

# **MANUAL TÉCNICO USAC-DELIVERY**

**Luis Carlos Corleto Marroquín**

**Dirigido a:**

Este manual va dirigido a la empresa USAC-Delivery y su propietario Freddy Alejandro Monterroso.

**Surgimiento:**

Este sistema surgió a raíz de la problemática causada por el virus COVID-19, que generó una pandemia a nivel mundial en el año 2020, la pandemia ocasionó que la mayor parte de la población guardara cuarentena dentro de sus casas, con razón de no perder ventas, mejorar la eficiencia de la empresa y poder realizar envíos sin la necesidad de que los clientes salgan de sus casas, surge la creación de este software de escritorio como demo.

**Descripción:**

Este software es una tienda de paquetería, esta puede realizar ventas a distintos departamentos y municipios del mismo departamento, la aplicación cuenta con la opción de crear usuarios con distintos roles, en dónde el rol Kiosco, podrá manejar las propias ventas de su Kiosco.

El programa cuenta con una pestaña de autenticación, en donde identifica automáticamente si el usuario es el usuario administrador, de lo contrario se verificará si el usuario existe registrado en el sistema, de lo contrario solicitará que el usuario se registre en el sistema.

Si alguna persona desea registrarse lo dirigirá a la pestaña de registro, en donde ingresará datos como el correo, contraseña con ciertos caracteres obligatorios, un rol, fecha de nacimiento, Dpi y muchos datos más.

Si se identifica que el usuario es administrador, se hará visible el menú del administrador, en donde podrá manejar las distintas opciones a su cargo, como el agregar, eliminar o modificar Regiones, con su respectivo código, precio especial y estándar, también podrá modificar los Kioscos de la misma manera y

los Departamentos y municipios, el usuario administrador también posee la opción de ver reportes, en donde podrá visualizar, los usuarios con más ventas, el total de todas las ventas, el total de todas los paquetes enviados y la región con más paquetes enviados.

Si el usuario es un usuario “normal” o cliente, este tendrá la opción de agregar tarjetas para el pago de el envío, una opción para agregar sus datos de facturación, nombre, nit, dirección, otra opción para cotizar el pago de los envíos y posteriormente cancelarlo y una última opción para ver los envíos que ha solicitado.

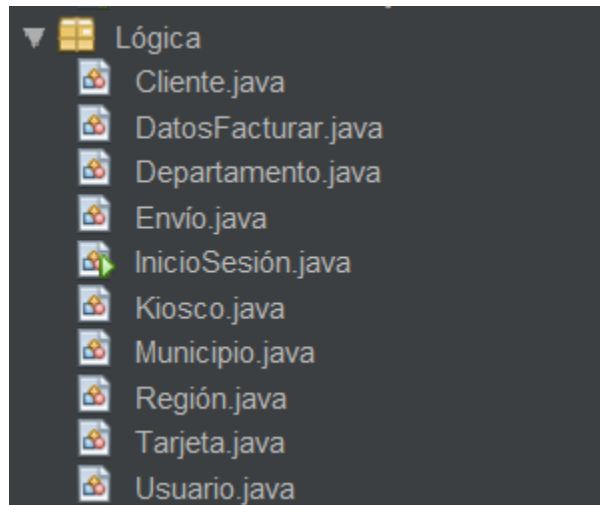
El usuario podrá agregar las tarjetas de crédito/débito que desee en el sistema, ofuscando todos los dígitos exceptuando los últimos 4, además de agregar el número de su tarjeta, el nombre de la tarjeta, su fecha de vencimiento y el VSC que se agrega manualmente, este no se guardará en el sistema.

El usuario también podrá agregar los datos que él desee para la facturación, su nombre, su dirección y su número de NIT, estos se guardarán en el sistema y se mostrarán en la cotización para poder seleccionar el que el cliente desee.

El usuario podrá ingresar a la pestaña de cotizar para poder realizar las compras que él desee, seleccionar el Departamento, municipio de Origen, departamento y municipio de destino, el número de paquetes que desea enviar, el tamaño del paquete, el tipo de envío, especial y estándar, en donde varia el precio de envío, posteriormente podrá seleccionar el tipo de pago, contra entrega con un pago adicional de Q 5.0, pago con tarjeta en donde podrá seleccionar una de las tarjetas que agregó anteriormente y los datos de facturación que él desee, luego de cancelar se le mostrará el tipo de envío que seleccionó y el precio total del envío, este podrá descargar una factura y una guía con distintos datos.

Luego de realizar una compra el usuario podrá visualizar los envíos que ha realizado en el menú, ver envíos solicitados.

