Instituto Politécnico Nacional

Escuela Superior de Cómputo

DOCUMENTO TÉCNICO

Generador y Aplicador de Exámenes

Programación Orientada a Objetos

Profesor: Roberto Tecla Parra

Grupo: 2CM3

Fecha: 11 / Diciembre /2017

Alumnos:

- Álvarez Barajas Enrique 2014030045
- Calva Hernández José Manuel 2017630201
- Ruíz López Luis Carlos 2014081397

Índice

Instrucciones de ejecución

Para la compilación, seguiremos estos pasos:

1. Cargar el script de la Base de Datos (AplicadorExamen.sql) en nuestro sistema.

```
mysql> create database AplicadorExamen;
Query OK, 1 row affected (0.00 sec)

mysql> use AplicadorExamen;
Database changed
mysql> source C:\Users\manue\Desktop\Proyecto POO\AplicadorExamen.sql
```

2. Abriremos dos terminales, una en la carpeta del Cliente, y otra en la carpeta del Servidor. A continuación, ejecutaremos el Servidor sin olvidar adjuntar el paquete referente a sql por medio del siguiente comando.

```
akotadi@Laptop: ~/Documents/POO/ProyectoExamen
akotadi@Laptop:~/Documents/POO/ProyectoExamen$ java -cp .:mysqlcon.jar TestServe
r
```

3. En la terminal del Cliente ejecutaremos el siguiente comando para iniciar el aplicador de examen.

```
akotadi@Laptop: ~/Documents/POO/ProyectoExamen
akotadi@Laptop:~/Documents/POO/ProyectoExamen$ java Connect
```

4. Una vez ejecutado el comando, se iniciará el programa y se podrá proceder a usarlo normalmente.

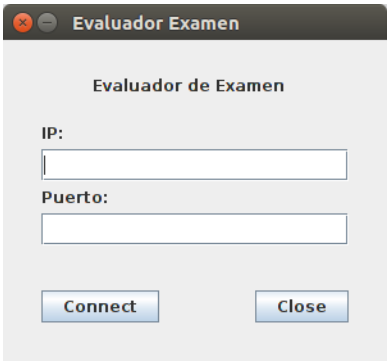
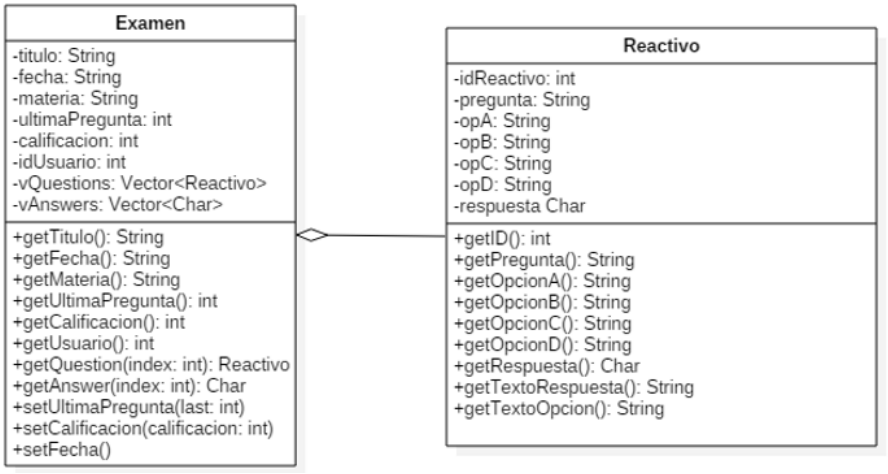
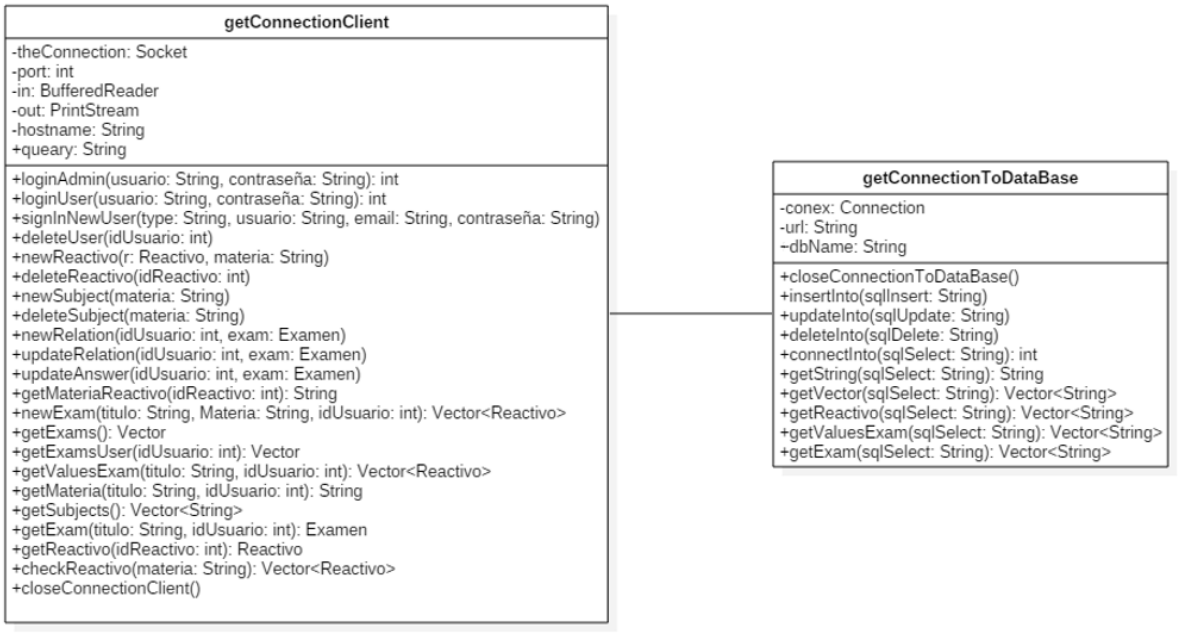


