

Proyecto GUI con acceso a datos

2º DAM -B

Módulo de AED - DAD

Jonathan Briones Castilla

Mario S. Pinto Miranda

Índice de contenidos

1. Descripción de la Actividad.
2. Comentarios de la Actividad.
3. Código y construcción.

1) Descripción de la Actividad.

Los alumnos desarrollarán por parejas y durante las próximas dos semanas la interfaz gráfica de una aplicación con Java Swing, usando las librerías que consideren oportunas. La temática es libre y deberá cubrir tanto los requisitos proporcionados en el módulo AED, como los de este módulo, DAD, que se detallan a continuación:

- JFrame principal.
- Al menos 2 JPanel.
- JTabbedPane que nos permita mostrar o desactivar ambos paneles.
- Usar al menos 2 gestores de composición o posicionamiento (Layout Manager).
- Para la recogida de datos: JLabel, JTextField, JTextArea con JScrollPane, JPasswordField, JCheckbox, JRadioButton, JComboBox, JFileChooser, JButton.
- Cuadros de diálogo con JOptionPane.
- Fijar un Icono y título a la ventana.
- Deben implementarse eventos, entre ellos:

Un botón que nos permita “limpiar” todos los componentes y dejarlos con su valor por defecto.

- Implementar en un JTextField un campo email del usuario, validar el email cuando el JTextField pierda el foco (FocusListener), si es incorrecto mostrar en un JOptionPane de alerta “Debes introducir un email correcto”, luego debes vaciar el contenido del JTextField y volver a poner el foco en el JTextField del email hasta que se introduzca un valor correcto.
- Deben implementarse dos eventos más de libre elección.

- Con los datos insertados por el usuario o los usuarios, debe mostrarse un gráfico de barras o de tarta (circular). Debes acceder a las opciones del gráfico y personalizarlo como el color del gráfico, tipo de letra,....

Debes usar la librería JFreeChart para mostrar el gráfico y para personalizarlo Renderer y Plot de JfreeChart.

Evaluación:

Para la evaluación del proyecto se tendrán en cuenta los siguientes elementos:

- Código de la aplicación desarrollada y su funcionamiento. (se valorará la claridad del código, indentación, comentarios, eficiencia) (5 pts)
- Memoria que recoja toda la información y documentación relacionada con el proceso de desarrollo de la aplicación. (2,5 puntos)

- Documento PowerPoint utilizado para presentar la aplicación. (1 pto)
- Presentación de 5-10 minutos sobre la aplicación (expresión, soltura, seguridad, reparto de tiempo,...). (1,5 puntos)

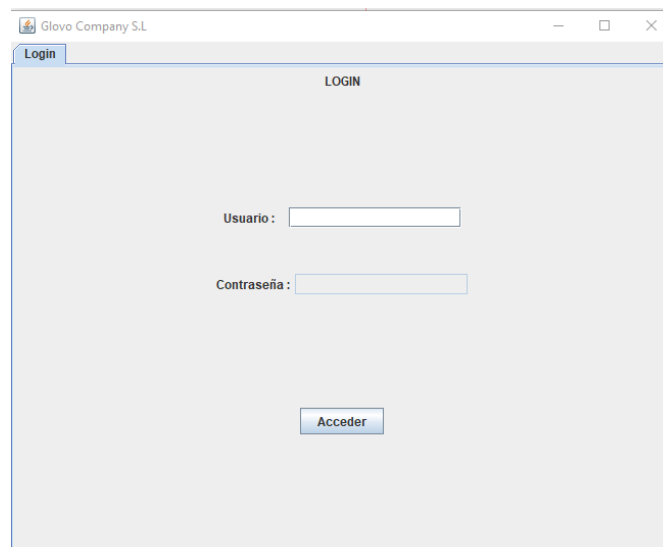
2) Comentarios de la Actividad

La actividad a realizar será un programa para una empresa de mensajería.

Login

La idea es un programa que podría estar en una central de reparto. Tendría 4 tipos de usuarios: cliente, repartidor, administrativos y “admin”.

Los usuarios podrán acceder al programa usando el correo y una contraseña. Luego se les derivará a la pestaña del usuario que tenga los permisos de dicho usuario.



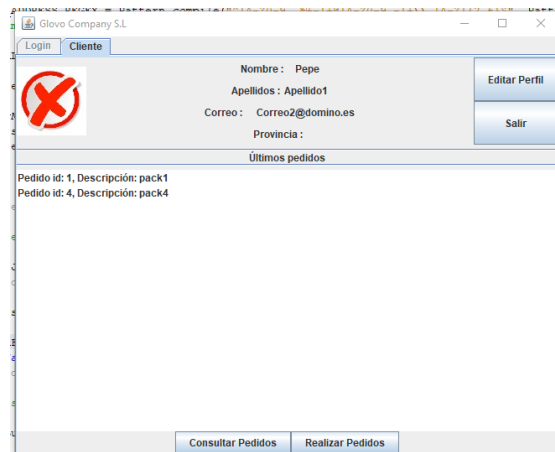
The screenshot shows a web browser window with the title 'Glovo Company S.L.' and a 'Login' tab. The main content area is titled 'LOGIN'. It features two input fields: 'Usuario:' and 'Contraseña:'. Below these fields is a button labeled 'Acceder'.

La pestaña de acceso tiene un evento de focus que controla que el usuario introduce un correo para poder logearse. Una vez lo introduce se habilita el campo de contraseña.

Cuando pulsas acceder se realiza una consulta a la base de datos y se verifica los campos de usuario y contraseña. Si son validos se le redirige a la pestaña del usuario si no se le informa que no son validos.

Usuario cliente:

Cuando un usuario de tipo “cliente” accede se le redirige a la pestaña cliente.



El panel principal tendrá dos partes: una superior con los datos del usuario y otra central con la lista de pedidos realizados por el usuario.

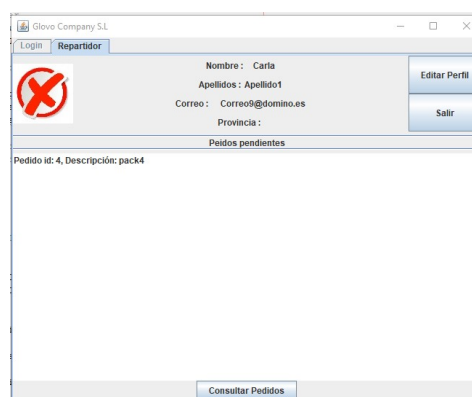
La parte superior tendría una foto del usuario (actualmente no esta disponible el almacenamiento de ficheros, solo esta implementado como una imagen), junto con los datos del usuario y dos botones.

El primer botón es para abrir un formulario con los datos y poder editarlos. El segundo botón es para salir, lo que devolvería al login.

En la parte central tendremos una lista con los últimos pedidos realizados por el cliente.

La parte inferior tenemos dos botones que permiten o consultar los pedidos o crear uno nuevo.

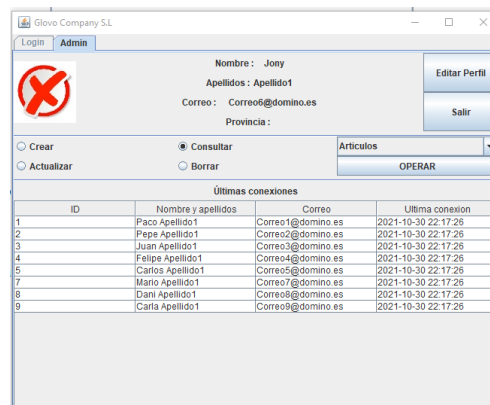
Usuario repartidor:



Para el usuario repartidor es parecido al del cliente. El repartidor puede consultar sus pedidos asignados.

Usuario admin

El admin tiene una cabecera como el resto de usuarios. Sin embargo la parte central tiene un menú de operaciones. En la parte inferior tiene una tabla con los últimos usuarios conectados.

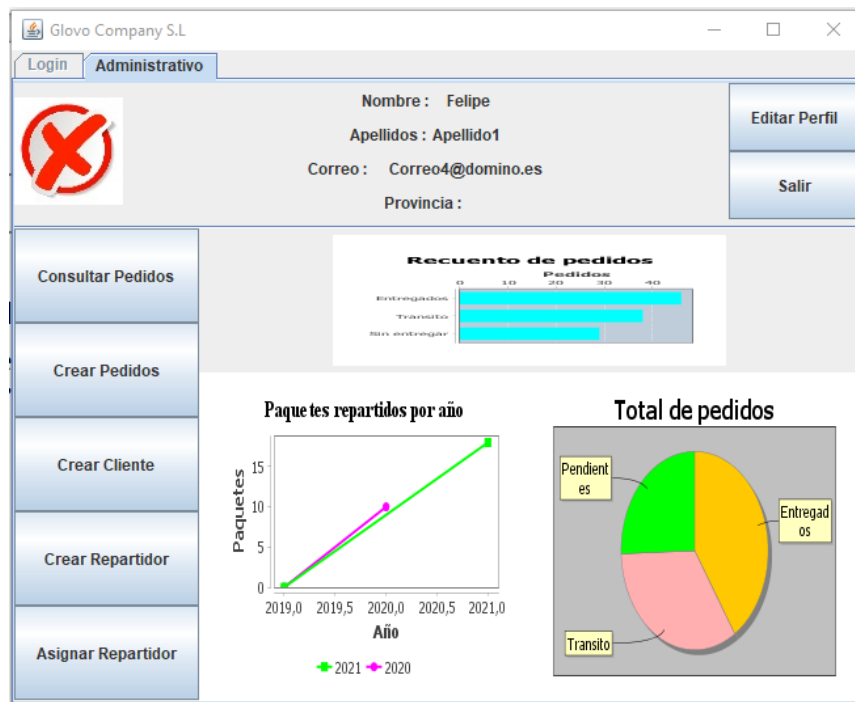


The screenshot shows a web application window titled 'Glovo Company S.L.' with a 'Login' button and an 'Admin' tab. The user profile section displays the name 'Jony', the last name 'Apellido1', the email 'Correo6@domino.es', and the province. There are buttons for 'Editar Perfil' and 'Salir'. Below the profile, there are radio buttons for 'Crear', 'Actualizar', 'Consultar' (selected), and 'Borrar'. A dropdown menu for 'Articulos' is set to 'OPERAR'. At the bottom, there is a table titled 'Últimas conexiones' with columns for ID, Nombre y apellidos, Correo, and Última conexión.