Para la creación del software se utilizaron las siguientes clases:



- **Cliente:** Objeto de tipo cliente, este es un objeto que contiene los datos del cliente.
- **DatosFacturar:** Objeto de tipo Datos de facturación con sus respectivos atributos.
- **Departamento:** Objeto de tipo Departamentos con sus respectivos atributos.
- **Envío:** Objeto de tipo Envío, donde se almacenan los datos de los envíos realizados.
- **InicioSesión:** Clase main que ejecuta el programa.
- **Kiosco:** Objeto de tipo Kiosco, con sus respectivos atributos.
- **Municipio:** Objeto de tipo Municipio con sus respectivos atributos.
- **Región:** Objeto de tipo Región con sus respectivos atributos.
- **Tarjeta:** Objeto de tipo tarjeta, este es un objeto que almacena los datos de la tarjeta del cliente.
- **Usuario:** Objeto de tipo usuario que maneja y almacena los datos del usuario a registrar por el administrador.



```
package Lógica;

import Interfaz.Autenticación;

public class InicioSesión {

    public static void main(String[] args) {
        Autenticación pantal = new
Autenticación();setVisible(true);
        pantal.setLocationRelativeTo(null);
    }
}
```

La clase main “InicioSesión” contiene el código presentado en la imagen anterior, en donde se realiza una instancia del JFrame llamado Autenticación, que es la raíz de todo el programa, no se realizó ningún método, función o declaración de variable, todo fue realizado en los JFrames que se presentarán a continuación, agregando funcionalidad a los botones.

Se tienen JFrames con códigos importantes como:

## Autenticación

```
private void botiniciosesionActionPerformed(java.awt.event.ActionEvent evt) {  
  
    this.setVisible(false);  
    //Instancias de los JFrames.  
    if (Ma == null) {  
        Ma = new MenúAdmin();  
    }  
    Mu = new MenúUsuario();  
    boolean usuarioEncontrado = false;  
    //Si el usuario es admin.  
    if ("1".equals(correo.getText()) && "2".equals(contraseña.getText())) {  
        Ma.setVisible(true);  
        Ma.setLocationRelativeTo(null);  
        usuarioEncontrado = true;  
    } //El usuario es normal, verifica si el usuario está en el arreglo.  
    else {  
        for (int i = 0; i < listaUsuarios.size(); i++) {  
            if (listaUsuarios.get(i).getCorreo().equals(correo.getText()) &&  
listaUsuarios.get(i).getContraseña().equals(contraseña.getText())) {  
                usuarioActual = listaUsuarios.get(i);  
                Mu.setVisible(true);  
                Mu.setLocationRelativeTo(null);  
                usuarioEncontrado = true;  
                break;  
            }  
        }  
    } //Si el usuario no es normal, ni administrador entonces no se encuentra registrado.  
    if (!usuarioEncontrado) {  
        JOptionPane.showMessageDialog(this, "Este correo no se encuentra registrado", "Error",  
JOptionPane.ERROR_MESSAGE);  
        this.setVisible(true);  
    }  
}
```

Esta parte es importante en el manejo del programa, ya que si el usuario principalmente el Administrador, no recuerda su contraseña, no podrá acceder al sistema, los datos del administrador van quemados en el sistema, el código verifica si las cadenas de caracteres ingresadas en los TextFields son iguales a los datos quemados del administrado, si es así, identifica que es un administrador y lo dirige a la pestaña del administrador, de lo contrario, verificará si el usuario es algún cliente, si este se encuentra registrado, se le dirigirá a la pestaña correspondiente del usuario, si ninguna de las 2 condiciones se cumple, se hace otra verificación para comprobar si el valor booleano es distinto de 0 y mostrará un mensaje en donde se indicará que el usuario no es ni administrador, ni se encuentra registrado.

Dentro del menú del Administrador, posee las opciones de, registrar usuarios, registrar, mostrar, eliminar y modificar, Kioscos, Regiones, Departamentos y Municipios además de poder ver los reportes correspondientes.

Para el registro, la eliminación y la modificación se utilizaron botones con distintos métodos para la realización de las mismas opciones mencionadas.

**Como ejemplo los Kioscos:**

```

private void agregarKioscoActionPerformed(java.awt.event.ActionEvent evt) {

    /*Verificaciones para los TextFields, el código solo va en mayúsculas
    y debe llevar por lo menos un número para un mejor control de los códigos.
    Las Regiones únicamente pueden ser las quemadas o las ya agregadas, de lo contrario
    mostrará un JOptionPane con el error de que la región no fue encontrada.
    Los TextFields no pueden estar vacíos.*/
    boolean encontrado = false;
    for (int i = 0; i < listaRegiones.size(); i++) {
        if (codRegión.getText().equals(listaRegiones.get(i).getAgcódigoRegión())) {
            encontrado = true;
            break;
        }
    }
    if (codRegión.getText().equals(Regiones.regionm) || codRegión.getText().equals(Regiones.regionnt) ||
        codRegión.getText().equals(Regiones.regionno) || codRegión.getText().equals(Regiones.regionso) ||
        codRegión.getText().equals(Regiones.regionsoc) || codRegión.getText().equals(Regiones.regionnoc)) {
        encontrado = true;
    }
    if (!encontrado) {
        JOptionPane.showMessageDialog(this, "Esta región no se encuentra disponible", "Error",
        JOptionPane.ERROR_MESSAGE);
        this.setVisible(true);
        return;
    }
    if (NombreKiosco.getText().equals("")) {
        JOptionPane.showMessageDialog(this, "Debe ingresar el nombre del Kiosco", "Error",
        JOptionPane.ERROR_MESSAGE);
        this.setVisible(true);
        return;
    }
    if (codKiosco.getText().equals("")) {
        JOptionPane.showMessageDialog(this, "Debe ingresar el código del Kiosco", "Error",
        JOptionPane.ERROR_MESSAGE);
        this.setVisible(true);
        return;
    }
    if (codRegión.getText().equals("")) {
        JOptionPane.showMessageDialog(this, "Debe ingresar el código de la región", "Error",
        JOptionPane.ERROR_MESSAGE);
        this.setVisible(true);
        return;
    }
    if (!codKiosco.getText().matches(".*[0-9].*") || !codKiosco.getText().matches(".*[A-Z].*")) {
        JOptionPane.showMessageDialog(this, "El código debe tener al menos un número y solo letras
        mayúsculas", "Error", JOptionPane.ERROR_MESSAGE);
        this.setVisible(true);
        return;
    }
}

