

Universidad San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ingeniería en Ciencias y Sistemas

Introducción a la Programación y Computación 1, Sección A

Ing. Marlon Francisco Orellana López

Auxiliar: José Daniel Fajardo

Primer Semestre 2023



Manual Técnico Proyecto 1

Nombre: Carlos Manuel Lima y Lima

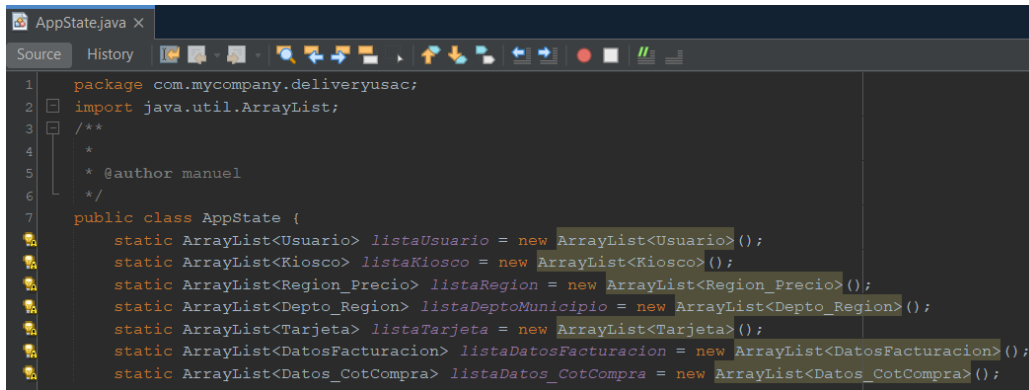
Registro Académico: 202201524

Cui: 3009368850101

Guatemala, 09 de marzo del 2023

1. Clase AppState

Esta clase fue utilizada para crear ArrayList de todos los distintos objetos que son utilizados en el programa. Para la creación de objetos, se utilizaron distintas clases en donde se definió los distintos tipos de atributos de cada objeto.



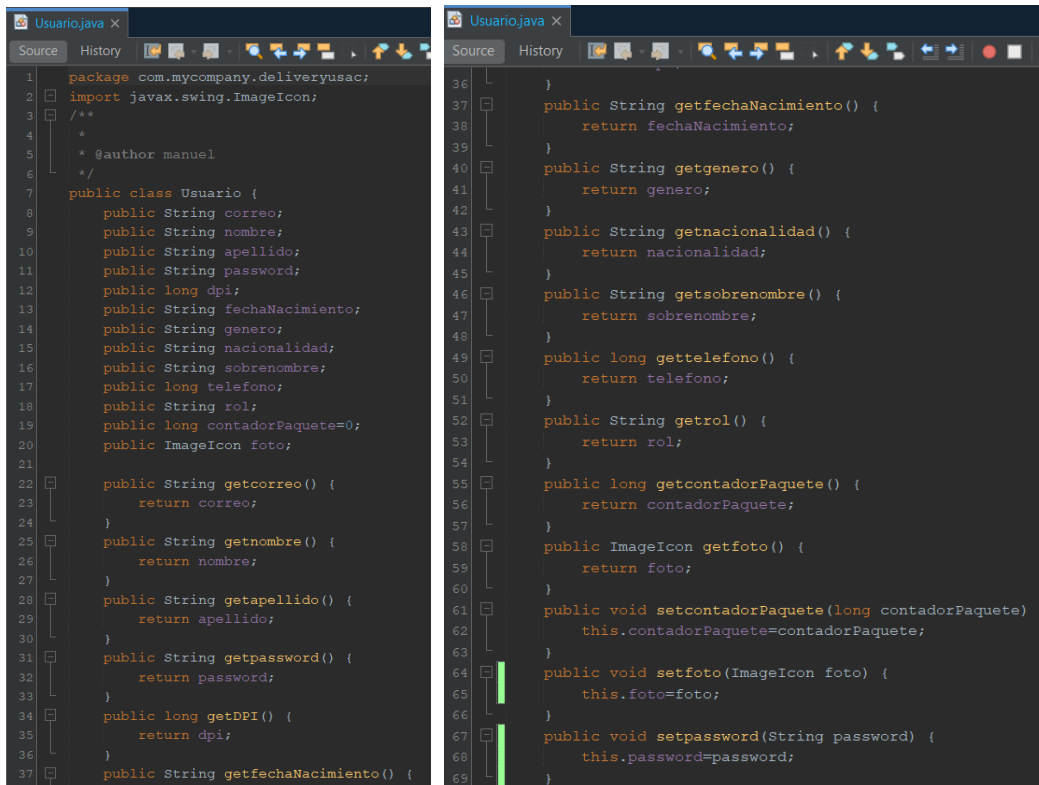
```
1 package com.mycompany.deliveryusac;
2 import java.util.ArrayList;
3 /**
4  *
5  * @author manuel
6  */
7 public class AppState {
8     static ArrayList<Usuario> listaUsuario = new ArrayList<Usuario>();
9     static ArrayList<Kiosco> listaKiosco = new ArrayList<Kiosco>();
10    static ArrayList<Region_Precio> listaRegion = new ArrayList<Region_Precio>();
11    static ArrayList<Depto_Region> listaDeptoMunicipio = new ArrayList<Depto_Region>();
12    static ArrayList<Tarjeta> listaTarjeta = new ArrayList<Tarjeta>();
13    static ArrayList<DatosFacturacion> listaDatosFacturacion = new ArrayList<DatosFacturacion>();
14    static ArrayList<Datos_CotCompra> listaDatos_CotCompra = new ArrayList<Datos_CotCompra>();
15 }
```