ID	Nombre y apellidos	Correo	Última conexión
1	Paco Apellido1	Correo1@domino.es	2021-10-30 22:17:26
2	Paco Apellido1	Correo2@domino.es	2021-10-30 22:17:26
3	Juan Apellido1	Correo3@domino.es	2021-10-30 22:17:26
4	Felipe Apellido1	Correo4@domino.es	2021-10-30 22:17:26
5	Carlos Apellido1	Correo5@domino.es	2021-10-30 22:17:26
7	Mario Apellido1	Correo7@domino.es	2021-10-30 22:17:26
8	Dani Apellido1	Correo8@domino.es	2021-10-30 22:17:26
9	Carla Apellido1	Correo9@domino.es	2021-10-30 22:17:26

Usuario administrativo:

Este usuario sería el responsable que se encuentra en las oficinas (algo así como el administrativo que “organiza” y atiende a los clientes).



Al igual que el resto de usuarios, el administrativo tiene sus datos en la parte superior.

Debido a que puede tratar con los clientes, el usuario tiene distintas opciones a la hora de realizar operaciones (crear nuevos clientes, repartidores, etc) y una serie de gráficas para mostrar información sobre los pedidos.

Además de las pestañas de los usuarios existen otros dos Jframe que se enlazan con las pestañas.

Tabla_dialog:

Este frame es el que se usa para cargar datos de forma en tabla (por ejemplo consultar los pedidos).

El frame tiene un pequeño “buscador” para insertar los datos y luego dos botones: limpiar (que limpia) y buscar (que abre el otro frame con los datos cargados).

En la parte inferior esta la tabla con los datos de la consulta solicitada (por ejemplo, si consultamos los pedidos la tabla se carga con los pedidos mostrando los distintos campos).

La tabla tiene una escucha que permite pinchar en la misma tabla y cargará los datos en los campos del buscador.

Seleccione un elemento de la tabla o use el buscador

Cliente Repartidor

Lugar de Entr... Articulo

Cliente	Repartidor	Lugar de Entre...	Articulo	Nº articulos
Pepe	Sin asignar	Santa Cruz de ...	pack1	12
Juan	Sin asignar	Jaén	pack2	2
Felipe	Dani	Madrid	pack3	1
Pepe	Carla	Burgos	pack4	20
Juan	Dani	A Coruña	pack5	4
Felipe	Carla	Álava	pack6	11

Hay que destacar que dependiendo que tipo de usuario accede a este JFrame, puede que hayan campos desactivados o simplemente el botón de buscar desactivado.

Formulario_dilog

Este frame se usa para editar, mostrar información sobre un registro (por ejemplo ver un pedido) o crear un nuevo registro.

Editar perfil

apellidos :

id_usuario :

correo :

nombre :

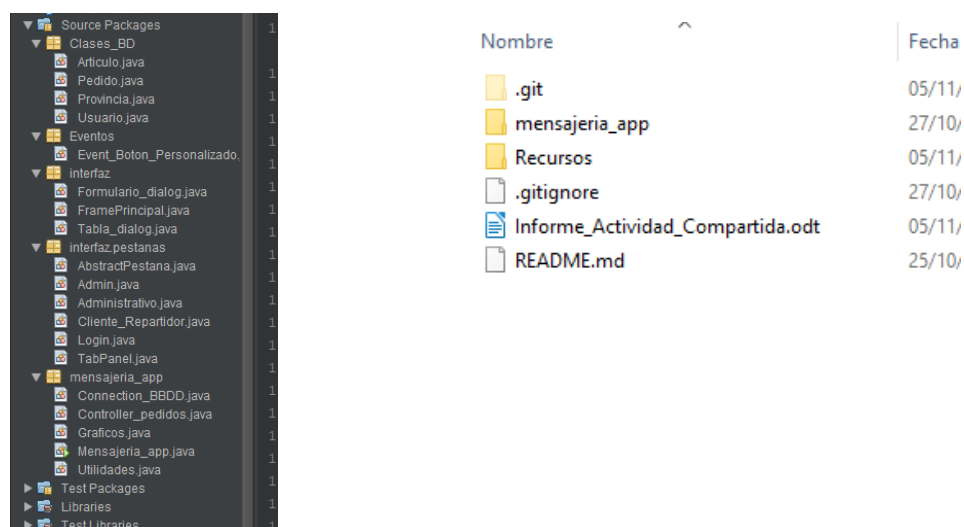
id_provincia :

Este frame simplemente es un formulario con una serie de campos que se auto-generan a partir de los datos de la tabla. Contiene 2 botones: cancelar o realizar la operación.

3) Código y construcción.

IMPORTANTE: Las capturas mostradas no están actualizadas del todo, fueron tomadas durante el proceso final de desarrollo. Los iconos ya han sido cambiados y están personalizados, por ejemplo.

En esta sección vamos a comentar como se construyeron las distintas secciones y el código que hay detrás. Lo primero es ver como esta organizado el proyecto (como proyecto y como carpetas):



Contenido de la carpeta recursos:

Nombre	Fecha de modificación	Tipo	Tamaño
db_mensajeria_AEDAD.sql	02/11/2021 15:48	SQL Text File	4 KB
error.jpg	27/10/2021 16:04	Archivo JPG	5 KB
generar_cosas.sql	02/11/2021 15:28	SQL Text File	3 KB
jcommon-1.0.23.jar	28/10/2021 14:27	Executable Jar File	323 KB
jfreechart-1.0.19.jar	28/10/2021 14:27	Executable Jar File	1.534 KB
Modelo relacional de Mensajeria.mwb	29/10/2021 16:44	MySQL Workbenc...	10 KB
mysql-connector-java-8.0.27.jar	25/10/2021 16:03	Executable Jar File	2.418 KB
Presentacion proyecto.odp	03/11/2021 18:17	Presentación Ope...	845 KB

Como se puede ver en las imágenes, la estructura que usamos para montar el proyecto usaba GIT a la hora de guardar la información (así solo tenemos que actualizar la rama y el proyecto podríamos llevarlo a cualquier sitio). Además el proyecto tiene una carpeta “externa” que contiene la imagen de “prueba” para las fotos o el icono, las librerías del conector y los gráficos, la presentación, la construcción de la base de datos (es un script SQL), el modelo relacional y un script para meter algunos datos dentro de la base de datos (para pruebas y demás).

A parte del GIT (que se ve en la imagen y seguramente no estará en la entrega de la actividad), esta este mismo informe/memoria del proyecto.

Ahora entrando a el código, la estructura de archivos (la organización) que hemos usado separa los componentes de la manera más intuitiva:

- Las clases de la base de datos, que hacen de “respaldo” para usarlas como contenedores a la hora de manejar los datos.
- Los eventos, que es el evento que se usa como clic en las pestañas
- La interfaz que son los JFrame que hay.
- Interfaz de pestañas, que son las pestañas que usamos junto con el Jtabpanel.
- Mensajería_app, es todo lo que no incluya la parte del JFrame: la parte del conector, la clase utilidades, los gráficos, etc.

Faltaría un paquete más que seria el de recursos pero como lo tenemos en la carpeta exterior al programa no lo incluimos dentro del mismo (sería para las cosas externas al código, como imágenes y archivos, etc).

Una vez explicado la forma de organizar el proyecto tenemos que hablar de la lógica del programa: como la idea es una empresa de reparto, tenemos que tener distintos usuarios que puedan interaccionar con él. Los usuario estarían divididos en función de los permisos (**cliente, repartidor, administrativo y administrador**).