Diagrama de Clases

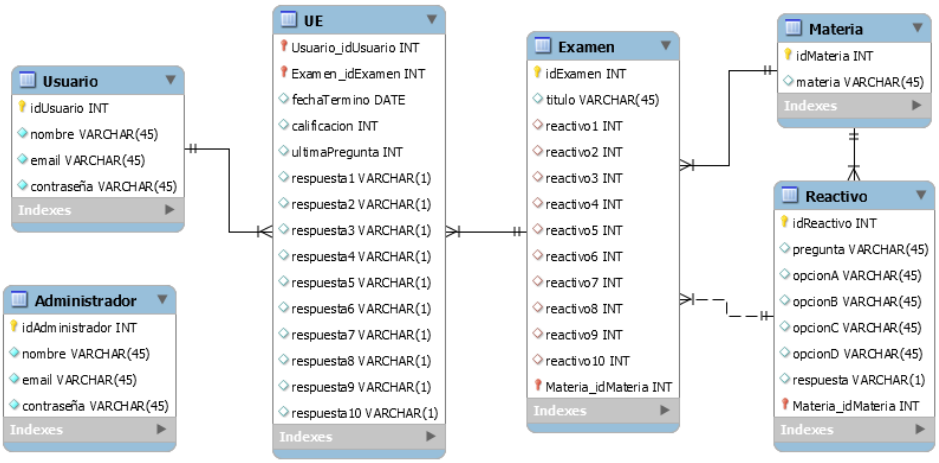




Desarrollo

Base de Datos.

Para comenzar el proyecto, lo primero que se hizo fue modelar la base de datos que sería usada para la base de datos, esto fue hecho por medio de “MySQL Workbench”.



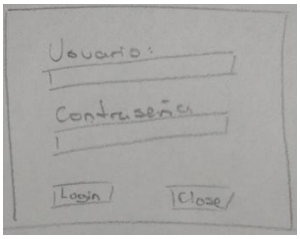
Una vez modelada, se procede a su creación por medio de la terminal de MySQL, se mostrará brevemente la forma de hacerlo.

```
mysql> create database AplicadorExamen;
Query OK, 1 row affected (0.04 sec)

mysql> use AplicadorExamen;
Database changed
mysql> create table Usuario(
    -> idUsuario int not null auto_increment,
    -> nombre varchar(50) not null,
    -> email varchar(50) not null,
    -> contraseña varchar(20) not null,
    -> primary key(idUsuario)
    -> );
Query OK, 0 rows affected (0.23 sec)
```

Interfaces.