Figura 1: Clase AppState

2. Clase Objeto Usuario

En esta clase fueron creados todos los atributos que un nuevo usuario puede tener, es decir correo, nombre, apellido, contraseña, fecha de nacimiento, genero, nacionalidad, sobrenombre, teléfono, DPI, rol, foto y un contador paquete. Este contador paquete será utilizado para mostrar el reporte de cuantos paquetes ha enviado un usuario.

La creación de este tipo de objetos se repite para kioscos, regiones, departamentos-municipios, tarjetas, datos de facturación y datos de cotización-compra.



```
1 package com.mycompany.deliveryusac;
2 import javax.swing.ImageIcon;
3 /**
4  *
5  * @author manuel
6  */
7 public class Usuario {
8     public String correo;
9     public String nombre;
10    public String apellido;
11    public String password;
12    public long dpi;
13    public String fechaNacimiento;
14    public String genero;
15    public String nacionalidad;
16    public String sobrenombre;
17    public long telefono;
18    public String rol;
19    public long contadorPaquete=0;
20    public ImageIcon foto;
21
22    public String getcorreo() {
23        return correo;
24    }
25    public String getnombre() {
26        return nombre;
27    }
28    public String getapellido() {
29        return apellido;
30    }
31    public String getpassword() {
32        return password;
33    }
34    public long getDPI() {
35        return dpi;
36    }
37    public String getfechaNacimiento() {
38
39    }
40    public String getgenero() {
41        return genero;
42    }
43    public String getnacionalidad() {
44        return nacionalidad;
45    }
46    public String getsobrenombre() {
47        return sobrenombre;
48    }
49    public long gettelefono() {
50        return telefono;
51    }
52    public String getrol() {
53        return rol;
54    }
55    public long getcontadorPaquete() {
56        return contadorPaquete;
57    }
58    public ImageIcon getfoto() {
59        return foto;
60    }
61    public void setcontadorPaquete(long contadorPaquete) {
62        this.contadorPaquete=contadorPaquete;
63    }
64    public void setfoto(ImageIcon foto) {
65        this.foto=foto;
66    }
67    public void setpassword(String password) {
68        this.password=password;
69    }
70 }
```

Figura 2: Clase Objeto Usuario

3. Función Registrar

En esta función se registró un nuevo objeto con sus respectivos atributos. Para realizar esto, se deben validar que los campos de texto estén llenos, si no se muestra un mensaje que solicita al usuario completar todos los campos.

Luego de esto, se valida que el objeto que registraremos no exista dentro del ArrayList, si el objeto existe (si uno de los atributos que debe ser único para cada objeto coincide con uno que esté registrado) se muestra en pantalla que el objeto ya ha sido registrado.