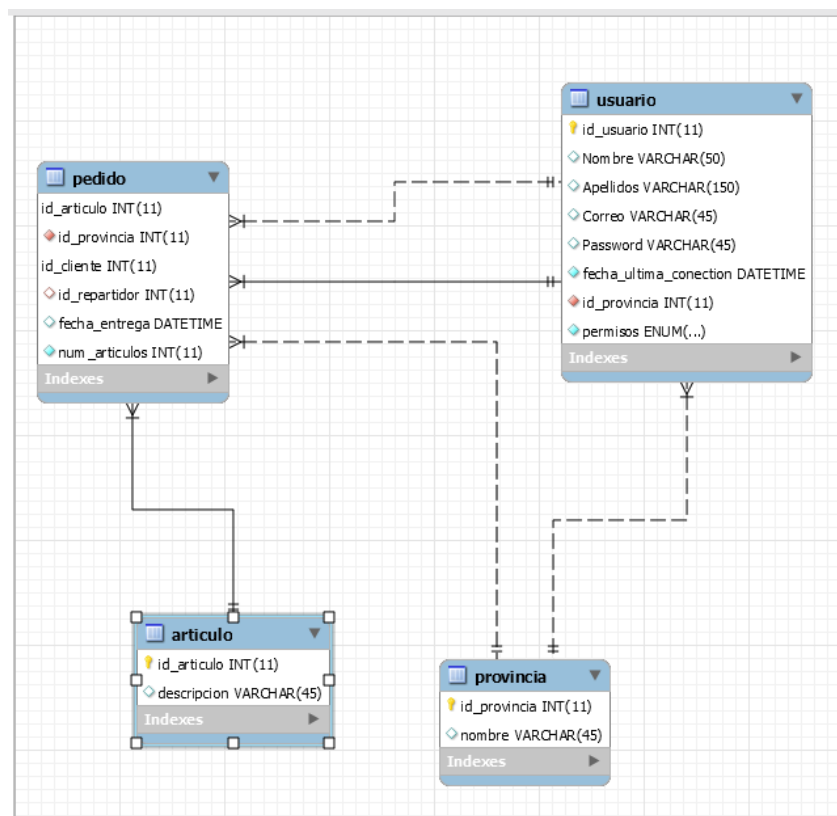
Todos tendrían un “login” inicial y en función del permiso se le redirige a una pestaña u otra. Inicialmente las pestañas iban a estar todas construidas pero deshabilitadas. Con una posterior revisión se decidió que se construyen cuando el usuario acceda.

En resumen, el programa arranca y crea el JFrame. Este llama al Jtabpanel y crea la pestaña de *login*. Cuando el usuario accede, el Jtabpanel crea una pestaña, que varia en función del permiso del usuario (en este punto se acaba de crear la pestaña del usuario). Si el usuario desloguea, la pestaña se destruye y se redirige a la pestaña de *login* otra vez.

Una vez hemos visto la lógica, toca empezar hablar de la parte del código: para esto iremos archivo por archivo (las clases de DB las comentaremos de manera global porque son estructuras iguales y parecidas a las de la tablas).

El paquete de “clases_db” contiene las clases que hacen referencia a la base de datos, la estructura es idéntica a la de las tablas. Tienen un constructor y getter/setter para los atributos poco más.

Como no tienen mucho de que hablar aprovecharemos y explicamos un poco la relación que tienen las tablas y su cardinalidad.



Los atributos de la clases son las mismas que las tablas.

Tenemos una tabla articulo, usuario y provincias. Como usuario y articulo tienen una relación de tipo n:m esto genera una tabla (la típica “n:m” que es una relación de mucho a muchos). Es decir, un usuario puede tener muchos artículos y, a la vez, un artículo puede tener más de un usuario.

De esta relación nace la tabla de pedidos. Además pedidos puede tener un repartidor (es 1:n, nulleable). La relación de provincias siempre es 1:n (1:n es que una provincia puede estar varias veces en otra tabla, por ejemplo un usuario puede tener 1 provincia pero una provincia puede estar en varios usuarios).

También hay que destacar que la tabla de provincias se añadió para “complementar” datos para las consultas y tener más “candidatos” para hacer las peticiones (vamos que es una tabla complementaria).

Quizás lo más interesante de las tablas es que el usuario no tiene una tabla permisos, sino que este atributo está controlado por un enumerable: al final solo tenemos cuatro permisos (que son los distintos tipos de usuarios) y no es necesario crear una tabla para 4 registros si podemos controlarlos de otra forma.

Volviendo al código, las clases son bastante simples:

```
public class Provincia {
    private int id_provincia;
    private String nombre;

    // constructor
    public Provincia(int id_provincia, String nombre) {
        this.id_provincia = id_provincia;
        this.nombre = nombre;
    }

    public Provincia() {
    }

    // getter y setter
    public int getId_provincia() {
        return id_provincia;
    }

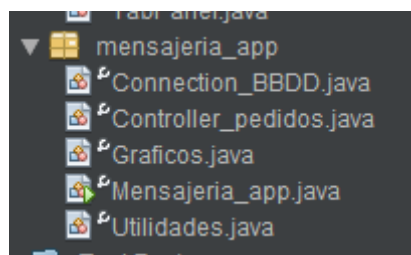
    public String getNombre() {
        return nombre;
    }

    public void setId_provincia(int id_provincia) {
        this.id_provincia = id_provincia;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

Tienen los atributos de las tablas junto con constructores y getter/setter, nada más.

A continuación vamos a hablar de la parte de mensajería_app (las partes gráficas las dejaremos para el final que son bastantes más densas).



Mensajería se podría separar en 3 “elementos”: manipulador de datos, “librerías” y extra.

La parte del manipulador estaría compuesta por Controller_pedidos. Las librerías serían gráficos y connection_BBDD. Extra es mensajería_app (es el main) y utilidades.

Extra

Aquí está el main y utilidades:

- Main (Mensajería_app) funciona como cargador del JFrame, no tiene más ciencia.

```
public class Mensajeria_app {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws SQLException {
        // el cargador
        new FramePrincipal();
    }

}
```

- Utilidades es una clase en la que ponemos métodos comunes (y normalmente genéricos) que usamos por algunas partes del proyecto. Todos son métodos estáticos.

```
public static void centrarPantalla(JFrame elFrame){
    Toolkit pantalla = Toolkit.getDefaultToolkit();
    Dimension tamanoPantalla = pantalla.getScreenSize();

    int height = tamanoPantalla.height - elFrame.getHeight()/2;
    int width = tamanoPantalla.width - elFrame.getWidth()/2;

    elFrame.pack();

    elFrame.setLocation(width/2 - elFrame.getWidth()/2 , height/2 - elFrame.getHeight()/2);
}

public static DefaultTableModel ArrayList_to_DefaultTableModel(ArrayList<String[]> lista_Datos) {
    DefaultTableModel tableModel = new DefaultTableModel(lista_Datos.get(0), 0);

    for (int i = 1; i < lista_Datos.size(); i++) {
        Object[] objs = lista_Datos.get(i);
        tableModel.addRow(objs);
    }

    for(String[] rowAdd : lista_Datos){
        Object[] objs = rowAdd;
        tableModel.addRow(objs);
    }

    return tableModel;
}
```

Un ejemplo son estos dos métodos. El primero, cuando se le llama hay que pasar el JFrame, se usa para centrar el JFrame en el centro de la pantalla. Esto es algo que íbamos a usar bastante a la hora de centrar frames. El segundo método lo que hace es convertir un ArrayList de tipo String array (ArrayList<String[]>) en un tablemodel (DefaultTableModel) que se usa varias veces para construir las tablas. Es decir se le pasa el array con los nombres de la cabecera de la tabla como índice 0 y el construye el tablemodel para que pueda ser usado en cualquier Jtable.

Los otro métodos de utilidades son para formatear fechas.

Librerías

Aquí se incluyen el conector de la base de datos y gráficos.

El primero es un archivo que contiene la conexión de la base de datos de forma genérica junto con varios métodos que incluyen las consultas típicas.

El conector originalmente es el que tenía Mario ya de otras prácticas, que como era genérico podíamos usarlo aquí para simplificar un poco.

El conector guarda los datos cuando se construye.

```
private String user;
private String password;
private String ip_connection;
private String BBDD;
private Connection conexion;
private Statement query;

// conector
public Connection_BBDD(String user, String password, String ip_connection, String data_base) {

    this.user = user;
    this.password = password;
    this.ip_connection = ip_connection;
    this.BBDD = "jdbc:mysql://" + this.ip_connection + "/" + data_base;
}
```

Y tiene dos métodos principales para “abrir” y “cerrar” la conexión.

```
private void connect() {

    try {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(Connection_BBDD.class.getName()).log(Level.SEVERE, null, ex);
        }
        conexion = DriverManager.getConnection(BBDD, user, password);
        query = conexion.createStatement();
    } catch (SQLException ex) {
        Logger.getLogger(Connection_BBDD.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

```
public void close_connection() {

    try {

        if (query != null) {
            query.close();
        }

        if (conexion != null) {
            conexion.close();
        }

    } catch (SQLException ex) {
        Logger.getLogger(Connection_BBDD.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

La ventaja de ser genérico es que se podría exportar a otro proyecto o programa y seguiría funcionando.

Los siguientes métodos son de tratamiento del resultado y generación de las peticiones (queries) de la base de datos. A continuación un ejemplo de como se “genera” el filtro, es decir como construye la sección de “where” de la petición a la base de datos.

Esta manera nos permite generar cualquier “where” sin importar la petición.

Después de estos métodos tenemos las operaciones de la base de datos. Esto sería un “delete”:

```
public void delete (String tabla, String filtros){
    String sentencia = "DELETE FROM " + tabla + " ";
    sentencia = add_filters(sentencia, filtros) ;
    try {
        this.connect();
        query.execute(sentencia);
        System.out.println("Se ha borrado el registro");
    } catch (SQLException ex) {
        System.out.println(ex.toString());
    } catch (Exception e){
        System.out.println(e.toString());
    }
}
```

Como ya se comentó, los métodos son genéricos. Ahí podemos ver el método de los filtros que comentamos antes, que se usa para generar las condiciones del “where” del “delete”.