```



Antes de poder ingresar un nuevo Kiosco, se deben hacer varias verificaciones, como que todos los campos estén llenos, que el código de la región exista en el ArrayList listaRegiones o en su defecto que el código pertenezca a alguna de las regiones agregadas con datos quemados dados en la redacción del proyecto.

```
//Llenado del objeto
Kiosco kioscos = new Kiosco(codKiosco.getText(), NombreKiosco.getText(), codRegión.getText());
//Agregándolo al ArrayList.
listaKioscos.add(kioscos);
//Agregándolo a la tabla listadeKioscos.
model.addRow(new Object[]{codKiosco.getText(), NombreKiosco.getText(), codRegión.getText()});
listadeKioscos.setModel(model);
JOptionPane.showMessageDialog(this, "Kiosco agregado correctamente", "Proceso Exitoso",
JOptionPane.INFORMATION_MESSAGE);
this.setVisible(true);
//Vaciando los TextFields.
codKiosco.setText("");
NombreKiosco.setText("");
codRegión.setText("");
}
```

Si todas las condiciones se cumplen, se procede a llenar Objetos dentro del ArrayList de tipo Kiosco, además de agregarlos en un JTable en donde se podrán visualizar de mejor manera los Kioscos agregados, con esta JTable es más fácil modificar y eliminar los datos.

```
private void editarKioscoActionPerformed(java.awt.event.ActionEvent evt) {  
    //Verificando si alguna fila no fue seleccionada.  
    if (listadeKioscos.getSelectedRow() == -1) {  
        JOptionPane.showMessageDialog(null, "Debe seleccionar una fila", "Advertencia",  
JOptionPane.WARNING_MESSAGE);  
    } else {  
        /*Si alguna fila fue seleccionada, procede a enviar los datos de la fila a los TextFields  
        para poder editarlos.*/  
        filaSeleccionada = listadeKioscos.getSelectedRow();  
        codKiosco.setText(listadeKioscos.getValueAt(listadeKioscos.getSelectedRow(), 0).toString());  
        NombreKiosco.setText(listadeKioscos.getValueAt(listadeKioscos.getSelectedRow(), 1).toString());  
        codRegión.setText(listadeKioscos.getValueAt(listadeKioscos.getSelectedRow(), 2).toString());  
    }  
}
```

Para la modificación del objeto correspondiente en este caso los Kioscos, se utilizó este método del JTable “getSelectedRow” dentro del botón Editar, su función es obtener la fila seleccionada, primero se hace una verificación, si no hay ninguna seleccionada, mostrará un mensaje de error y deberá de seleccionarse una, si alguna de las filas es seleccionada, se manda a llamar el objeto dentro del ArrayList al que corresponde la fila seleccionada y se muestran sus datos en sus respectivos TextFields para proceder a editarlos.

```
Kiosco kioscoSeleccionado = listaKioscos.get(filaSeleccionada);
//Colocar el nuevo Kiosco modificado en la tabla y en el ArrayList.
kioscoSeleccionado.setCódigoKiosco(codKiosco.getText().trim());
kioscoSeleccionado.setNombreKioscos(NombreKiosco.getText().trim());
kioscoSeleccionado.setCódigoRegión(codRegión.getText().trim());
//Actualizar los datos en la tabla.
listadeKioscos.setValueAt(codKiosco.getText().trim(), filaSeleccionada, 0);
listadeKioscos.setValueAt(NombreKiosco.getText().trim(), filaSeleccionada, 1);
listadeKioscos.setValueAt(codRegión.getText().trim(), filaSeleccionada, 2);
JOptionPane.showMessageDialog(null, "Kiosco modificado correctamente", "Proceso exitoso",
JOptionPane.INFORMATION_MESSAGE);
//Limpiando los TextFields.
codKiosco.setText("");
NombreKiosco.setText("");
codRegión.setText("");
```