Si el objeto no existe, se obtiene el texto que fue ingresado en las cajas de texto y se almacenan en distintas variables respectivamente, luego se crea una instancia hacia el objeto que crearemos, igualamos los atributos del objeto a las variables creadas según corresponda y por ultimo agregamos dicho objeto a la lista correspondiente y mostramos un mensaje de confirmación.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
  
    if(correoTXT.getText().isEmpty() || nombreTXT.getText().isEmpty() || direccionTXT.getText().isEmpty() || nitTXT.getText().isEmpty()){  
        JOptionPane.showMessageDialog(parentComponent: null, message: "Campos Vacios, Debe llenar todos los campos.", title: "REGISTRO DE DATOS DE FACTURACIÓN")  
    }else{  
        boolean existe=false;  
        for (int i=0; i<AppState.listaDatosFacturacion.size(); i++){  
            if (correoTXT.getText().equals( anObject: AppState.listaDatosFacturacion.get( index: i ).getcorreo())==true && direccionTXT.getText().equals( anObject  
                existe=true;  
                break;  
            )  
        }  
        if(existe){  
            JOptionPane.showMessageDialog(parentComponent: null, message: "Esta Dirección Ya Ha Sido Registrado a Este Usuario. Intenta con una dirección nue  
        }else{  
            String correo=correoTXT.getText();  
            String nombreCompleto=nombreTXT.getText();  
            String direccion=direccionTXT.getText();  
            String nit=nitTXT.getText();  
            DatosFacturacion nuevoDato = new DatosFacturacion();  
            nuevoDato.correo=correo;  
            nuevoDato.nombreCompleto=nombreCompleto;  
            nuevoDato.direccion=direccion;  
            nuevoDato.nit=nit;  
            AppState.listaDatosFacturacion.add( e: nuevoDato);  
            JOptionPane.showMessageDialog(parentComponent: null, message: "Datos Registrados Correctamente", title: "REGISTRO DE DATOS DE FACTURACIÓN", messageTy  
            nombreTXT.setText( : null);  
            direccionTXT.setText( : null);  
            nitTXT.setText( : null);  
            for(int i = 0; i<AppState.listaDatosFacturacion.size(); i++){  
                System.out.println(AppState.listaDatosFacturacion.get( index: i ).getcorreo()+" - "+AppState.listaDatosFacturacion.get( index: i ).getnombreC  
            }  
            System.out.println( x: "-----");  
        }  
    }  
}
```

Figura 3: Función Registrar Nuevos Datos De Facturación

4. Función Modificar

En esta función modifica los atributos de un objeto registrado. Para realizar esto, se deben validar que los campos de texto estén llenos, si no se muestra un mensaje que solicita al usuario completar todos los campos necesarios para la modificación.

Luego de esto, se valida que el objeto que modificaremos exista dentro del ArrayList, si el objeto existe (si uno de los atributos que debe ser único para cada objeto y no puede ser modificado coincide con uno que esté registrado), se obtiene el texto que fue ingresado en las cajas de texto y se almacenan en distintas variables respectivamente, luego se recorre el ArrayList para buscar el atributo único de cada objeto y al encontrarlo, con los métodos set, se envían los nuevos atributos a modificar por medio de las variables creadas según corresponda. Por ultimo mostramos un mensaje que el objeto fue modificado.

Si el objeto no existe (si uno de los atributos que debe ser único para cada objeto y no puede ser modificado no coincide con uno que esté registrado) se muestra en pantalla un mensaje indicando al usuario que el atributo único del objeto no coincide con el registrado en el ArrayList.