```
String sentencia = "DELETE FROM " + tabla + " ";
sentencia = add_filters(sentencia, filtros) ;
try {
```

Las operaciones incluyen todas las de un CRUD típico (crear, modificar, actualizar y borrar).

Además incluimos un método para operaciones complejas, como *queries* en las que hay varios *joins* o consultas de varias tablas.

Este método recoge directamente la consulta (raw query) y la manda a la base de datos sin tener que montar la petición.

```
public ResultSet raw_select(String query_sended){
    try {
        this.connect();
        return query.executeQuery(query_sended);
    } catch (SQLException ex) {
        System.out.println(ex.toString());
    } catch (Exception e){
        System.out.println(e.toString());
    }
    return null;
}
```

Lo último a destacar de este archivo es que las operaciones están controladas por try-catch para evitar problemas a la hora de que falle alguna de las peticiones.

El siguiente archivo es el de “gráficos”. Este archivo contiene la parte de la creación de las gráficas.

Tiene tres métodos “principales” que son las distintas gráficas que hay presentes en la pestaña de administración. La estructura de las gráficas siempre es la misma: un constructor, un “cargador de datos” y un dibujante.

El constructor lo que hace es crear un chartPanel (el panel de la gráfica) usando los dos métodos restantes.

```
public static ChartPanel grafica_Queso() throws SQLException{
    DefaultPieDataset dataset = createDataset();
    JFreeChart chart = createChart(dataset);

    cambiarColor(chart, "queso");

    ChartPanel chartPanel = new ChartPanel(chart);
    chartPanel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));
    chartPanel.setBackground(Color.white);

    return chartPanel;
}
```

El cargador de datos pide la consulta a la base de datos y prepara el “contenedor” de datos que usa la gráfica (lo devuelve al constructor).

```
private static DefaultPieDataset createDataset() throws SQLException {
    Integer[] valores;

    // ----- AQUI VA LA CONSULTA -----

    SELECT SUM(mensajeria.pedido.id_articulo) FROM mensajeria.pedido WHERE fecha_entrega IS NOT NULL;
    SELECT SUM(mensajeria.pedido.id_articulo) FROM mensajeria.pedido WHERE fecha_entrega IS NULL AND id_repartidor IS NOT NULL;
    SELECT SUM(mensajeria.pedido.id_articulo) FROM mensajeria.pedido WHERE fecha_entrega IS NULL AND id_repartidor IS NULL;
    valores = DB.get_estadistica_pedidos();

    int total = 0;
    for (int i = 0; i < valores.length; i++) {
        total = total + valores[i];
    }

    DefaultPieDataset dataset = new DefaultPieDataset();
    dataset.setValue("Entregados", Math.round(valores[0]*100/total));
    dataset.setValue("Transito", Math.round(valores[1]*100/total));
    dataset.setValue("Pendientes", Math.round(valores[2]*100/total));

    return dataset;
}
```

En la imagen podemos ver las consultas que se piden al controller_pedidos para que puedan ensamblar luego el contenedor del modelo de datos (DefaultPieDataset, en este caso).

Por último tenemos el “dibujante”, que es el responsable de dibujar la gráfica como tal.

```
private static JFreeChart createChart(DefaultPieDataset dataset) {
    JFreeChart pieChart = ChartFactory.createPieChart(
        "Total de pedidos",
        dataset,
        false, true, false);

    return pieChart;
}
```

De esta manera, tenemos un “constructor” de gráficas que se podría personalizar para cada cualquier proyecto (solo habría que modificar algunos parámetros y se personaliza a cualquier otro programa).

Las gráficas que contiene el archivo son la gráfica de barras, de líneas y el “queso” (“pie”).

Manipulador de datos

Esta sección solo tiene el controller_pedidos.

Este archivo contiene la información referente a los métodos y tratamientos de los datos. Para que se entienda, es el responsable de hacer las peticiones a al conector y “traducir” los datos a el tipo de variable que necesite el método que se lo llamó (vamos que es el puente entre el conector y el frame).

El archivo podemos diferenciarlo en 3 partes: una primera con los atributos, otra con las consultas y otra con el tratamiento de datos.

Cuando llamamos a la clase, esta se crea ensamblando un “conector” (la clase no tiene un constructor como tal).

```
public class Controller_pedidos {  
  
    ArrayList<Articulo> articulos = new ArrayList<Articulo>();  
    ArrayList<Pedido> pedidos = new ArrayList<Pedido>();  
    ArrayList<Provincia> provincias = new ArrayList<Provincia>();  
  
    private final String USER = "root";  
    private final String PASS = "";  
    private final String IP = "localhost:3306";  
    private final String DATABASE = "mensajeria";  
  
    Connection_BBDD DB = new Connection_BBDD(USER, PASS, IP, DATABASE);  
  
    private ArrayList<Usuario> get_generic_usuarios(ResultSet respuesta) thro
```

Esta es la parte de los atributos y es donde se crea el conector.

Luego las partes de consultas y tratamientos se suelen diferenciar porque sus métodos tienen “get_tabla” o to_string_tabla” como nombre del método (los get son las consultas y los to_string el tratamiento de datos).

También es cierto que hay otros métodos que no son tan fáciles de diferenciar como el *login* que contiene la consulta y hace las operaciones de manera interna.

```
public Usuario Login(String correo, String password) throws SQLException {  
    String filtros = "correo = '" + correo + "' AND password = '" + password + "'";  
    ArrayList<Usuario> usuario_loggin = get_generic_usuarios("", filtros);  
    if (!usuario_loggin.isEmpty()) {  
        return usuario_loggin.get(0);  
    } else {  
        return null;  
    }  
}
```

Sin embargo, como podemos ver el método llama a otro método para realizar la consulta a la base de datos.

Un ejemplo de tratamiento de datos puede ser:


```
public ArrayList<String[]> to_string_pedidos(ArrayList<Pedido> pedidos) {
    ArrayList<String[]> resultado = new ArrayList<String[]>();
    String column[] = {"ID", "Cliente", "Repartidor", "Lugar de Entrega", "Articulo", "Nº articulos"};
    resultado.add(column);
    for (Pedido pedido : pedidos) {
        String[] pedido_string = {pedido.getId_articulo() + "", pedido.getNombre_cliente(), pedido.getNombre_repartidor(),
            pedido.getNombre_provincia(), pedido.getNombre_articulo(), pedido.getNum_articulos() + ""};
        resultado.add(pedido_string);
    }
    return resultado;
}
```

Uno de consulta:

```
public ArrayList<String[]> lista_pedidos(int id_user, int permiso) throws SQLException {
    ResultSet respuesta;
    if (permiso == 1) {
        System.out.println("id: " + id_user);
        respuesta = DB.raw_select("SELECT art.* from articulo AS art INNER JOIN pedido AS ped ON art.id_articulo = "
            + "ped.id_articulo WHERE ped.id_repartidor = "
            + id_user + " AND ped.fecha_entrega IS NULL ORDER BY art.id_articulo ASC");
    } else {
        respuesta = DB.raw_select("SELECT art.* from mensajeria.articulo AS art INNER JOIN mensajeria.pedido AS ped "
            + "ON art.id_articulo = ped.id_articulo WHERE ped.id_cliente = " + id_user
            + " ORDER BY ped.fecha_entrega DESC LIMIT 15");
    }
    ArrayList<String[]> articulos = new ArrayList<String[]>();
    if (respuesta != null) {
        while (respuesta.next()) {
            String[] articulo = {respuesta.getString("id_articulo"), respuesta.getString("descripcion")};
            articulos.add(articulo);
        }
        DB.close_connection();
    } else {
        respuesta = null;
    }
    return articulos;
}
```

Ambos métodos son bastante explicativos: el primero coge los datos y los convierte en un `ArrayList<String[]>` (por ejemplo para una de las tablas) y el otro método lo que hace es hacer una petición de datos devuelve la lista de pedidos (es la consulta de pedidos de cliente-repartidor, siendo el repartidor el permiso 1).

Un ejemplo de las graficas (es la gráfica de rayas):

```
public double get_pedidos_periodo(String fecha_Last_1Enero, String fecha_Last_31Diciembre) throws SQLException {
    ResultSet respuesta = DB.raw_select("SELECT sum(mensajeria.pedido.id_articulo) as 'sum_art' "
        + "FROM mensajeria.pedido WHERE mensajeria.pedido.fecha_entrega BETWEEN '"
        + fecha_Last_1Enero + "' AND '" + fecha_Last_31Diciembre + "'");
    Double valor = 0.0;
    if (respuesta != null) {
        if (respuesta.next()) {
            try {
                valor = Double.parseDouble(respuesta.getString("sum_art"));
            } catch (Exception e) {
                valor = 0.0;
            }
        }
    }
    DB.close_connection();
    return valor;
}
```

Lo importante de este archivo es que es donde estás las operaciones de “personalización” para las consultas y pedir datos desde los frames/gráficas.

Una vez hemos terminado con la parte de “backend” podemos empezar con la parte “gráfica” (el frontend). Tenemos 3 JFrame con son la parte visual del programa.

Como se comentó anteriormente, la separación de componentes que usamos intenta ser lo más intuitiva, por eso, el frame y el tabPanel, que vamos a ver a continuación prácticamente tiene lo justo para trabajar.

El frame cuando se crea llama al tabpanel.

