.addGroup(layout.createParallelGrou
p(javax.swing.GroupLayout.Alignment
.LEADING)
00108
.addComponent(jLabel2)
00109
.addGroup(layout.createParallelGrou
p(javax.swing.GroupLayout.Alignment
.TRAILING)
00110
.addGroup(layout.createSequentialGr
oup()
00111
.addGroup(layout.createParallelGrou
p(javax.swing.GroupLayout.Alignment
.LEADING, false)
00112
.addGroup(layout.createSequentialGr
oup()
00113
.addComponent(jTextField1,
javax.swing.GroupLayout.PREFERRED_S
IZE, 97,
javax.swing.GroupLayout.PREFERRED_S
IZE)
00114
.addPreferredGap(javax.swing.Layout
```

```
Style.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZ
E, Short.MAX_VALUE))
00115
.addGroup(layout.createSequentialGr
oup()
00116
.addComponent(jLabel3)
00117
.addPreferredGap(javax.swing.Layout
Style.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZ
E, Short.MAX_VALUE)
00118
.addComponent(jLabel1)
00119
.addGap(42, 42, 42)))
00120
.addGroup(layout.createParallelGrou
p(javax.swing.GroupLayout.Alignment
.LEADING)
00121
.addComponent(jLabel4)
00122
.addGroup(layout.createSequentialGr
oup()
00123
.addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_S
IZE, 93,
javax.swing.GroupLayout.PREFERRED_S
IZE)
00124
.addGap(35, 35, 35)
00125
.addComponent(jButton1))))
00126
.addGroup(layout.createParallelGrou
p(javax.swing.GroupLayout.Alignment
.LEADING)
00127
.addComponent(jButton2)
00128
.addComponent(jScrollPane2,
javax.swing.GroupLayout.PREFERRED_S
IZE, 401,
```

```
javax.swing.GroupLayout.PREFERRED_S
IZE))))
00129
.addContainerGap(161,
Short.MAX_VALUE))
00130         );
00131
layout.setVerticalGroup(
00132
layout.createParallelGroup(javax.sw
ing.GroupLayout.Alignment.LEADING)
00133
.addGroup(layout.createSequentialGr
oup()
00134
.addGroup(layout.createParallelGrou
p(javax.swing.GroupLayout.Alignment
.LEADING)
00135
.addGroup(layout.createSequentialGr
oup()
00136
.addGap(38, 38, 38)
00137
.addComponent(jLabel1,
javax.swing.GroupLayout.PREFERRED_S
IZE, 22,
javax.swing.GroupLayout.PREFERRED_S
IZE))
00138
.addGroup(layout.createSequentialGr
oup()
00139
.addGap(22, 22, 22)
00140
.addComponent(jLabel2)
00141
.addGap(18, 18, 18)
00142
.addGroup(layout.createParallelGrou
p(javax.swing.GroupLayout.Alignment
.BASELINE)
00143
.addComponent(jLabel3)
00144
.addComponent(jLabel4))))
```

```
00145                 .addGap(18,
18, 18)
00146
.addGroup(layout.createParallelGrou
p(javax.swing.GroupLayout.Alignment
.LEADING)
00147
.addGroup(layout.createParallelGrou
p(javax.swing.GroupLayout.Alignment
.BASELINE)
00148
.addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_S
IZE,
javax.swing.GroupLayout.DEFAULT_SIZ
E,
javax.swing.GroupLayout.PREFERRED_S
IZE)
00149
.addComponent(jButton1))
00150
.addComponent(jTextField1,
javax.swing.GroupLayout.PREFERRED_S
IZE,
javax.swing.GroupLayout.DEFAULT_SIZ
E,
javax.swing.GroupLayout.PREFERRED_S
IZE))
00151
.addPreferredGap(javax.swing.Layout
Style.ComponentPlacement.RELATED,
29, Short.MAX_VALUE)
00152
.addComponent(jButton2)
00153                 .addGap(18,
18, 18)
00154
.addComponent(jScrollPane2,
javax.swing.GroupLayout.PREFERRED_S
IZE, 256,
javax.swing.GroupLayout.PREFERRED_S
IZE)
00155                 .addGap(67,
67, 67))
00156         );
00157
00158         pack();
```