```
private void jButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here
    if(nombreActualTXT.getText().isEmpty()||nombreNuevoTXT.getText().isEmpty()||direccionNuevoTXT.getText().isEmpty()||nitNuevoTXT.getText().isEmpty())
        JOptionPane.showMessageDialog(parentComponent, null, "Campos Vacios, Debe llenar todos los campos modificación.", "REGISTRO DE DATOS DE FACTURACION");
    else{
        boolean existe=false;
        for (int i=0;i<AppState.listaDatosFacturacion.size();i++){
            if (correoTXT.getText().equals(AppState.listaDatosFacturacion.get(i).getcorreo())&& nombreActualTXT.getText().equals(AppState.listaDatosFacturacion.get(i).getnombreCompleto())){
                existe=true;
                break;
            }
        }
        if(existe){
            for (int i=0;i<AppState.listaDatosFacturacion.size();i++){
                if (correoTXT.getText().equals(AppState.listaDatosFacturacion.get(i).getcorreo())&& nombreActualTXT.getText().equals(AppState.listaDatosFacturacion.get(i).getnombreCompleto())){
                    AppState.listaDatosFacturacion.get(i).setnombreCompleto(nombreNuevoTXT.getText());
                    AppState.listaDatosFacturacion.get(i).setdireccion(direccionNuevoTXT.getText());
                    AppState.listaDatosFacturacion.get(i).setnit(nitNuevoTXT.getText());
                    break;
                }
            }
            for(int i = 0; i<AppState.listaDatosFacturacion.size(); i++){
                System.out.println(AppState.listaDatosFacturacion.get(i).getcorreo()+" - "+AppState.listaDatosFacturacion.get(i).getnombreCompleto());
            }
            System.out.println("-----");
            JOptionPane.showMessageDialog(parentComponent, null, "Datos de Facturación Modificados Correctamente.", "REGISTRO DE DATOS DE FACTURACION");
            nombreActualTXT.setText("");
            nombreNuevoTXT.setText("");
            direccionNuevoTXT.setText("");
            nitNuevoTXT.setText("");
        }else{
            JOptionPane.showMessageDialog(parentComponent, null, "El Nombre Actual no Coincide con el Nombre Registrado.", "REGISTRO DE DATOS DE FACTURACION");
        }
    }
}
```

Figura 4: Función Modificar Datos De Facturación

5. Función Eliminar

Esta función elimina un objeto que fue registrado anteriormente. Para realizar esto, se deben validar que los campos de texto estén llenos, si no se muestra un mensaje que solicita al usuario completar todos los campos necesarios para la eliminación.

Luego de esto, se valida que el objeto que eliminaremos exista dentro del ArrayList, si el objeto existe (si uno de los atributos que debe ser único para cada objeto coincide con uno que esté registrado) se recorre el ArrayList para buscar el atributo único de cada objeto y al encontrarlo, con el método remove, se elimina el objeto del ArrayList. Por ultimo mostramos un mensaje que el objeto fue eliminado.

Si el objeto no existe (si uno de los atributos que debe ser único para cada objeto no coincide con uno que esté registrado) se muestra en pantalla un mensaje indicando al usuario que el atributo único del objeto no coincide con el registrado en el ArrayList, por ende no puede ser eliminado.

```
private void jButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here
    if(nombreEliminarTXT.getText().isEmpty()){
        JOptionPane.showMessageDialog(parentComponent, null, "Campos Vacios, Debe llenar todos los campos eliminación.", "REGISTRO DE DATOS DE FACTURACION");
    }
    else{
        boolean existe=false;
        for (int i=0;i<AppState.listaDatosFacturacion.size();i++){
            if (correoTXT.getText().equals(AppState.listaDatosFacturacion.get(i).getcorreo())&& nombreEliminarTXT.getText().equals(AppState.listaDatosFacturacion.get(i).getnombreCompleto())){
                existe=true;
                break;
            }
        }
        if(existe){
            for (int i=0;i<AppState.listaDatosFacturacion.size();i++){
                if (correoTXT.getText().equals(AppState.listaDatosFacturacion.get(i).getcorreo())&& nombreEliminarTXT.getText().equals(AppState.listaDatosFacturacion.get(i).getnombreCompleto())){
                    AppState.listaDatosFacturacion.remove(i);
                    break;
                }
            }
            for(int i = 0; i<AppState.listaDatosFacturacion.size(); i++){
                System.out.println(AppState.listaDatosFacturacion.get(i).getcorreo()+" - "+AppState.listaDatosFacturacion.get(i).getnombreCompleto());
            }
            System.out.println("-----");
            JOptionPane.showMessageDialog(parentComponent, null, "Datos de Facturación Eliminados Correctamente.", "REGISTRO DE DATOS DE FACTURACION");
            nombreEliminarTXT.setText("");
        }else{
            JOptionPane.showMessageDialog(parentComponent, null, "El Nombre no Coincide con el Nombre Registrado.", "REGISTRO DE DATOS DE FACTURACION");
        }
    }
}
```

Figura 5: Función Eliminar Datos De Facturación

6. Función Llenar Tabla

Esta función llena una tabla con los atributos de un objeto deseables que se encuentran registrados en un ArrayList. Para realiza esto debemos crear una variable en la cual se tenga un atributo único para cada objeto con la cual condicionaremos el llenado de la tabla.

Luego se crea un modelo para la tabla y se definen las columnas que tendrá nuestra tabla, además de crear una subfunción en donde se restrinja que la tabla no pueda ser editada por el usuario. Se para el modelo creado a la tabla que se agregó.

Por último se recorre el ArrayList, dentro del corrido se busca el atributo único para cada objeto obtenido anteriormente y se instancia el objeto a su clase. Con dicha instancia se obtiene los atributos deseables de cada objeto y se envían a la tabla.