```
public class FramePrincipal extends JFrame {  
  
    public FramePrincipal() throws SQLException {  
  
        setTitle("Glovo Company S.L");  
        setDefaultCloseOperation(EXIT_ON_CLOSE);  
  
        ImageIcon img = new ImageIcon("../Recursos/error.jpg");  
        setIconImage(img.getImage());  
  
        // el tabpanel  
        add(new TabPanel());  
  
        // centramos  
        mensajeria_app.Utilidades.centrarPantalla(this);  
  
        setVisible(true);  
    }  
}
```

Y este a su vez crea la pestaña de *login*.

```

public class TabPanel extends JTabbedPane {
    private int id_usuario;
    private Usuario Usuario_logueado;

    public Usuario getUsuario_logueado() {
        return Usuario_logueado;
    }

    public void setUsuario_logueado(Usuario Usuario_logueado) {
        this.Usuario_logueado = Usuario_logueado;
    }

    public int getId_usuario() {
        return id_usuario;
    }

    public void setId_usuario(int id_usuario) {
        this.id_usuario = id_usuario;
    }

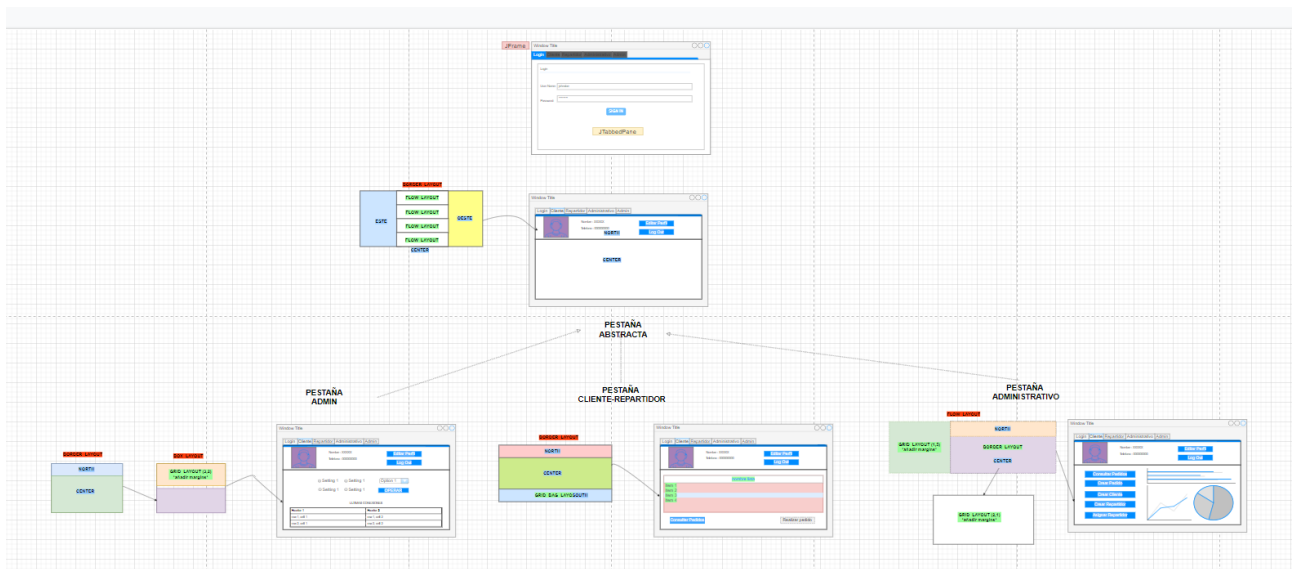
    public TabPanel() throws SQLException {
        // crea login
        addTab("Login", new Login(this));

        id_usuario = 0;

        setPreferredSize(new Dimension(650,500));
    }
}

```

Antes de continuar vamos a hablar un poco de las pestañas.



Esta imagen resume un poco como está formado el conjunto de pestañas (ya se comentó anteriormente).

La pestaña de login es la inicial, que como se a dicho antes, es creada por el tabpanel.

Esta pestaña es un panel con un boxLayout con axis en “Y”. Dentro tiene tres contenedores, Jpanel de tipo FlowLayout, con sus componentes.

El campo (JTextField) del usuario tiene un evento de pérdida de foco que verifica si el usuario tiene formato de correo (es el formato que usamos para los usuarios). Si lo tiene, habilita el campo de contraseña. Si no lo tiene, salta un JOptionPane que nos informa que “*debes introducir un email correcto*”, reiniciando el campo y manteniendo el campo contraseña desactivado.

```
campo_user_name.addFocusListener(new FocusListener() {
    @Override
    public void focusGained(FocusEvent e) {
    }

    @Override
    public void focusLost(FocusEvent e) {
        // patron de correo
        Pattern VALID_EMAIL_ADDRESS_REGEX = Pattern.compile("^[A-Z0-9._%+-]+@[A-Z0-9.-]+\\.[A-Z]{2,6}$", Pattern.CASE_INSENSITIVE);
        String emailStr = campo_user_name.getText();
        // verificamos
        Matcher matcher = VALID_EMAIL_ADDRESS_REGEX.matcher(emailStr.trim());
        if(matcher.find()){
            campo_password.setEditable(true);
        } else{
            JOptionPane.showMessageDialog(null, "Debes introducir un email correcto.", "Error", JOptionPane.ERROR_MESSAGE);
            campo_user_name.setText(" ");
            campo_password.setEditable(false);
        }
    }
});
```

El otro evento es el del botón, que verifica si el usuario y contraseña son correctos. Si es válido, se crea una pestaña con el tipo de usuario validado (recordar que solo se a creado la pestaña de login y es aquí cuando el programa crea las demás). De lo contrario se informa que el usuario no existe.

```
{
    String usuarioTrim = campo_user_name.getText();
    Usuario usuario_login = DB.Login(usuarioTrim.trim(), campo_password.getText());
    if (usuario_login != null ) {
        tab.setEnabledAt(0, false);
        int pestana_redirigida = 1;
        System.out.println(usuario_login.getPermisos());
        switch (usuario_login.getPermisos()) {
            case 0:
                tab.addTab("Cliente", new Cliente_Repartidor(0, tab, usuario_login));
                break;
            case 1:
                tab.addTab("Repartidor", new Cliente_Repartidor(1, tab, usuario_login));
                break;
            case 2:
                tab.addTab("Administrativo", new Administrativo(2, tab, usuario_login));
                break;
            case 3:
                tab.addTab("Admin", new Admin(3, tab, usuario_login));
                break;
        }
        tab.setSelectedIndex(pestana_redirigida);
        tab.setUsuario_logueado(usuario_login);
    } else {
        JOptionPane.showMessageDialog(null, "No se ha encontrado al usuario");
    }
} catch (SQLException ex) {
```

Antes de continuar con las pestañas vamos a hablar de los archivos “abstractos” o padres.

Hay dos en el proyecto: un evento y un clase. Ambos trabajan como como padres para las pestañas que se crean a continuación.

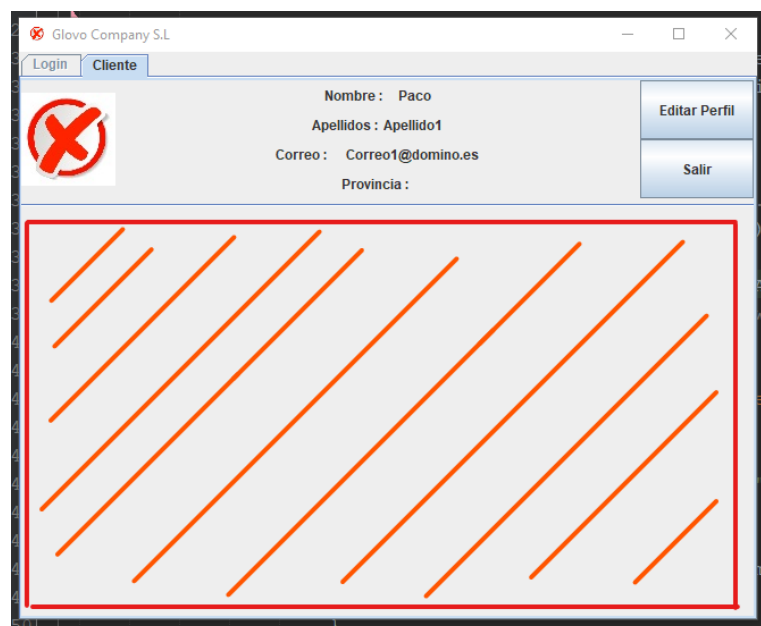
El evento es el que se va a usar para lo botones (de ahí que creamos un evento “padre”).