Una vez modelada, se procedió a diseñar las interfaces que serían usadas por los usuarios, el diseño se realizó primero en papel, y se fue ajustando al pasarlo a código. Un ejemplo de esto es el modelado de la pantalla de Login:

1. Modelamos en papel.



2. Dentro de la parte del código.
   a. Tomamos las librerías que vamos a usar.

```
// LIBRERÍAS
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
```

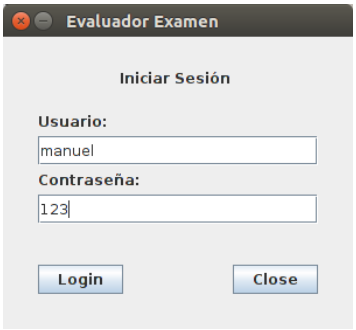
   b. Declaramos las variables que usaremos en la ventana.

```
// VARIABLES
JButton bLogin, bClose; // Botones de la ventana
JLabel lUsuario, lContraseña, lConnectionText, lTittle; // Etiquetas para colocar texto
JTextField tUsuario, tContraseña; // Campos de entrada para leer texto
JPanel p1,p2; // Paneles para contener otros elementos
```

   c. Mediante el constructor, creamos la ventana para el usuario.

```
// CONSTRUCTOR
public Login(){

    // INICIALIZACIÓN PARA LA VENTANA
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); // Añade el cierre predeterminado de la ventana
    setTitle("Evaluador Examen"); // Coloca título a la ventana
    setResizable(false); // Evita que el usuario cambie el tamaño predeterminado
    getContentPane().setLayout(null); // Inicializa el contenedor sin layout permitiéndonos trabajar libremente
    java.awt.Dimension screenSize = java.awt.Toolkit.getDefaultToolkit().getScreenSize(); // Tomamos las medidas de la pantalla del usuario
    setBounds((screenSize.width-300)/2, (screenSize.height-250)/2, 300, 250); // Colocamos la ventana en el centro

    // INICIALIZACIÓN DE COMPONENTES

    lTittle = new JLabel("Iniciar Sesión"); // Inicia una etiqueta con un texto predeterminado
    getContentPane().add(lTittle); // Añadimos la etiqueta a la ventana
    lTittle.setBounds(300/2-50, 20, 150, 30); // Colocamos la etiqueta en el lugar deseado de la ventana

    p1 = new JPanel(); // Inicializa un panel donde agregaremos los campos de entrada
    p1.setLayout(new GridLayout(4,1)); // Establecemos un diseño predeterminado
    lUsuario = new JLabel("Usuario: "); // Inicializamos una etiqueta para el usuario con un texto predeterminado
    tUsuario = new JTextField(30); // Inicializamos un campo de entrada para el usuario
    lContraseña = new JLabel("Contraseña: "); // Inicializamos una etiqueta para la contraseña con un texto predeterminado
    tContraseña = new JTextField(20); // Inicializamos un campo de entrada para la contraseña
    p1.add(lUsuario);                       p1.add(tUsuario); // Añadimos los campos referentes al usuario al panel
    p1.add(lContraseña);                    p1.add(tContraseña); // Añadimos los campos referentes a la contraseña al panel
    getContentPane().add(p1); // Añadimos el panel a la ventana
    p1.setBounds(30, 60, 240, 100); // Colocamos el panel en el lugar deseado de la ventana

    p2 = new JPanel(); // Inicializa un panel donde agregaremos los campos de entrada
    p2.setLayout(new BorderLayout()); // Establecemos un diseño predeterminado
    lConnectionText = new JLabel(" "); // Inicializa una etiqueta en blanco por si necesitásemos mostrar texto
    bLogin = new JButton("Login");          bLogin.addActionListener(this); // Inicializa un botón con un texto predeterminado y se añade el método de escucha
    bClose = new JButton("Close");          bClose.addActionListener(this); // Inicializa un botón con un texto predeterminado y se añade el método de escucha
    p2.add(lConnectionText, BorderLayout.NORTH); // Dentro del diseño predeterminado, colocamos la etiqueta en la parte norte
    p2.add(bLogin, BorderLayout.WEST); // Dentro del diseño predeterminado, colocamos la etiqueta en la parte oeste
    p2.add(bClose, BorderLayout.EAST); // Dentro del diseño predeterminado, colocamos la etiqueta en la parte este
    getContentPane().add(p2); // Añadimos el panel a la ventana
    p2.setBounds(300/2-120, 180, 240, 40); // Colocamos el panel en el lugar deseado de la ventana

}
```