```
00159     }// </editor-fold>//GEN-END:initComponents
00160
00161     private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_jButton1ActionPerformed
00162         // TODO add your handling code here:
00163         consultaLlamadas(jTextField1.getText(),jTextField2.getText());
00164     }//GEN-LAST:event_jButton1ActionPerformed
00165
00166     private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_jButton2ActionPerformed
00167         // TODO add your handling code here:
00168         llamadasFrecuentes();
00169     }//GEN-LAST:event_jButton2ActionPerformed
00170
00174     public static void main(String args[]) {
00175
00176
00177         try {
00178             for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
00179                 if ("Nimbus".equals(info.getName())) {
00180                     javax.swing.UIManager.setLookAndFeel(info.getClassName());
00181                     break;
00182                 }
00183             }
00184         } catch (ClassNotFoundException ex) {
```

```
00185 java.util.logging.Logger.getLogger(CallerID.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
00186         } catch (InstantiationException ex) {
00187 java.util.logging.Logger.getLogger(CallerID.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
00188         } catch (IllegalAccessException ex) {
00189 java.util.logging.Logger.getLogger(CallerID.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
00190         } catch (javax.swing.UnsupportedLookAndFeelException ex) {
00191 java.util.logging.Logger.getLogger(CallerID.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
00192         }
00193         //</editor-fold>
00194 
00195         final CallerID callerID = new CallerID();
00196         callerID.initTable();
00201 java.awt.EventQueue.invokeLater(new Runnable() {
00202             public void run() {
00203 callerID.setVisible(true);
00204             }
00205         });
00206 
00207         YACListener yacl = new YACListener();
```

```
00208          yacl.jLabel1 =
callerID.jLabel1;
00209          yacl.callerId =
callerID;
00210          yacl.mainpool();
00211
00212      }
00213
00218      void initTable(){
00222          FachadaBD fdb = new
FachadaBD();
00223
fdb.connectToAndQueryDatabase("loca
lhost", "3306","bitacora" , "root",
"admin123");
00224
00225          DefaultTableModel
model = new DefaultTableModel();
00226
00230
model.addColumn("Fecha");
00231
model.addColumn("Nombre");
00232
model.addColumn("Telefono");
00233
00234
jTable2.setModel(model);
00235
00236
00237          try{
00241              ResultSet rs =
fdb.ejecutaSQLreturnRS("Select *
from registro_llamadas");
00242
00246              while(rs.next()){
00247
System.out.println(rs.getString(2)
+ " " + rs.getString(3) + "
"+rs.getString(4)+"\n");
00248                      Object[]
fila = new Object[3];
00249
fila[0]=rs.getString(4);
```

```
00250
fila[1]=rs.getString(2);
00251
fila[2]=rs.getString(3);
00252
model.addRow(fila);
00253             }
00254         }catch(SQLException
sqlex){
00255
System.out.println("Excepcion
lanzada "+sqlex);
00256         }
00257
fdb.disconnectDatabase();
00258
00259     }
00260
00265     public void
consultaLlamadas(String
fechaInicio,String fechaFin){
00266         FachadaBD fdb = new
FachadaBD();
00271
fdb.connectToAndQueryDatabase("loca
lhost", "3306","bitacora" , "root",
"admin123");
00272
00273         DefaultTableModel
model = new DefaultTableModel();
00274
00279
model.addColumn("Fecha");
00280
model.addColumn("Nombre");
00281
model.addColumn("Telefono");
00282
00283
jTable2.setModel(model);
00284
00285         //jTable2.setCo
00286
00287         try{
```