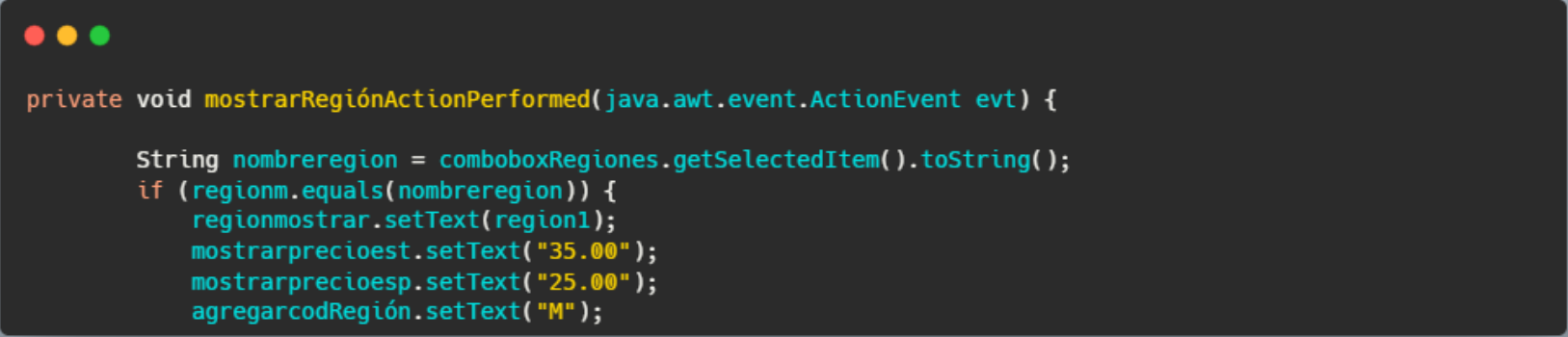
Este código se encuentra dentro del botón Modificar en la pestaña de Kioscos, antes de él se realizan las mismas verificaciones que al agregar un Kiosco, para que los datos estén ordenados y no existan anomalías como que un código no tenga sus restricciones correspondientes, de nuevo se utiliza un método del JTable, setValueAt que actualiza los datos de una tabla para establecer el nuevo valor correspondiente, antes de colocar los nuevos datos en la tabla también se utilizan los métodos Setters de cada atributo en nuestro Objeto listaKioscos, haciendo que los datos se modifiquen en el ArrayList y no solo en la tabla.

```
private void eliminarKioscoActionPerformed(java.awt.event.ActionEvent evt) {  
  
    //Obteniendo la fila seleccionada.  
    int selectedRow = listadeKioscos.getSelectedRow();  
    //Validando que la fila si haya sido seleccionado.  
    if (selectedRow != -1) {  
        //Obteniendo el objeto correspondiente al ArrayList dependiendo de la fila seleccionada.  
        Kiosco kiosco = listaKioscos.get(selectedRow);  
  
        //Método para eliminar la fila seleccionada.  
        listaKioscos.remove(kiosco);  
  
        //Actualizando la tabla sin el objeto que se eliminó previamente.  
        DefaultTableModel modelo = (DefaultTableModel) listadeKioscos.getModel();  
        modelo.removeRow(selectedRow);  
  
        JOptionPane.showMessageDialog(null, "Kiosco eliminado correctamente", "Proceso Exitoso",  
JOptionPane.INFORMATION_MESSAGE);  
        //Vaciando de nuevo los TextFields.  
        codKiosco.setText("");  
        NombreKiosco.setText("");  
        codRegión.setText("");  
    } else { //Si no se seleccionó ninguna fila mostrará un JOptionPane con el error.  
        JOptionPane.showMessageDialog(null, "Debe seleccionar una fila para eliminar", "Error",  
JOptionPane.ERROR_MESSAGE);  
    }  
}
```

Para la eliminación de los Kioscos, de nuevo se utiliza el método `getSelectedRow()`, también se utiliza el método `remove` para eliminar un dato en específico de una colección de datos en este caso un `ArrayList` de tipo `Kiosco`, de nuevo se verifica que alguna fila sea seleccionada, si esta condición se cumple, se procede a eliminar de la lista de objetos el objeto seleccionado, además de eliminarse del `JTable`.

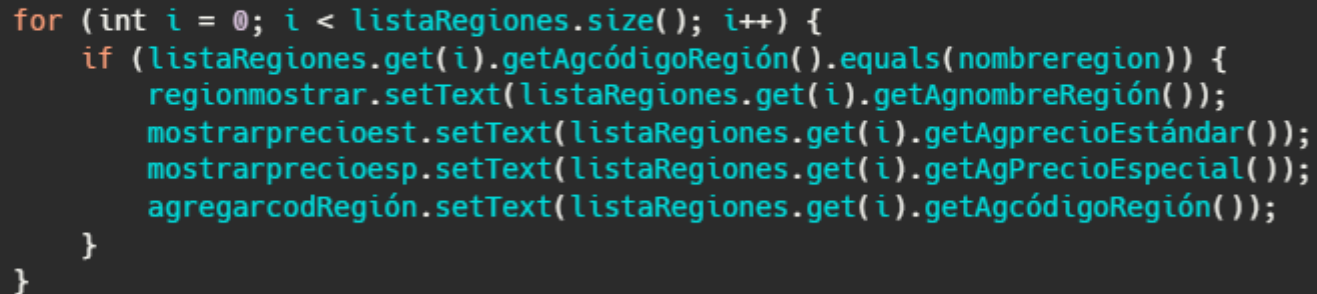
Estos métodos se utilizaron en 3 de las pestañas del administrador, como es la explicada anteriormente, “Kioscos”, “Departamentos” y “Municipios, en donde se realizan las mismas verificaciones, el mismo uso de métodos de eliminación, selección y modificación, cada pestaña con sus correspondientes listas de objetos y datos correspondientes, también se hizo el mismo uso de las funciones y métodos para 2 pestañas del menú Usuario, en donde este podrá manejar sus datos de facturación y sus tarjetas de crédito/débito, modificar y eliminar, claramente cada pestaña con sus datos correspondientes.