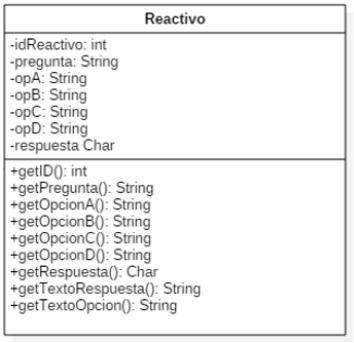
3. El resultado final es la siguiente ventana, que es como la habíamos diseñado al inicio.



Clases.

Una vez concluido el diseño de las interfaces, lo siguiente es el modelado de clases que se usarán para guardar datos, en nuestro caso particular, se usarán dos clases principalmente, la de Reactivo y la de Examen. Se mostrará el proceso de creación de la clase Reactivo:

1. Se modela la clase por medio de las especificaciones dadas y el programa StarUML.



2. Para este caso no se necesitará importar ninguna librería, por tanto, procedemos a establecer las variables de la clase.

```
// VARIABLES
private int idReactivo;
private String pregunta, opA, opB, opC, opD;
private char respuesta;
```

3. Estableceremos el constructor de la clase de la siguiente manera.

```
// CONSTRUCTOR
public Reactivo(int n, String s, String A, String B, String C, String D, String c){

    idReactivo = n; // Establecemos el ID del reactivo a partir del valor dado
    pregunta = s.toUpperCase(); // Para facilidad de trabajo, unificamos todo el texto en mayúsculas
    opA = A.toUpperCase();
    opB = B.toUpperCase();
    opC = C.toUpperCase();
    opD = D.toUpperCase();
    c = c.toUpperCase();
    respuesta = c.charAt(0); // Dado que recibimos un String, hacemos el cambio al valor requerido

}
```

4. Declaramos los métodos que serán utilizados en la clase, dado que son similares, se ejemplificará con los principales.

```java
// El método devolverá el ID del reactivo.
public int getID(){
    return idReactivo;
}
```

```java
/* A partir del valor de una opción recibida,
    devolvemos el texto asociado a ella. */
public String getTexto(char select){
    if (select == 'A') {
        return opA;
    }
    else if (select == 'B') {
        return opB;
    }
    else if (select == 'C') {
        return opC;
    }
    else if (select == 'D') {
        return opD;
    }
    return null;
}
```

Servidor.

A continuación, se creará el servidor para el programa, lo primero que necesitamos es establecer las librerías que vamos a necesitar:

```java
// LIBRERÍAS
import java.io.*;
import java.net.*;
import java.util.*;
```

Posteriormente declararemos las variables que usaremos en la clase:

```java
// CLASE PRINCIPAL
public class TestServer{

    // PUERTO DEL SERVIDOR
    public final static int answerPort = 1025;

    // VARIABLES
    private ServerSocket theServer;
    private Socket theClient;
    private BufferedReader in;
    private PrintStream out;
    private getConnectionToDataBase connection;
```

Usaremos el constructor por defecto, así que pasaremos directamente a la clase principal del programa, dentro de la cual inicializaremos el servidor en el puerto seleccionado, colocándolo dentro de un bloque try porque puede generar excepciones:

```java
// MAIN
public static void main(String[] args) {

    // INICIAMOS EL SERVIDOR EN EL PUERTO
    try{
        theServer = new ServerSocket(answerPort);
```