```
00288
00292             ResultSet rs =
fdb.ejecutaSQLreturnRS("Select *
from registro_llamadas where fecha
>= '"+fechaInicio+"' AND fecha <=
'"+fechaFin+"'");
00293
00294             while(rs.next()){
00295
System.out.println(rs.getString(2)
+ " " + rs.getString(3) + "
"+rs.getString(4)+"\n");
00296                       Object[]
fila = new Object[3];
00297
fila[0]=rs.getString(4);
00298
fila[1]=rs.getString(2);
00299
fila[2]=rs.getString(3);
00300
model.addRow(fila);
00301             }
00302         }catch(SQLException
sqlex){
00303
System.out.println("Excepcion
lanzada "+sqlex);
00304         }
00305
fdb.disconnectDatabase();
00306     }
00307
00311     public void
llamadasFrecuentes(){
00312         FachadaBD fdb = new
FachadaBD();
00313
fdb.connectToAndQueryDatabase("loca
lhost", "3306","bitacora" , "root",
"admin123");
00314
00315         DefaultTableModel
model = new DefaultTableModel();
00316
```

```
00320
model.addColumn("Nombre");
00321
model.addColumn("Número");
00322
model.addColumn("Llamadas");
00323
00324
jTable2.setModel(model);
00325
00326         try{
00327
00331             String query =
"SELECT nombre,numero,fecha,
count(numero) as num_registros FROM
registro_llamadas GROUP BY numero
ORDER BY num_registros desc";
00332
00333             ResultSet rs =
fdb.ejecutaSQLreturnRS(query);
00334
00335             while(rs.next()){
00336
//System.out.println(rs.getString(2
) + " " + rs.getString(3) + "
"+rs.getString(4)+"\n");
00337                     Object[]
fila = new Object[3];
00338
fila[0]=rs.getString(1);
00339
fila[1]=rs.getString(2);
00340
fila[2]=rs.getString(4);
00341
model.addRow(fila);
00342             }
00343         }catch(SQLException
sqlex){
00344
System.out.println("Excepcion
lanzada "+sqlex);
00345         }
00346
fdb.disconnectDatabase();
```

```
00347     }
00348
00349
00350     // Variables declaration
- do not modify//GEN-
BEGIN:variables
00351     private
javax.swing.JButton jButton1;
00352     private
javax.swing.JButton jButton2;
00353     private
javax.swing.JLabel jLabel1;
00354     private
javax.swing.JLabel jLabel2;
00355     private
javax.swing.JLabel jLabel3;
00356     private
javax.swing.JLabel jLabel4;
00357     private
javax.swing.JScrollPane
jScrollPane2;
00358     private
javax.swing.JTable jTable2;
00359     private
javax.swing.JTextField jTextField1;
00360     private
javax.swing.JTextField jTextField2;
00361     // End of variables
declaration//GEN-END:variables
00362
00363 }
00364
00365
00366 class YACListener
00367 {
00368         static String
SMTPSERVER = "smtpint.azc.uam.mx";
00369         static String
MAILSENDERADDRESS =
"alguien@mudo.com";
00370         static String
MAILRECIPIENTADDRESS =
"oha@correo.azc.uam.mx";
00371
```

```
00372         public
javax.swing.JLabel jLabel1;
00373         public CallerID
callerId;
00374
00375
00376         public void
mainpool()
00377         {
00378
00383             FachadaBD fdb =
new FachadaBD();
00384
fdb.connectToAndQueryDatabase("loca
lhost", "3306","bitacora" , "root",
"admin123");
00385
00391                 int port =
10629+0;
00392                 try
00393                 {
00394
00395
ServerSocket serverSocket = new
ServerSocket(port);
00396
System.out.println("Server waiting
for client on port " +
serverSocket.getLocalPort());
00397
00398
00399
String message = null;
00400
00401
00402                           while
(true)
00403                           {
00404
Socket socket =
serverSocket.accept();
00405
00409
BufferedReader input = new
BufferedReader(new
```