```
public class Event_Boton_Personalizado implements ActionListener {  
    // solo se usa la propiedad de source antes tenia más atributos pero se simplifico  
    protected JButton source;  
  
    // constructor  
    public Event_Boton_Personalizado() {  
    }  
    // getter / setter  
    public JButton getSource() {  
        return source;  
    }  
  
    public void setSource(JButton source) {  
        this.source = source;  
    }  
    // obligamos que tenga un action  
    @Override  
    public void actionPerformed(ActionEvent ae) {  
    }  
}
```

Como se puede ver tiene un atributo protegido (que usaremos como “source” del evento), constructor, getter, setter y el evento de del actionPerformed (que dejaremos vacío para que lo sobrescriba cada hijo).

Por otro lado tenemos una clase padre que no usaremos directamente sino que las pestañas usaran esta clase para heredar sus cosas y personalizar el resto (vamos que nos interesa crear un marco “común” que hereden los hijos para luego personalizar el centro).

Si cargamos al padre veríamos esto:



La parte coloreada (la zona centra con las rayas naranjas) es la zona que personalizan los hijos (de ahí que primero hemos hablado del padre).

Entrando en código, el padre esta formado por un panel con BorderLayout que en el norte tiene la parte del usuario y en el centro tiene un JPanel (BorderLayout) con el norte ocupado por un separador (JSeparator) y en el centro, el contenido “libre” del hijo.

En la región norte tenemos un layout de tipo BorderLayout. La sección “oeste” es para una foto de usuario (JLabel para la imagen, no había requisito de implementación de archivos, pero se preparó). En la “este” están los botones (JPanel con GridLayout (2,1)) de “*Editar Perfil*” y “*Salir*”. En el centro esta los datos del usuario (JPanel con BoxLayout, y dentro cada par de componentes, está un panel con FlowLayout).

Ya tenemos una idea visual vamos al código. Tiene distintos atributos que heredan los hijos (algunos son protegidos y otros no), un constructor (es quien monta la cabecera y el cuerpo central), un método “carga de datos” (que es quien prepara la consulta y da los datos a los JTextField) y por último el evento del Logout.

El cargador de datos solo hace la consulta y coloca los datos:

```
public void cargaDatosPermisos(Usuario usuario) throws SQLException{
    // cargador de datos

    /* ----- CARGAR DATOS AQUI ----- */
    // AQUI SERÍA SOLO UN USUARIO Y EL NOMBRE DE LA PROVINCIA -> PUEDES INSERTARLO ANTES DE TRAERLO

    //SELECT usu.*, pro.nombre FROM mensajeria.usuario AS usu JOIN mensajeria.provincia AS pro ON usu.id_provincia = pro.id_provincia W

    // ejemplo

    campo_user_name.setText(usuario.getNombre());
    campo_user_apellido.setText(usuario.getApellidos());
    campo_user_mail.setText(usuario.getCorreo());

    //campo_user_city.setText(DB.get_provincia_by_id(usuario.getId_provincia()));
}
```

Y es llamado al final del constructor (vamos que se cargan los datos una vez ha terminado de preparar la cabecera).

Por último el evento de salir es el que nos manda a la pestaña de login otra vez y destruye las pestañas que hubiera creadas.

```
private class click_Logout implements ActionListener{
    private final int pestana;

    public click_Logout(int pestana) {
        this.pestana = pestana;
    }

    @Override
    public void actionPerformed(ActionEvent ae) {
        tab.setEnabledAt(1, false);
        tab.setEnabledAt(0, true);
        tab.setSelectedIndex(0);
        tab.remove(1);
    }
}
```

El evento de editar usa el evento “padre” que se comentamos antes y son los hijos quienes los personalizan.

Ahora si vamos con las pestañas.

Como vimos anteriormente, el login redirige a una pestaña definida pro el permiso del usuario. Empezaremos con el **cliente-repartidor**.

En la parte superior de este informe ya vimos la parte visual de estas pestañas. Pero en realidad las pestañas solo son una. El cliente y el repartidor comparten la misma pestaña pero alterando un par de parámetros (**el cliente ve sus pedidos y puede crear pedidos, el repartidor ve los pedidos asignados a él y no puede crear pedidos**).

La pestaña consta de unos atributos (los paneles), un constructor, y un evento.

El constructor lo primero que hace es llamar al padre para que cree el marco y luego continua el creando los elementos.

```
public Cliente_Repartidor(int permiso, TabPanel tab, Usuario usuario) throws SQLException {
    /* cosas padre */
    super(permiso, tab, usuario);
    boton_Edit.addActionListener(new click_operar());

    // esto es el "body" que cambia en los hijos
    panelCentro.setLayout(new BorderLayout());

    // los paneles
    panelCentralNorte = new JPanel();
    panelCentralNorte.setLayout(new BorderLayout());
    panelCentro.add(panelCentralNorte, BorderLayout.NORTH);
}
```

Aquí también se encuentra la sección que carga los datos (la consulta a la base de datos pasando por el controller_pedidos).

```
panelSeparator.add(panelLabelTituloLista, BorderLayout.CENTER);
panelCentralNorte.add(panelSeparator);

// ----- centro -----
JList listaPedidos = new JList<>();

lista_datos = new ArrayList<String[]>();
/* ----- CARGAR DATOS AQUI ----- */
// EN AMBOS CASOS DEBERIAMOS TENER SOLO UN ArrayList<String[]> CON ARTICULOS

lista_datos = DB.lista_pedidos(usuario.getId_usuario(), super.permisoUser);

System.out.println(lista_datos);

DefaultListModel modelo = new DefaultListModel();
// id articulo
// nombre articulo
for (String[] dat : lista_datos) {
    modelo.addElement("Pedido id: " + dat[0] + ", Descripción: " + dat[1]);
}
```

El elemento principal es el Jlist, que está encima de los botones (se ve en la imagen de la carga de datos).

En caso de que sea cliente se le crea los dos botones (repartidor solo tiene 1).

```
// ----- sur -----
JButton botonConsultarPedido = new JButton();
botonConsultarPedido.setText("Consultar Pedidos");
botonConsultarPedido.setName("Select_pedidos");
botonConsultarPedido.addActionListener(new click_operar()); // evento
panelCentralSur.add(botonConsultarPedido);

// solo si es cliente
if (permisoUser == 0) {
    JButton botonCrearPedido = new JButton();
    botonCrearPedido.setText("Realizar Pedidos");
    botonCrearPedido.setName("New_Pedido");
    botonCrearPedido.addActionListener(new click_operar()); // evento
    panelCentralSur.add(botonCrearPedido); // evento editar
}
```

Si nos fijamos veremos que todos los botones tienen una escucha en común, que es el evento padre que comentamos arriba.

El método es bastante extenso pero lo que hace es “ver” quien es el “source” (el botón que ha llamado) y actúa en función de esto.

Un ejemplo:

```
private class click_operar extends Eventos.Event_Boton_Personalizado {

    @Override
    public void actionPerformed(ActionEvent ae) {
        this.source = (JButton) ae.getSource();

        switch (source.getName()) {
            case "Editar_Perfil": {
                try {
                    // edit -> modal formulario: edit tabla usuario, id_usuario, permisos
                    new Formulario_dialog("Editar perfil", 3, 1, id_user, permisoUser);
                } catch (SQLException ex) {
                    Logger.getLogger(Cliente_Repartidor.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
            break;
            case "New_Pedido": {
                try {
                    // new edit -> modal formulario: edit tabla pedidos, 0, permisos
                    new Formulario_dialog("Nuevo pedido", 1, 1, 0, getPermisoUser());
                } catch (SQLException ex) {
                    Logger.getLogger(Cliente_Repartidor.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }
    }
}
```

Cuando pinchas en editar perfil se genera un nuevo frame con el formulario (la nomenclatura la veremos con los Jframes modales).

El siguiente usuario sería el **administrador** (dejamos para luego el administrativo).

La parte centra de este usuario esta dividida por un BorderLayout. En el norte, un GridLayout con las opciones de radioButton, un comboBox y el botón de operar (el evento lo vemos luego).

Y en el centro tiene un Jtable con las ultimas conexiones de los usuarios.

El constructor, igual que antes, llama al padre.

```
public Admin(int permiso, TabPanel tab, Usuario usuario) throws SQLException {
    /* cosas padre */
    super(permiso, tab, usuario);
    boton_Edit.addActionListener(new click_operar());
    boton_Logout.addActionListener(new click_Logout());

    panelCentro.setLayout(new BoxLayout(panelCentro, BoxLayout.Y_AXIS));

    //panel operaciones
    JPanel panelComandos = new JPanel();
    panelComandos.setLayout(new GridLayout(2, 3));
    panelCentro.add(panelComandos);

    //panel tabla datos
    JPanel panelTabla = new JPanel();
    panelTabla.setLayout(new BorderLayout());
    panelCentro.add(panelTabla);

    //----- operaciones -----
}
```

Prepara los elementos y carga los datos dentro de la tabla.

El botón de operar y el de editar está controlado por mismo evento (igual que pasaba con el cliente-repartidor). El evento mira la source y, dependiendo quien sea, realiza una operación u otra.