Procederemos a establecer la conexión con el cliente, nuevamente dentro de un bloque try, a su vez inicializaremos el flujo de datos con el cliente, y comenzaremos a recibir comandos por medio de éste, dado que es un flujo de datos, puede generar excepciones por lo que también se coloca en un bloque try:

```
// RECIBIMOS LA CONEXIÓN DEL CLIENTE
try{
    theClient = theServer.accept();
    System.out.println("Connection with client established.\n");

    // ESTABLECEMOS EL FLUJO DE ENTRADA/SALIDA CON EL CLIENTE
    in = new BufferedReader(new InputStreamReader(theClient.getInputStream()));
    out = new PrintStream(theClient.getOutputStream());

    // VARIABLES PARA RECIBIR Y ENVIAR COMANDOS AL SERVIDOR
    String theInputLine, query;
    try{

        // INICIAMOS LA CONEXIÓN CON LA BASE DE DATOS
        connection = new getConnectionToDataBase();

        // ITERACIÓN INFINITA PARA INTERCAMBIO DE MENSAJES CON EL CLIENTE
        while(true){
            theInputLine = in.readLine();
            System.out.println("0");
            System.out.println(theInputLine);
```

Seguiremos con la conexión con la Base de Datos previamente creada, para ello se creará otra clase denominada getConnectionToDataBase, la cuál será la encargada de gestionar todos los intercambios con la Base de Datos. Iniciaremos escribiendo las librerías necesarias:

```
// LIBRERÍAS
import java.sql.*;
import java.util.*;
```

Procederemos a establecer las variables necesarias para la clase:

```
// CLASE PRINCIPAL
public class getConnectionToDataBase extends JPanel{

    // VARIABLES
    static Connection conex; // Clase de la librerías sql que permite la conexión con este tipo de BD
    static String url = "jdbc:mysql://localhost:3306/"; // URL de la BD predefinida
    static String dbName = "AplicadorExamen"; // Nombre de la BD en nuestro Servidor
```

Por medio del constructor iniciaremos la conexión con la BD local, pero esto puede generar excepciones en caso de no encontrarse la BD, por tanto se colocará dentro de un bloque try:

```
// CONSTRUCTOR
public getConnectionToDataBase(){

    //CONEXIÓN CON LA BASE DE DATOS
    try {
        Class.forName("com.mysql.jdbc.Driver"); // Usaremos el driver de MySQL
        // Introduciremos las credenciales necesarias para conectarse a la base de datos
        conex=DriverManager.getConnection(url+dbName, "root", "77Manuel95:" );
        System.out.println("Connection with database established.");
    }
    catch (Exception ex){
        System.out.println(ex);
        throw new EmptyStackException();
    }
}
```

Cliente.

Para el cliente, iniciaremos importando las librerías que necesitaremos.

```
// LIBRERÍAS
import java.io.*;
import java.net.*;
import java.util.*;
```

A continuación, declaramos las variables de clase que utilizaremos.

```
// VARIABLES
private Socket theConnection; // Socket para establecer conexión con el servidor
private int port; // Puerto a utilizar
private BufferedReader in; // Flujo de entrada
private PrintStream out; // Flujo de salida
private String hostname, query; // URL del servidor, órdenes en formato MySQL que serán enviadas
```

Utilizaremos el constructor para iniciar la conexión con el servidor, ésta será creada a partir de una URL y un puerto proporcionado por medio de la interfaz de la aplicación, a su vez, estableceremos el flujo de datos con el servidor. Como tanto inicializar la conexión como establecer el flujo de datos pueden generar excepciones, los colocaremos en bloques try.

```
// CONSTRUCTOR
public getConnectionClient(String ip, int port) { // Recibimos la IP y el Puerto por medio de la interfaz

    // Inicialización de variables de conexión
    hostname = ip;
    this.port = port;

    // INICIAMOS LA CONEXIÓN CON EL SERVIDOR
    try{
        theConnection = new Socket(hostname, port);

        // ESTABLECEMOS EL FLUJO DE ENTRADA/SALIDA CON EL SERVIDOR
        out = new PrintStream(theConnection.getOutputStream());
        in = new BufferedReader(new InputStreamReader(theConnection.getInputStream()));

    } catch (UnknownHostException e) {
        System.err.println(e);
        throw new EmptyStackException();
    } catch (IOException e) {
        System.err.println(e);
        throw new EmptyStackException();
    }
}
```

Interacción Cliente-Servidor.

Con esto, la aplicación ya puede comunicarse y únicamente queda establecer los métodos que interactuarán entre las interfaces y el servidor, para ello, el cliente proporcionará un comando a modo de frase que le indicará al servidor qué quiere hacer, a continuación, el cliente mandará la sentencia a ejecutar y el servidor procederá a ejecutarla. Como ejemplo de esto, Se mostrará el procedimiento para registrarse como un nuevo usuario.