```
InputStreamReader(socket.getInputSt
ream()));
00413
message = input.readLine();
00414
System.out.println(message);
00418
if(message!=null){
00419
00420
String[] registro = new String[3];
00421
registro[0] = message;
00422
message = input.readLine();
00423
registro[1] = message;
00424
message = input.readLine();
00425
registro[2] = message;
00426
00427
this.jLabel1.setText("Llamada
entrante: " + registro[0]+"
"+registro[1]+" "+registro[2]);
00428
00436
int exito = fdb.EjecutaSQL("INSERT
INTO
registro_llamadas(nombre,numero,fec
ha)
values('"+registro[1]+"','"+registr
o[2]+"','"+registro[0]+"')");
00437
if(exito!= 0){
00438
System.out.println("Insercion
exitosa");
00439
}else{
00440
System.out.println("Insercion
fallida");
00441
}
```

```
00445
for(int i = 0; i < 4; i++){
00446
try{
00447
Toolkit.getDefaultToolkit().beep();
00448
Thread.sleep(500);
00449
} catch (Exception e){
00450
System.out.println(e.getMessage());
00451
}
00452
}
00456
callerId.initTable();
00460
break;
00461
}
00462
00463
socket.close();
00464
System.out.println("Connection
closed by client");
00465
00466                          }
00467
00468                  } catch
(IOException e)
00469                  {
00470
e.printStackTrace();
00471                  }
00472          }
00473 }
```

•Todo Clases Namespaces Archivos Funciones Variables**CallerID.java**

- Generado

DOCUMENTACION DEL CÓDIGO FUENTE

Documentación de namespaces

Paquetes examples

Clases

- **class** CallerID
- **class** YACListener

Descripción detallada

Administrador de Llamadas vía módem

Documentación de las clases

Referencia de la Clase examples.CallerID

Métodos públicos

- CallerID **()**
- **void** consultaLlamadas **(String fechaInicio, String fechaFin)**
- **void** llamadasFrecuentes **()**

Métodos públicos estáticos

- **static void** main **(String args[])**

Funciones del 'package'

- **void** initTable **()**

Descripción detallada

Autor:
José Arroyo
Definición en la línea 32 del archivo CallerID.java.

Documentación del constructor y destructor

examples.CallerID.CallerID ()
Creates new form **CallerID**
Definición en la línea 37 del archivo CallerID.java.

Documentación de las funciones miembro

void examples.CallerID.consultaLlamadas (String *fechaInicio*, String *fechaFin*)
Consulta de llamadas por Fecha
Conexion a la Base de Datos
Creación de las columnas de consulta
Query de Consulta a la Base de Datos por fecha
Definición en la línea 265 del archivo CallerID.java.

void examples.CallerID.initTable () [package]
Inicia la tabla buscando informacion en base de datos
Relizamos la conexión a la BD

Creación de las columnas de consulta

Realizamos la consulta a la base de datos para desplegar las llamadas almacenadas

Agregamos información al modelo para insertar en la tabla de despliegue

Definición en la línea 218 del archivo CallerID.java.

Gráfico de llamadas a esta función:

void examples.CallerID.llamadasFrecuentes ()

Consulta las llamadas frecuentes

Creación de las columnas de consulta

Query para la consulta de los número frecuentes

Definición en la línea 311 del archivo CallerID.java.

static void examples.CallerID.main (String *args*[]) [static]

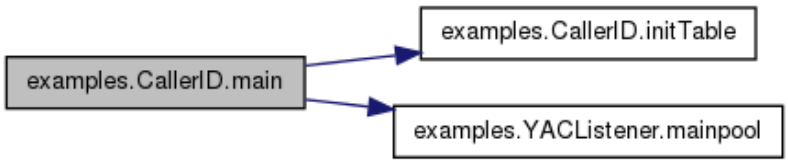
Parámetros:

args	the command line arguments

Create and display the form

Definición en la línea 174 del archivo CallerID.java.

Gráfico de llamadas para esta función:

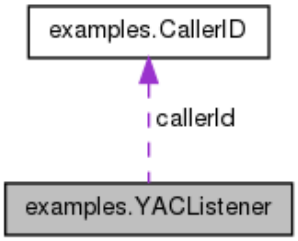


La documentación para esta clase fue generada a partir del siguiente fichero:

- examples/**CallerID.java**

Referencia de la Clase examples.YACListener

Diagrama de colaboración para examples.YACListener:



Métodos públicos

- **void** mainpool **()**

Atributos públicos

- **javax.swing.JLabel** jLabel1
- CallerID callerId

Atributos Estáticos del 'package'

- **static String** SMTPSERVER **= "smtpint.azc.uam.mx"**
- **static String** MAILSENDERADDRESS **= "alguien@mudo.com"**
- **static String** MAILRECIPIENTADDRESS **= "oha@correo.azc.uam.mx"**

Descripción detallada

Definición en la línea 366 del archivo CallerID.java.

Documentación de las funciones miembro

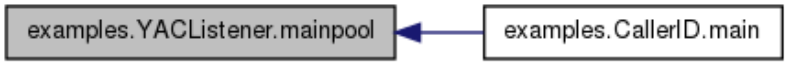
void examples.YACListener.mainpool ()

Inicia la base de datos
Puerto de Lectura de YAC
handle incoming call
Lectura de la linea de entrada
Si existe mensaje se leeen las demas líneas y se almacenan para su uso
Guardamos llamada en la bitácora
Query para inserción en la bitácora
Se envía notificacion audible 4 veces
Volvemos a cargar datos

Se detiene el ciclo

Definición en la línea 376 del archivo CallerID.java.

Gráfico de llamadas a esta función:



Documentación de los datos miembro

CallerID examples.YACListener.callerId

Definición en la línea 373 del archivo CallerID.java.

javax.swing.JLabel examples.YACListener.jLabel1

Definición en la línea 372 del archivo CallerID.java.

String examples.YACListener.MAILRECIPIENTADDRESS = "oha@correo.azc.uam.mx" [static, package]

Definición en la línea 370 del archivo CallerID.java.

String examples.YACListener.MAILSENDERADDRESS = "alguien@mudo.com" [static, package]

Definición en la línea 369 del archivo CallerID.java.

String examples.YACListener.SMTPSERVER = "smtpint.azc.uam.mx" [static, package]

Definición en la línea 368 del archivo CallerID.java.

La documentación para esta clase fue generada a partir del siguiente fichero:

- examples/**CallerID.java**

Documentación de archivos

Referencia del Archivo examples/CallerID.java

Clases

- **class** examples.CallerID
- **class** examples.YACListener

Paquetes

- **package** examples

11. **CONCLUSIONES**

Durante el desarrollo e implementación del presente proyecto se cumplieron los siguientes objetivos.

- Analizar el funcionamiento de un módem y sus comandos de programación.
- Utilizar un lenguaje de programación para acceder al módem.
- Diseñar una base de datos que incluya números telefónicos e información adicional con la finalidad de dar notificaciones visuales y auditivas en la pantalla y el sistema de audio de la computadora que hospeda al módem.
- Programar una Interface de Programación de Aplicaciones existente para controlar la salida de audio de una computadora personal que envía notificaciones de texto y auditivas personalizadas correspondientes a una llamada entrante.
- Diseñar y programar una herramienta software que proporcione una interfaz gráfica de usuario amigable para los usuarios del sistema administrador de llamadas telefónicas entrantes.

10. **REFERENCIAS**

[Sch05] Stephen R. Schach. Análisis y diseño orientado a objetos con UML y el Proceso Unificado.
McGraw Hill, México, 2005.
[1] TELMEX:
http://www.telmex.com/mx/ (2008)

[2] http://www.devx.com/Java/Article/28249

[3] http://support.usr.com/support/3cxm756/3cxm756-spanish-ug/atcoms.htm