```
String correo = correoTXT.getText();
DefaultTableModel model = new DefaultTableModel(new String[]{"Código de paquete", "Tipo de servicio", "Destinatario", "Total de envío", "Tipo de pago"})
{
    @Override
    public boolean isCellEditable(int row, int column) {
        if (column==5){
            return true;
        }else{
            return false;
        }
    }
};
jTable1.setModel( dataModel: model);
TableModel ModeloDato = jTable1.getModel();
for (int i = 0; i < AppState.listaDatos_CotCompra.size(); i++) {
    if(correo.equals( anObject:AppState.listaDatos_CotCompra.get(index:i).getcorreo())==true){
        Datos_CotCompra nuevoEnvio = AppState.listaDatos_CotCompra.get(index:i);
        ModeloDato.setValueAt( aValue:nuevoEnvio.getcodigoPaquete(), rowIndex:i, columnIndex: 0);
        ModeloDato.setValueAt( aValue:nuevoEnvio.gettipoServicio(), rowIndex:i, columnIndex: 1);
        ModeloDato.setValueAt( aValue:nuevoEnvio.getdestinatario(), rowIndex:i, columnIndex: 2);
        ModeloDato.setValueAt( aValue:nuevoEnvio.gettotalPago(), rowIndex:i, columnIndex: 3);
        ModeloDato.setValueAt( aValue:nuevoEnvio.gettipoPago(), rowIndex:i, columnIndex: 4);
    }
}
```

Figura 6: Función Llenar Tabla de Datos de Datos Cotización-Compra

7. Función Contar Caracteres Contraseña

Esta función valida si la contraseña contiene letras mayúsculas, minúsculos, números, caracteres especiales y una longitud de 8 caracteres. Para realizar esto, pero se debe crear una variable la cual será igualada al contenido que tenga la caja de texto donde se ingresó la contraseña.

Luego de esto, se crean 5 constantes del tipo entero que serán utilizadas para determinar que la contraseña cumpla con los requisitos antes mencionados y 4 variables del tipo entero que serán utilizadas para contar la cantidad de letras mayúsculas, minúsculas, números y caracteres especiales que la contraseña ingresada tiene.

Se realiza un recorrido por cada carácter de la contraseña y se condiciona que cada vez que aparezca una letra mayúscula, el contador de las letras mayúsculas aumente en 1, así sucesivamente para todas las condiciones.

Por último se verifica que la contraseña cumpla con la cantidad mínima de letras mayúsculas, minúsculas, números, caracteres especiales y cantidad de caracteres necesarios y se muestra un mensaje al usuario si su contraseña es válida o no.

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String password2 = jTextField1.getText();
    final int MAX=8, MIN_Uppercase=1, MIN_Lowercase=1, NUM_Digits=1, Special=1;
    int uppercaseCounter=0, lowercaseCounter=0, digitCounter=0, specialCounter=0;
    for (int i=0; i < password2.length(); i++) {
        char c = password2.charAt(i);
        if(Character.isUpperCase(c))
            uppercaseCounter++;
        else if(Character.isLowerCase(c))
            lowercaseCounter++;
        else if(Character.isDigit(c))
            digitCounter++;
        if(c>=33&&c<=46||c==64){
            specialCounter++;
        }
    }
    if (password2.length()>= MAX && uppercaseCounter>= MIN_Uppercase && lowercaseCounter>= MIN_Lowercase && digitCounter>= NUM_Digits && specialCounter>= Special)
        JOptionPane.showMessageDialog(parentComponent: null, message: "La Contraseña Es Correcta.", title: "REGISTRAR USUARIO", messageType: JOptionPane.WARNING_MESSAGE);
    else{
        JOptionPane.showMessageDialog(parentComponent: null, message: "La Contraseña debe contener letras mayusculas, minusculas, números y caracteres especiales.", title: "REGISTRAR USUARIO", messageType: JOptionPane.WARNING_MESSAGE);
    }
}

```

Figura 7: Función Contar Caracteres de la Contraseña

8. Función Colocar Fotografía en un JLabel

Esta función llena componente JLabel con una fotografía que el usuario seleccione. Para realizar esto, se crea una variable del tipo JFileChooser, este objeto servirá para abrir una ventana en donde aparezca un tipo explorador de archivos con el que se podrá seleccionar la foto.