1. Se manda a llamar el método contenido en el cliente desde la interfaz

```
// En caso de haberse seleccionado el botón de registrar
if(selected == bSignIn){

    String usuario, email, contraseña;
    usuario = tUsuario.getText(); // Iniciamos la variable usuario tomando el texto introducido en la ventana
    email = tEmail.getText(); // Iniciamos la variable email tomando el texto introducido en la ventana
    contraseña = tContraseña.getText(); // Iniciamos la variable contraseña tomando el texto introducido en la ventana

    // Verificamos que el usuario haya introducido valores mínimos en cada campo
    if (usuario.length()>2 && email.length()>2 && contraseña.length()>2) {

        // En caso de haber seleccionado el tipo de usuario "Administrador"
        if (rbA.isSelected()) {
            connect.SignInNewUser("Administrador",usuario,email,contraseña);
            Inicio start = new Inicio(connect);
            this.dispose();
            start.setVisible(true);
        }

        // En caso de haber seleccionado el tipo de usuario "Usuario"
        else if (rbU.isSelected()) {
            connect.SignInNewUser("Usuario",usuario,email,contraseña);
            Inicio start = new Inicio(connect);
            this.dispose();
            start.setVisible(true);
        }
```

2. En el cliente procedemos a ejecutar el método seleccionado

```
/*
Descripción: Método encargado de registrar un nuevo usuario en la base de datos
Recibe: String type (tipo de usuario), String usuario (nombre que se va a registrar),
    String email (email del usuario), String contraseña (contraseña proporcionada por el usuario)
Devuelve:
Observaciones:
*/
public void SignInNewUser(String type, String usuario, String email, String contraseña){
    // Comando a ejecutar por el servidor
    out.println("SignInNewUser");
    // Sentencia que será enviada al servidor para su ejecución
    out.println("INSERT INTO "+type+" (nombre, email, contraseña) VALUES ('"
        +usuario.toUpperCase()+"',  '"+email.toUpperCase()+"', '"+contraseña.toUpperCase()+"')");
}

public int LoginAdmin(String usuario, String contraseña){
    out.println("LoginAdmin");
    out.println("SELECT idAdministrador from Administrador where (nombre = '"+usuario.toUpperCase()
        +"' or email = '"+usuario.toUpperCase()+"') and contraseña = '"+contraseña.toUpperCase()+"'");
    try{
        String s = in.readLine();
        System.out.println(s);
        if (!s.equalsIgnoreCase("Failed")) {
            return Integer.parseInt(s);
        }
        else
            return -1;
    }
    catch(IOException e){
        System.err.println(e);
        return -1;
    }
}
```

3. El servidor leerá el comando y procederá a su ejecución por medio de la clase que interactúa con la Base de Datos.

```
// ITERACIÓN INFINITA PARA INTERCAMBIO DE MENSAJES CON EL CLIENTE
while(true){
    // Se lee la sentencia de entrada
    theInputLine = in.readLine();

    // Comparamos el comando para decidir cómo ejecutar la acción
    if (theInputLine.equalsIgnoreCase("SignInNewUser")) {
        // Leemos la sentencia a ejecutar
        query = in.readLine();
        // Ejecutamos la sentencia en la base de datos
        connection.RegisterNewUser(query);
    }
```

4. Por medio de la clase que interactúa con la Base de Datos procederemos a ejecutar el comando de Inserción.

```
/*
Descripción: Método encargado de registrar nuevos usuarios en la Base de Datos
Recibe: String sqlInsert (Sentencia que se va a insertar en la Base de Datos)
Devuelve:
Observaciones: Se coloca dentro de un bloque try porque puede generar excepciones
*/
public static void RegisterNewUser(String sqlInsert){
    try{
        int i; // Variable que nos dirá el número de tuplas afectadas
        Statement statement=conex.createStatement(); // Variable que permite ejecutar cambios en la BD
        i=statement.executeUpdate(sqlInsert); // executeUpdate regresa como entero el número de tuplas afectadas
        System.out.println("QUERY OK, "+i+" row affected.");
    } catch (Exception ex){
        System.out.println(ex);
        System.exit(0);
    }
}
```