```

public void actionPerformed(ActionEvent ae) {
    this.source = (JButton) ae.getSource();

    if("Editar_Perfil".equals(source.getName())){
        try {
            // edit -> formulario edit id usuario
            new Formulario_dialog("Editar perfil", 3, 1, id_user, permisoUser);
        } catch (SQLException ex) {
            Logger.getLogger(Admin.class.getName()).log(Level.SEVERE, null, ex);
        }
    } else {
        int opcion_Marcada = 1; // para simplificar por defecto edit
        boolean nuevo = false; // para controlar el nuevo registro

        if(radioButton_New.isSelected()){
            // New edit -> formulario edit 0
            nuevo = true;
        } else if(radioButton_Select.isSelected()){
            //select -> modal tabla modo 0
            opcion_Marcada = 0;
        } else if(radioButton_Delete.isSelected()){
            // delete -> modal tabla: modo 2

```

En este caso es un if el que controla la operación. Si es editar abre un formulario y permite editar el perfil del usuario. Si no (else), recoge que valores se han seleccionado y se redirige a donde toca.

```

        } else {
            int opcion_Marcada = 1; // para simplificar por defecto edit
            boolean nuevo = false; // para controlar el nuevo registro

            if(radioButton_New.isSelected()){
                // New edit -> formulario edit 0
                nuevo = true;
            } else if(radioButton_Select.isSelected()){
                //select -> modal tabla modo 0
                opcion_Marcada = 0;
            } else if(radioButton_Delete.isSelected()){
                // delete -> modal tabla: modo 2
                opcion_Marcada = 2;
            } else{
                // edit -> modal tabla: modo 1
            }

            int tabla = (int)comboTablas.getSelectedIndex();

            if (nuevo) {
                try {
                    new Formulario_dialog("Nuevo", tabla, 1, 0, getPermisoUser());
                } catch (SQLException ex) {
                    Logger.getLogger(Admin.class.getName()).log(Level.SEVERE, null, ex);
                }
            } else{
                try {
                    new Tabla_dialog(tabla, opcion_Marcada, getId_user(), getPermisoUser());
                } catch (SQLException ex) {
                    Logger.getLogger(Admin.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }
    }
}

```

La pestaña que falta es **administrativo** (vamos el responsable que estaría en la oficina).

Igual que antes constructor que llama al padre y luego crea sus propios elementos.

En este caso es un BorderLayout con dos paneles: al oeste uno con los botones y otro en el centro con las gráficas.

El oeste usa un GridLayout (5,1) para organizar los botones.

El centro usa un BorderLayout, separando el norte (gráfica de barras) y el centro (otro panel con GridLayout para las otras dos gráficas).

Las gráficas estarían montadas en esta pestaña para que el administrativo pudiera usarlas para actividad comercial (ya que es él quien tiene contacto con los repartidores y los clientes. Un ejemplo el funcionario que esta en la central de correos y que atiende a los clientes).

El montaje de las gráficas sería algo como esto.

```
//----- panel graficos centro - este -----  
  
// panel con el gráfico de queso  
  
ChartPanel chartPanel_Queso = Graficos.grafica_Queso();  
chartPanel_Queso.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));  
chartPanel_Queso.setBackground(Color.white);  
chartPanel_Queso.setPreferredSize(new Dimension(50, 100)); // ajustar tamaño  
panelGraficosCentro.add(chartPanel_Queso);  
  
}
```

Esta es la parte que faltaba de la grafica de “queso” que habíamos comentado más arriba.

La otra cosa a destacar de esta pestaña son los botones, todos usan el mismo evento.

```
private class click_operar extends Eventos.Event_Boton_Personalizado{  
    @Override  
    public void actionPerformed(ActionEvent ae) {  
        this.source = (JButton) ae.getSource();  
  
        if("Editar_Perfil".equals(source.getName())){  
            try {  
                // edit -> formulario edit id usuario  
                new Formulario_dialog("Editar perfil", 3, 1, id_user, permisoUser);  
            } catch (SQLException ex) {  
                Logger.getLogger(Administrativo.class.getName()).log(Level.SEVERE, null, ex);  
            }  
        } else {  
            if(OPCIONES[0].equals(source.getName())){  
                try {  
                    // select pedidos -> modal tabla: select tabla pedidos  
                    new Tabla_dialog(1, 0, getId_user(), getPermisoUser());  
                } catch (SQLException ex) {  
                    Logger.getLogger(Administrativo.class.getName()).log(Level.SEVERE, null, ex);  
                }  
            } else if(OPCIONES[1].equals(source.getName())){  
                try {  
                    // new pedido -> mandar a modal formulario: edit tabla pedidos, id_pedido = 0  
                    new Formulario_dialog("Nuevo pedido", 1, 1, 0, getPermisoUser());  
                } catch (SQLException ex) {  
                    Logger.getLogger(Administrativo.class.getName()).log(Level.SEVERE, null, ex);  
                }  
            }  
        }  
    }  
}
```

En este caso el evento se controla con varios “if”. Dependiendo de cual es el botón pulsado se le redirige a un sitio u otro.

```

    }
    } else {
        if(OPCIONES[0].equals(source.getName())){
            try {
                // select pedidos -> modal tabla: select tabla pedidos
                new Tabla_dialog(1, 0, getId_user(), getPermisoUser());
            } catch (SQLException ex) {
                Logger.getLogger(Administrativo.class.getName()).log(Level.SEVERE, null, ex);
            }
        } else if(OPCIONES[1].equals(source.getName())){
            try {
                // new pedido -> mandar a modal formulario: edit tabla pedidos, id_pedido = 0
                new Formulario_dialog("Nuevo pedido", 1, 1, 0, getPermisoUser());
            } catch (SQLException ex) {
                Logger.getLogger(Administrativo.class.getName()).log(Level.SEVERE, null, ex);
            }
        } else if(OPCIONES[2].equals(source.getName())){
            try {
                // new user 0 -> mandar a modal formulario: edit tabla user, id_user = 0, permisos = 0
                new Formulario_dialog("Nuevo Cliente", 3, 1, 0, getPermisoUser());
            } catch (SQLException ex) {
                Logger.getLogger(Administrativo.class.getName()).log(Level.SEVERE, null, ex);
            }
        } else if(OPCIONES[3].equals(source.getName())){
            try {
                // new user 0 -> mandar a modal formulario: edit tabla user, id_user = 0, permisos = 1
                new Formulario_dialog("Nuevo Repartidor", 3, 1, 0, getPermisoUser());
            } catch (SQLException ex) {
                Logger.getLogger(Administrativo.class.getName()).log(Level.SEVERE, null, ex);
            }
        } else if(OPCIONES[4].equals(source.getName())){
            try {
                // select pedidos -> modal tabla: edit tabla pedidos, no repartidor
                new Tabla_dialog(1, 3, getId_user(), getPermisoUser());
            } catch (SQLException ex) {
                Logger.getLogger(Administrativo.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    } else {
        JOptionPane.showMessageDialog(null, "No deberias estar viendo esto....", "Error", JOptionPane.ERROR_MESSAGE);
    }
}

```

Esta era la última pestaña. Ahora nos quedan los JFrame de formulario y tabla. Ambos tienen cierta peculiaridad, se auto-construyen en función de los datos que reciben.

El formulario se crea usando una nomenclatura que recibe el constructor (los parámetros del constructor).

```

*/
public Formulario_dialog(String titulo, int tabla, int modo, int id_consulta, int permiso) throws SQLException {

    // el nuevo repartidor y nuevo cliente son iguales esto lo separa
    boolean nuevo_cliente = false;
    if ("Nuevo Cliente".equals(titulo)) {
        nuevo_cliente = true;
    }

    setTitle(titulo);

    boolean editable = false; // modo 0 y 2

    if (modo == 1 || modo == 3) {
        editable = true;
    }
}

```

Estos parámetros le dicen al JFrame: título que tiene, tipo de “lectura” que tiene el formulario (consultar, editar, borrar, nuevo registro), el id de la consulta (si es nuevo entra en 0) y el permiso de la persona que realiza la operación.

Con estos datos puede realizar una petición y la base de datos le dará los campos y los datos.

Un ejemplo:

```

    }
} else if (tabla == 3) { // usuario
    if (permiso > 1) { // administrativo o admin
        if (id_consulta == 0) {

            String[] columnas = {"id_usuario", "nombre", "apellidos", "correo", "id_provincia"};
            for (String columna : columnas) {
                datos.put(columna, "");
            }
            click_execute_listener = new click_Execute("insertar", tablaString);

        } else {
            datos = DB.get_usuario_by_id(id_consulta);

            if (modo == 2) {

                click_execute_listener = new click_Execute("borrar", tablaString);
            } else {

                click_execute_listener = new click_Execute("actualizar", tablaString);
            }
        }
    } else {
        click_execute_listener = new click_Execute("actualizar", tablaString);
        datos = DB.get_usuario_by_id(id_consulta);
    }
} else { // pedidos
    if (permiso > 1) { // cliente, admin, administrador

```

Esta sería el formulario para la tabla de usuarios. Hace la preparación y pide los datos.