Luego de esto se crea una variable del tipo File para almacenar el archivo seleccionado, se envía la ruta del archivo a una caja de texto por medio de método set. Se crea una variable del tipo Image para obtener la imagen seleccionada. Se modifica la escala de dicha imagen.

Por último, se agrega la imagen al JLabel por medio del método setIcon.

```

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser archivo = new JFileChooser();
    int ventana= archivo.showOpenDialog( parent: null);
    if (ventana == JFileChooser.APPROVE_OPTION){
        File file= archivo.getSelectedFile();
        rutaTXT.setText( t: String.valueOf( obj: file));
        Image foto = getToolkit().getImage( filename: rutaTXT.getText());
        foto = foto.getScaledInstance( width: fotoLBL.getWidth(), height: fotoLBL.getHeight(), hints: Image.SCALE_DEFAULT);
        fotoLBL.setIcon(new ImageIcon( image: foto));
    }
}

```

Figura 8: Función Colocar Imagen en un JLabel

9. Función Ordenamiento de Forma Descendente

Esta función ordena los objetos de una lista de forma descendente según un atributo. Para esto se debe realizar un primer recorrido por cada objeto de la lista, se crea una variable del tipo Objeto para almacenar el objeto analizado en el primer recorrido de forma temporal. Para el segundo recorrido, se iterará de forma que el recorrido se realice en el objeto que se encuentra continuo al objeto analizado en el primer recorrido. Luego se compara si el objeto analizado en el primer recorrido es menor al objeto analizado en el segundo recorrido.

Si esta condición se cumple, el objeto analizado en el segundo recorrido será almacenado en el objeto temporal, se enviará al ArrayList el objeto de primer recorrido en la segunda posición y el objeto temporal en la primera posición. Esto se repetirá para cada objeto en el ArrayList.

```

public static void burbujadescendenteRegion(ArrayList<Region_Precio> listaRegion) {
    for(int i=0;i<AppState.listaRegion.size();i++){
        Region_Precio temporal;
        for(int j=i+1;j<AppState.listaRegion.size();j++){
            if(AppState.listaRegion.get(index:i).getcontador()<AppState.listaRegion.get(index:j).getcontador()){
                temporal=AppState.listaRegion.get(index:j);
                AppState.listaRegion.set(index:j, element:AppState.listaRegion.get(index:i));
                AppState.listaRegion.set(index:i, element:temporal);
            }
        }
    }
}

```

Figura 9: Función Ordenamiento de Forma Descendente

10. Función Generar Código De Caracteres

Esta función genera una cadena de caracteres de forma aleatoria. Para realizar esto se deben declarar una constante del tipo String en estarán todas las posibles letras que el código puede tomar y una constante del tipo int que será igualada a la cantidad de letras que el código tomará de la constante declarada anteriormente.

Se declara una variable del tipo StringBuilder para almacenar los caracteres, con un ciclo for, se realiza un recorrido con la cantidad de letras que el código puede tomar, se declara una variable random para obtener una letra aleatoria, se obtiene la posición del carácter, se convierte dicha posición a una variable carácter y finalmente se agrega el carácter a la variable del tipo StringBuilder.

Par los números, se crea una variable del tipo int para almacenar los números, se declara una viable random para obtener números aleatorios del 0 hasta el límite que se declara.

Por último, se concatena todas las variables obtenidas para generar el código de forma aleatorio en una variable del tipo String y se retorna dicha variable.

```

public static String generarCodigo() {
    //LETRAS ALEATORIAS
    final String cadena="QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm";
    final int longi=3;
    StringBuilder sb1 = new StringBuilder();
    for (int i = 0;i<longi; i++) {
        double aleatorio1=Math.random()*cadena.length();
        int posicion1=(int)aleatorio1;
        char letral =cadena.charAt(index:posicion1);
        sb1.append(c:letral);
    }
    //NUMEROS ALEATORIOS
    int aleatorio1=0;
    Random codigoRandom = new Random();
    aleatorio1=(int) (codigoRandom.nextDouble()*10);
    //CODIGO
    String codigo="IPC1A_"+sb1.toString()+aleatorio1;
    //RETORNO
    return codigo;
}

```

Figura 10: Función Generar Código De Caracteres

11. Función Generar un Archivo del Tipo HTML

Esta función genera un archivo HTML con el contenido que se desea. Para realizar esto, se declara una variable del tipo `StringBuilder` con la cual, por medio de la función `append`, se agregará todas las etiquetas de HTML que se desean. También se podrán agregar variables de cualquier tipo con forme se necesitan.