Para el manejo de las regiones es distinto, puesto que se utilizó un `JComboBox` para el manejo de estas, utilizando como ítems por defectos los códigos de las regiones agregadas por datos quemados, y agregando regiones si se desea, en donde estas agregadas se podrán modificar y eliminar, las que están agregadas por datos quemados no pueden eliminarse, puesto que estas vienen por defecto en el sistema y se encontrarán para siempre en cualquier vez que se ejecute el programa.



```
private void mostrarRegiónActionPerformed(java.awt.event.ActionEvent evt) {  
  
    String nombreregion = comboboxRegiones.getSelectedItem().toString();  
    if (regionm.equals(nombreregion)) {  
        regionmostrar.setText(region1);  
        mostrarprecioest.setText("35.00");  
        mostrarprecioesp.setText("25.00");  
        agregarcodRegión.setText("M");  
    }  
}
```


La pestaña posee un botón “Mostrar” en dónde se mostrarán los datos en sus correspondientes TextFields, se hace una verificación para saber cuál item está seleccionando y en base al item seleccionado muestra su precio estándar y especial, el nombre de la región almacenadas en variables y su código, este método se repite para cada región agregada por el sistema, para las regiones agregadas por el administrador tenemos el siguiente método:



```
for (int i = 0; i < listaRegiones.size(); i++) {  
    if (listaRegiones.get(i).getAgcódigoRegión().equals(nombreregion)) {  
        regionmostrar.setText(listaRegiones.get(i).getAgnombreRegión());  
        mostrarprecioest.setText(listaRegiones.get(i).getAgprecioEstándar());  
        mostrarprecioesp.setText(listaRegiones.get(i).getAgPrecioEspecial());  
        agregarcodRegión.setText(listaRegiones.get(i).getAgcódigoRegión());  
    }  
}
```

Aquí se crea un ciclo para verificar si el item seleccionado es igual a cualquier código dentro del ArrayList listaRegiones, si encuentra alguna coincidencia procede de nuevo a mostrarse su respectivo precio estándar y especial, su nombre y su código de región.

En la siguiente parte del código hace una verificación extra, en donde no se pueden repetir los códigos de las regiones que se agregarán, de lo contrario la región no será agregada, antes de también se verifica que ninguno de los campos estén vacíos, si se cumplen todas las condiciones, proceden a guardarse los objetos dentro del ArrayList listaRegiones, estas regiones también podrán ser utilizadas para cotizaciones, ingreso de departamentos y Kioscos con sus respectivos precios, luego de agregar la región se limpian todos los TextFields por si el administrador desea agregar uno, no debe borrar campo por campo.



```
String codigoRegion = agregarcodRegión.getText();
for (Región r : listaRegiones) {
    if (r.getAgcódigoRegión().equals(codigoRegion)) {
        JOptionPane.showMessageDialog(this, "El código de región ya existe", "Error",
JOptionPane.ERROR_MESSAGE);
        this.setVisible(true);
        return;
    }
}
Región reg = new Región(agregarcodRegión.getText(), regionmostrar.getText(),
mostrarprecioest.getText(), mostrarprecioesp.getText());
listaRegiones.add(reg);
comboBoxRegiones.addItem(codigoRegion);
JOptionPane.showMessageDialog(this, "Región agregada correctamente", "Proceso exitoso",
JOptionPane.INFORMATION_MESSAGE);
this.setVisible(true);
regionmostrar.setText("");
mostrarprecioest.setText("");
mostrarprecioesp.setText("");
agregarcodRegión.setText("");
}
```



```

private void eliminarRegiónActionPerformed(java.awt.event.ActionEvent evt) {

    String regionAEliminar = comboboxRegiones.getSelectedItem().toString();
    for (Región region : listaRegiones) {
        if (region.getAgcódigoRegión().equals(regionAEliminar)) {
            listaRegiones.remove(region);
            break;
        }
    }
    comboboxRegiones.removeItemAt(comboboxRegiones.getSelectedIndex());
    JOptionPane.showMessageDialog(this, "Región Eliminada", "Proceso Exitoso",
JOptionPane.INFORMATION_MESSAGE);
    regionmostrar.setText("");
    mostrarprecioest.setText("");
    mostrarprecioesp.setText("");
    agregarcodRegión.setText("");
}

```

En este caso si se utilizaron los mismo métodos que en las pestañas anteriores, con la diferencia de que los datos se eliminaran ahora, tanto como del comboBox y del ArrayList de objetos, obtiene el item seleccionado, que será comparado con algún código dentro del ArrayList y procederá a eliminarse de los 2 lados, aunque para los datos ya agregados es más sencillo, solo elimina los datos del comboBox y anula el uso de las variables correspondientes a la región eliminada.