```

----- Carga datos ----- */
container_principal.setLayout(new GridLayout(datos.size() + 1, 1));

ArrayList<JTextField> valores = new ArrayList<JTextField>();

for (String columna : datos.keySet()) {

    JPanel flow_container = new JPanel();

    flow_container.add(new JLabel(columna + " : "));

    JTextField valor = new JTextField(15);
    valor.setText(datos.get(columna));
    valor.setEditable(editable);

    valor.setName(columna);

    valores.add(valor);

    flow_container.add(valor);

    container_principal.add(flow_container, CENTER_ALIGNMENT);

}

click_execute_listener.setComponents_toGetData(valores);

```

Luego el frame (GridLayout) crea los campos de manera automática. Luego añade los dos botones: cancelar y operar.

El evento de cancelar es:

```

    */
    @Override
    public void actionPerformed(ActionEvent e) {
        setVisible(false);
        dispose();
    }

```

Y el de operar es:

```
@Override
public void actionPerformed(ActionEvent ae) {
    HashMap<String, Object> data = new HashMap<String, Object>();
    String filtro;
    for (JTextField JTF : JTFvalores) {
        if (!JTF.getText().equals("")) {
            data.put(JTF.getName(), JTF.getText());
        }
    }

    switch (tabla) {
        case "usuario":
            filtro = "id_usuario = " + data.get("id_usuario");
            break;
        case "provincia":
            filtro = "id_provincia = " + data.get("id_provincia");
            break;
        case "articulo":
            filtro = "id_articulo = " + data.get("id_articulo");
            break;
        case "pedido":
            filtro = "id_articulo = " + data.get("id_articulo");
            break;
        default:
            filtro = "";
            break;
    }

    if (modo.equals("actualizar")) {
        DB.update(tabla, data, filtro);
    } else if (modo.equals("insertar")) {
        DB.insert(tabla, data);
    } else if (modo.equals("borrar")) {
        DB.delete(tabla, filtro);
    }

    JOptionPane.showMessageDialog(null, "Se ha realizado la operación", "Aviso", JOptionPane.PLAIN_MESSAGE);
}
```

En este punto se realiza la operación y se informa al usuario.

El siguiente JFrame es un poco más complicado de seguir: **Jframe tabla**.

El mayor problema de este frame es la cantidad de elementos que tiene que cargar y ordenar.

```
public Tabla_dialog(int tabla, int modo, int id, int per) throws SQLException {
    // atributos
    this.tabla = tabla;
    this.modo = modo;
    this.id_user = id;
    this.permiso = per;

    // propiedades frame
    setTitle(TABLAS[this.tabla]);
    setLayout(new BorderLayout());
    setDefaultCloseOperation(DISPOSE_ON_CLOSE);

    // cargar los datos de las consultas
    cargadorDeDatos();

    /* ----- empezamos a construir los paneles ----- */
    //panel operaciones
    add(constructor_Panel_Cabecera(), BorderLayout.NORTH);

    //panel tabla datos
    add(constructor_Panel_Tabla(), BorderLayout.CENTER);

    // panel botones sur
    add(constructor_Panel_Botones(), BorderLayout.SOUTH);

    // para centrar la pantalla
    mensajeria_app.Utilidades.centrarPantalla(this);

    setVisible(true);
}
```

Lo primero que hace el constructor es “guardar los parámetros que se le pasan.

Luego realiza la consulta y guarda los datos.

```
private void cargarDeDatos() throws SQLException {

    lista_Datos = new ArrayList<String[]>();
    //----- carga de datos -----

    if (permiso == 3) { // admin siempre select *

        lista_Datos = DB.select_all_tabla(TABLAS[tabela]);
        // Select * From "Tabla"
    } else if (modo == 0 && (permiso == 0 || permiso == 1)) { // select pedidos desde cliente
        lista_Datos = DB.select_cliente_repartidor(TABLAS[tabela], permiso, id_user);
        // Select * From 'pedidos' where id_cliente = id
        // select pedidos desde repartidor
        // select * from pedidos where id_repartidor = id
    } else if (modo == 3 && permiso == 2) {
        lista_Datos = DB.select_cliente_repartidor(TABLAS[tabela], permiso, id_user);
        // select * from pedidos where id_repartidor is not null -> esto va a ir a un edit
    } else if (modo == 0 && permiso == 2) {
        lista_Datos = DB.select_all_tabla("pedido");
    } else {
        JOptionPane.showMessageDialog(null, "Esto no deberias estarlo viendo, contacte con un administrador", "Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

Y por ultimo empieza a construir el frame. Se han separado las secciones en métodos que construyen cada sección para facilitar su seguimiento (el código puede ser un poco denso de leer si estuviera todo junto o si se busca alguna parte en concreto del mismo).

El JFrame esta compuesto por un JPanel que tiene un BorderLayout. Cada uno de los métodos se encarga de cada sección del mismo Layout (un método para norte otro para centro y otro para sur)

En la parte norte (GridLayout) se encuentra un pequeño formulario para facilitar la búsqueda o simplemente realizar las operación (recordar que se puede llegar a este JFrame desde distintas partes del programa) junto con dos botones (limpiar y buscar). Los eventos de los botones serán explicados al final.

Luego construye la parte central que tiene un separador y la tabla. Para facilitar el seguimiento de los componentes se ha usado un panel extra aquí (BorderLayout), dejando el separador en la sección norte del panel y la tabla en el centro.

La tabla es un Jtable que se construye usando los datos que se cargaron anteriormente y posee un evento anónimo.

```
// solo que se vea
tabla_datos.setEnabled(false);
//tablaConection.setPreferredSize(new Dimension(WIDTH, HEIGHT));

//evento
tabla_datos.addMouseListener(new java.awt.event.MouseAdapter() {
    @Override
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        int row = tabla_datos.rowAtPoint(evt.getPoint());
        int col = tabla_datos.columnAtPoint(evt.getPoint());
        if (row >= 0 && col >= 0) {
            accionEnTabla(row, col);
        }
    }
});

JScrollPane sp = new JScrollPane(tabla_datos);

panelTabla.add(sp, BorderLayout.CENTER);

return panelTabla;
}
```

El evento lo que hace es recoger en que punto se a pinchado de la tabla y ver la columna y fila que corresponde. Luego lo envía a un método y este carga los datos en el pequeño formulario de la cabecera (el de la sección norte). De esta manera si pinchamos en la tabla se cargan solo los datos de donde hagamos clic.

La sección sur solo es el botón de cancelar, pero se ha mantenido la misma estructura para facilitar la lectura del código (el JFrame tiene cerca de 450 línea el solo y ver todo el código junto puede intimidar o dificultar la lectura).

Dependiendo el permiso del usuario, el botón de búsqueda (en la cabecera) estará habilitado o no.

Si lo está permite abrir un formulario con los datos seleccionados (carga los datos, recupera el id y abre formulario con ese id) y así realizar la operación que se estaba haciendo (consulta, editar, etc).

```
private void envioFormulario(String[] rowEnvio) throws SQLException {
    String titulo = MODO_TABLAS[modo] + " " + TABLAS[tabela];
    int id;
    try {
        id = Integer.parseInt(rowEnvio[0]);
    } catch (Exception e) {
        id = 500; // <- EN TEORIA ESTABA MONTADO PARA QUE EL CAMPO 1 DE LA TABLA SIEMPRE SEA ID

        /* SI NO ES CAMPO ID EL INDICE 0 DE LA ROW HAY QUE PREGUNTAR A LA BASE DE DATOS
        POR EL REGISTRO Y RECUPERAR SU ID*/
    }

    new Formulario_dialog(titulo, tabela, modo, id, permiso);
}
```

Como siempre hay un control de errores que se encarga de que no falle el programa: en la imagen se puede ver el id 500 que es lo que sucede cuando falla la búsqueda por id, evitando que el programa se cuelgue.

Ese evento tiene un paso previo, que es el evento de “buscar”.

```
this.source = (JButton) ae.getSource();

switch (source.getName()) {
    case "Buscar":
        //JOptionPane.showMessageDialog(null, "opcion 1 ", "tester", JOptionPane.PLAIN_MESSAGE);
        if (permiso > 1) { // administrador o admin
            String[] rowEnvio = null;
            if (!"".equals(txt_id.getText()) && !txt_id.getText().isEmpty()) { // si id no esta vacio y no es
                for (String[] row : lista_Datos) {
                    if (txt_id.getText().equals(row[0])) {
                        rowEnvio = row;
                        break; // encontro la row -> salimos
                    }
                }
            } else if (!"".equals(txt_campo1.getText()) && !txt_campo1.getText().isEmpty()) {
                for (String[] row : lista_Datos) {
                    if (txt_campo1.getText().equals(row[1])) {
                        rowEnvio = row;
                        break; // encontro la row -> salimos
                    }
                }
            } else if (!"".equals(txt_campo2.getText()) && !txt_campo2.getText().isEmpty()) {
                for (String[] row : lista_Datos) {
                    if (row.length > 1 && txt_campo2.getText().equals(row[2])) {
                        rowEnvio = row;
                        break; // encontro la row -> salimos
                    }
                }
            } else if (!"".equals(txt_campo3.getText()) && !txt_campo3.getText().isEmpty()) {
                for (String[] row : lista_Datos) {
                    if (row.length > 2 && txt_campo3.getText().equals(row[3])) {
                        rowEnvio = row;
                        break; // encontro la row -> salimos
                    }
                }
            }
        }
    }
}
```

Este evento consiste en buscar la fila que contiene los datos de la tabla y ver si existe. Comprueba cada campo del formulario hasta localizar una coincidencia y busca en el ArrayList que tiene guardado los datos que id tiene ese registro luego se lo pasa a “envioFormulario” (el método de la

Página 32 de 31	Proyecto GUI con acceso a datos	06/11/21
-----------------	---------------------------------	----------

imagen anterior). Si no lo encontrara se le informa al usuario y no se llama al método, simplemente se limpian los campos de búsqueda.