Para generar el archivo, primero se declaran las variables con su respectivo contenido, se llama a la función creada anteriormente, se crea un variable del tipo `archivo` en donde se almacenará la dirección del nuevo archivo, por ultimo se creará un nuevo archivo con la extensión `.html` y se mostrará un mensaje que el archivo fue creado exitosamente.

```
public static String generarFacturaHTML(String numerofactura,String codigoPaquete,String origen,String destino,String nit,String tipoPago,String
    StringBuilder sb = new StringBuilder();
    sb.append( str: "<font font face='Verdana'" );
    //DATOS FACTURA
    sb.append( str: "<h5>" ).append( "Número de Factura: "+numerofactura ).append( str: "</h5>" );
    sb.append( str: "<h5>" ).append( "Código de Paquete: "+codigoPaquete ).append( str: "</h5>" );
    sb.append( str: "<h5>" ).append( "Origen: "+origen ).append( str: "</h5>" );
    sb.append( str: "<h5>" ).append( "Destino: "+destino ).append( str: "</h5>" );
    sb.append( str: "<h5>" ).append( "NIT: "+nit ).append( str: "</h5>" );
    sb.append( str: "<h5>" ).append( "Tipo De Pago: "+tipoPago ).append( str: "</h5>" );
    sb.append( str: "<h5>" ).append( "Nombre: "+nombre ).append( str: "</h5>" );
    // SE CREA LA TABLA
    sb.append( str: "<table border='1'" );
    // FILA 1 ENCABEZADO
    sb.append( str: "<tr>" );
    for (String encabezado : Encabezado) {
        sb.append( str: "<th>" ).append( str: encabezado ).append( str: "</th>" );
    }
    sb.append( str: "</tr>" );
    // FILA 2 DATOS
    sb.append( str: "<tr>" );
    sb.append( str: "<td>" );
    sb.append( str: "<td>" ).append( str: numeroPaquete ).append( str: "</td>" );
    sb.append( str: "<td>" ).append( str: tamanoPaquete ).append( str: "</td>" );
    sb.append( str: "<td>" ).append( str: totalPago ).append( str: "</td>" );
    sb.append( str: "</tr>" );
    //SE CIERRA LA TABLA
    sb.append( str: "</table>" );
    sb.append( str: "</font>" );
    return sb.toString();
}
```

Figura 11: Función Generar un Archivo del Tipo HTML

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    String[] Encabezado = {"Número De Paquetes","Tamaño Del Paquete", "Total De Pago"};
    String numerofactural=numeroFactura;
    String codigoPaquete1=codigoPaquete;
    String origen1=origen;
    String destino1=destino;
    String nit1=String.valueOf( l:nit);
    String tipoPagol=tipoPago;
    String tamanoPaquete1=String.valueOf( l:tamanoPaquete);
    String numeroPaquete1=String.valueOf( l:numeroPaquete);
    String total1=String.valueOf( d:TotalNeto);
    String nombrel=destinatario;

    String tablaHtml = generarFacturaHTML( numerofactura:numerofactural, codigoPaquete:codigoPaquete1, origen:origen1, destino:destino1, nit:nit1, tipoPagol:
    // Crea la carpeta de reportes si no existe
    File carpeta = new File( pathname: "C:/Users/manue/Downloads/" );
    if (!carpeta.exists()) {
        carpeta.mkdirs();
    }

    // Escribe el archivo .html dentro de la carpeta
    try {
        FileWriter fileWriter = new FileWriter( "C:/Users/manue/Downloads/Factura"+numeroFactura+".html" );
        fileWriter.write( str: tablaHtml );
        JOptionPane.showMessageDialog( parentComponent: null, message: "Factura Descargada.", title: "COTIZACIÓN DE PAQUETES Y COMPRA", messageType: JOpt
        fileWriter.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figura 12: Generar un Archivo del Tipo HTML