Ingresando al menú del usuario, nos encontramos con la opción de cotizar, en donde se podrá cotizar y realizar el envío que el cliente desee realizar, además de poder descargar su factura y la guía de su pedido.

```
private void cotizarActionPerformed(java.awt.event.ActionEvent evt) {  
  
    for (int i = 0; i < listaDepartamentos.size(); i++) {  
        if  
(departamentoDestino.getSelectedItem().toString().equals(listaDepartamentos.get(i).getNombreDpto()) &&  
paqPequeño.isSelected()) {  
            if (listaDepartamentos.get(i).getCodRegDpto().equals(Regiones.regionm)) {  
                int valornumérico = Integer.parseInt(númeroDePaquetes.getText());  
                totalEstándar = 35.00 * valornumérico;  
                totalEST.setText(String.valueOf(totalEstándar));  
                totalEspecial = 25.00 * valornumérico;  
                totalESP.setText(String.valueOf(totalEspecial));  
                ventaregionm++;  
                break;  
            }  
        }  
    }  
}
```

Para la cotización se hicieron verificaciones dentro de un ciclo del tamaño del ArrayList listaDepartamentos, esta condición verifica el tamaño del paquete, la región a la que pertenece o con la que fue registrada el departamento seleccionado, si es así se procede a hacer la operación para la cotización, se utiliza la misma función y condición para los demás tamaños de paquetes y las regiones.

Para las regiones agregadas por el usuario tenemos un código, función y verificación distinta.

```
if (!listaDepartamentos.isEmpty()) {
    for (int i = 0; i < listaDepartamentos.size(); i++) {
        if
        (departamentoDestino.getSelectedItem().toString().equals(listaDepartamentos.get(i).getNombreDpto()) &&
        paqPequeño.isSelected()) {
            if
            (listaDepartamentos.get(i).getCodRegDpto().equals(listaRegiones.get(i).getAgcódigoRegión())) {
                int valornumérico = Integer.parseInt(númeroDePaquetes.getText());
                Double precioEstándar = Double.parseDouble(listaRegiones.get(i).getAgprecioEstándar());
                Double precioEspecial = Double.parseDouble(listaRegiones.get(i).getAgprecioEspecial());
                totalEstándar = precioEstándar * valornumérico;
                totalEST.setText(String.valueOf(totalEstándar));
                totalEspecial = precioEspecial * valornumérico;
                totalESP.setText(String.valueOf(totalEspecial));
                break;
            }
        }
    }
}
```

Primero verifica si el ArrayList se encuentra por lo menos con un objeto dentro de él, esto para evitar errores y que no verifique una lista vacía, se verifica que el departamento seleccionado en el ComboBox coincida con alguno de los nombres agregados en algún objeto del ArrayList, luego se verifica la región a la que pertenece este departamento, para llamar su precio correspondiente y realizar la cotización, se debe aclarar que como estos datos fueron agregados por el administrador, los precios y el código dependen única y totalmente de lo que el administrador decidió al registrarlos.

```
public Cotización() {
    initComponents();
    for (int i = 0; i < listaDepartamentos.size(); i++) {
        departamentoOrigen.addItem(listaDepartamentos.get(i).getNombreDpto());
    }
    for (int i = 0; i < listaDepartamentos.size(); i++) {
        departamentoDestino.addItem(listaDepartamentos.get(i).getNombreDpto());
    }
    for (int i = 0; i < listaMunicipios.size(); i++) {
        municipioOrigen.addItem(listaMunicipios.get(i).getNombreMun());
    }
    for (int i = 0; i < listaMunicipios.size(); i++) {
        municipioDestino.addItem(listaMunicipios.get(i).getNombreMun());
    }
    for (int i = 0; i < listaClientes.size(); i++) {
        comboNombre.addItem(listaClientes.get(i).getNombreCompletoCliente());
    }
    for (int i = 0; i < listaClientes.size(); i++) {
        comboDirección.addItem(listaClientes.get(i).getDirecciónCliente());
    }
    for (int i = 0; i < listaClientes.size(); i++) {
        comboNit.addItem(listaClientes.get(i).getNITCliente());
    }
    model.addColumn("Número de tarjeta");
    model.addColumn("Nombre en la tarjeta");
    listadeTarjetas.setModel(model);
    for (int i = 0; i < listaTarjetas.size(); i++) {
        model.addRow(new Object[]{listaTarjetas.get(i).getNúmerodeTarjeta(),
        listaTarjetas.get(i).getNombredeTarjeta()});
    }
}
```

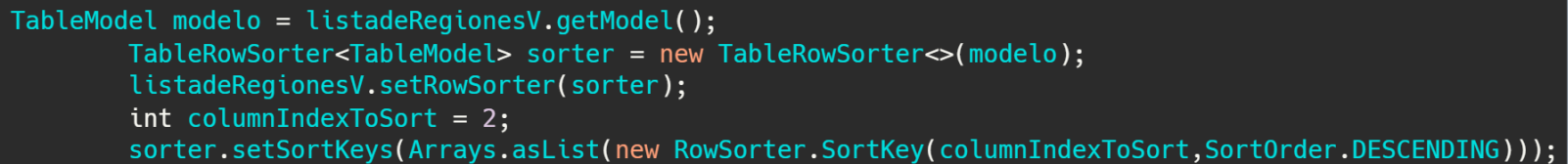
También se realizó esta parte de código en la inicialización del JFrame, en donde ingresa los datos de los ArrayList a sus correspondientes objetos, como los departamentos y municipios a los comobox de Origen y destino, además de las tarjetas de crédito/débito que fueron agregadas por el cliente.

Esta inicialización también se realizó en el apartado de reportes, inicializando con las regiones dentro de una tabla, los usuarios en otra y dos TextFields que indican la cantidad de paquetes enviados y el total de las ventas realizadas.

```
String nombreArchivo = "Factura_" + new Date().getTime() + ".html";
if (paqPequeño.isSelected()) {
    contenido = "<html>\n"
        + "<head>\n"
        + "<title>Factura</title>\n"
        + "<style>\n"
        + "table {\n"
        + "  border-collapse: collapse;\n"
        + "  width: 100%;\n"
        + "}\n"
        + "th, td {\n"
        + "  text-align: left;\n"
        + "  padding: 8px;\n"
        + "  border-bottom: 1px solid #ddd;\n"
        + "}\n"
        + "th {\n"
        + "  background-color: #4CAF50;\n"
        + "  color: white;\n"
        + "}\n"
        + "</style>\n"
        + "<style>\n"
        + "body, h1, h2, h3, h4, h5, h6 {\n"
        + "  font-size: 16px;\n"
        + "  font-family: Arial, sans-serif;\n"
        + "}\n"
        + ".text-box {\n"
        + "  border: 4px solid black;\n"
        + "  padding: 10px;\n"
        + "}\n"
        + "</style>\n"
```

Se utilizaron las librerías FileWriter e IOException para el manejo y descarga de la factura y guía en formato HTML, fue proporcionado un ejemplo básico por parte del tutor académico Freddy Monterroso, se modificó y expandió el código por parte del autor de este manual, modificando el tamaño de las letras, creando tablas, una región encerrada en un cuadro para el posicionamiento de algunos datos y la configuración de la Guía para agregar una imagen de un código de barras.

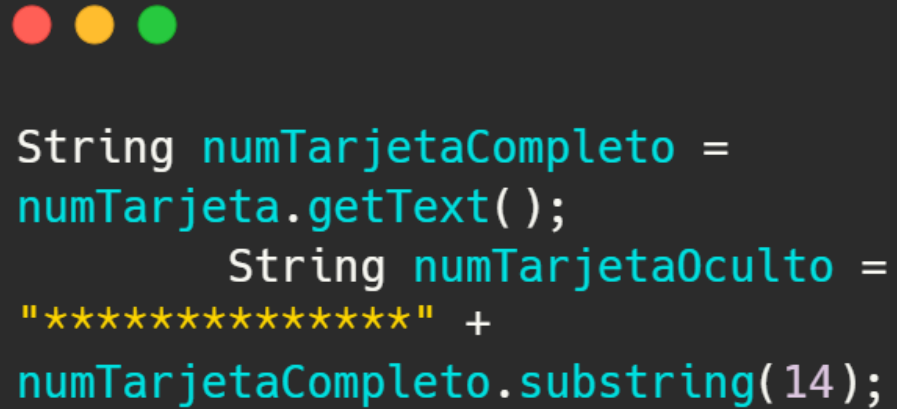
```
</h1>\n"
+ "</head>\n"
+ "<body>\n"
+ "<div class='text-box'>\n"
+ "<h1>" + "Factura número: " + númeroDeFactura + "</h1>\n"
+ "<h1>" + codigo.toString() + "</h1>\n"
+ "<h1>" + "Nombre: " + nombreFactura + "</h1>\n"
+ "<h1>" + "Dirección: " + direcciónFactura + "</h1>\n"
+ "<h1>" + "NIT: " + nitFactura + "</h1>\n"
+ "<h1>" + "Tipo de Pago: " + tipo + "</h1>\n"
+ "<h1>" + "Origen: " + departamentoOrigen.getSelectedItem().toString() + "</h1>\n"
+ "<h1>" + "Destino: " + departamentoDestino.getSelectedItem().toString() + "
+ "</div>\n"
+ "<br>"
+ "<table>\n"
+ "<thead>\n"
+ "<tr>\n"
+ "<th>Número de paquetes</th>\n"
+ "<th>Tamaño del paquete</th>\n"
+ "<th>Total de pago</th>\n"
+ "</tr>\n"
+ "</thead>\n"
+ "<tbody>\n"
+ "<tr>\n"
+ "<td>" + númeroDePaquetes.getText() + "</td>\n"
+ "<td>" + "Pequeño" + "</td>\n"
+ "<td>" + totalFinal.getText() + "</td>\n"
+ "</tr>\n"
+ "</tbody>\n"
+ "</table>\n"
+ "</body>\n"
+ "</html>";
}
```



```
TableModel modelo = listadeRegionesV.getModel();
TableRowSorter<TableModel> sorter = new TableRowSorter<>(modelo);
listadeRegionesV.setRowSorter(sorter);
int columnIndexToSort = 2;
sorter.setSortKeys(Arrays.asList(new RowSorter.SortKey(columnIndexToSort, SortOrder.DESCENDING)));
```

Para el orden descendente de las regiones con más ventas se hizo la implementación de este método, este recupera el modelo de la tabla de el objeto JTable “listaDeRegionesV” en donde se muestran las regiones agregadas que vienen predefinidas y las agregadas por el administrador, este objeto es almacenado en la variable “modelo” que a continuación crea un nuevo objeto TableWorSorter que se inicializa con el modelo de la tabla y lo almacena en la variable de nombre “sorter”. Luego se hace uso del método setRowSorter, que establece el orden de clasificación de la tabla en función del objeto “sorter”, se define el índice de la columna a ordenar y se establece el orden de la clasificación, son el método setSortKeys.

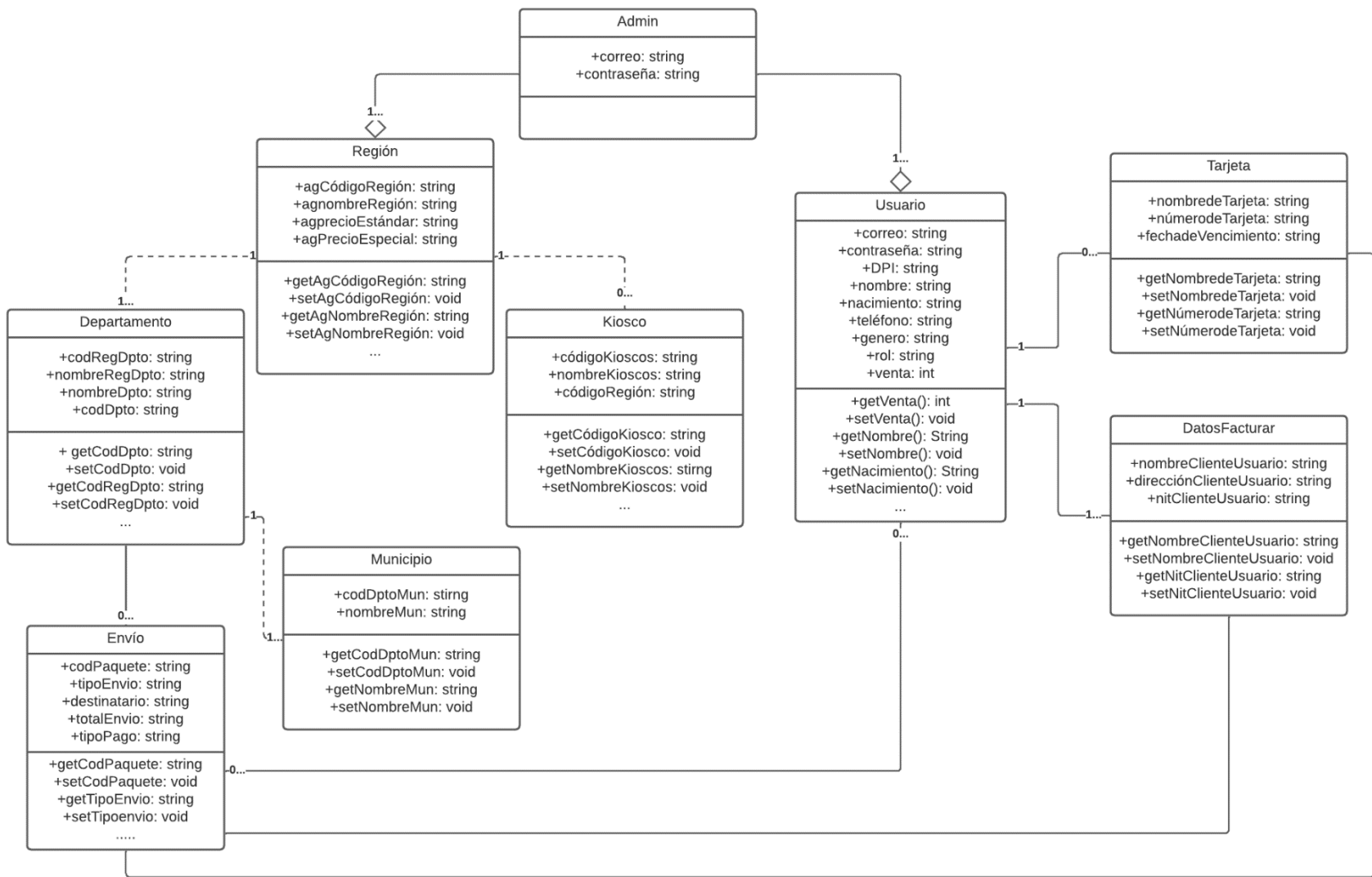
El código de la siguiente imagen toma el número de la tarjeta de crédito ingresado en el TextField correspondiente y lo convierte en una cadena oculta de 14 dígitos, contando los saltos que contiene una tarjeta de crédito cada 4 dígitos, para esto se utiliza el método subString y se utiliza para obtener una subcadena de una cadena dada.



```
String numTarjetaCompleto =  
numTarjeta.getText();  
String numTarjetaOculto =  
"*****" +  
numTarjetaCompleto.substring(14);
```

Este método se utilizó para ofuscar los primeros 12 dígitos de la tarjeta al modificar la tarjeta y al presentar las tarjetas al cliente durante su compra si desea cancelar con una tarjeta de Crédito/Débito agregada por el mismo anteriormente.





Este es el siguiente diagrama de clases, que explica la relación entre todas las clases del programa, no existe una clase llamada Admin, pero se realizó la relación con el usuario administrador, porque sin él, no surgiría ninguna parte del programa, tampoco se agregaron todos los métodos, porque quedaría demasiada extensa la imagen.