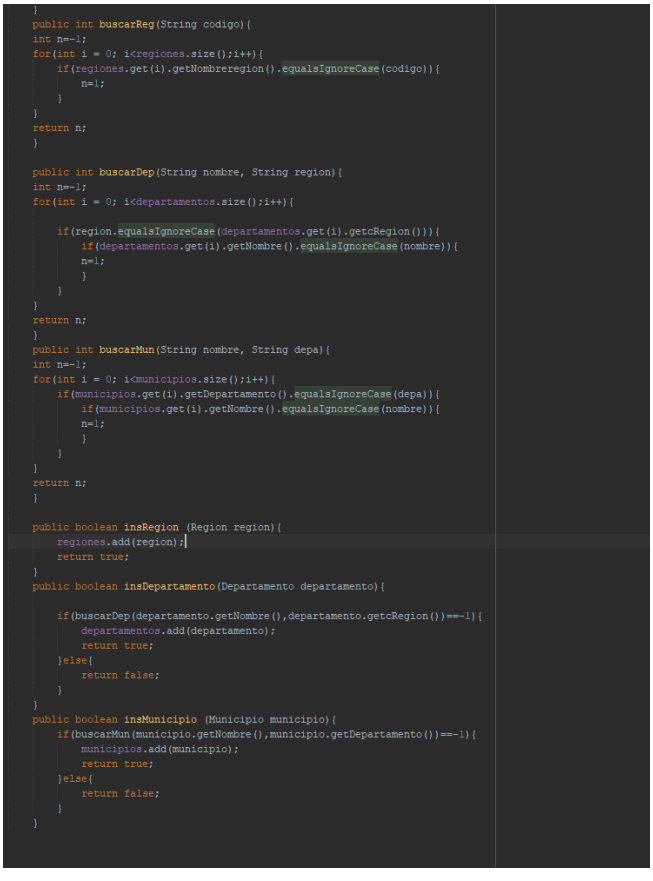


```
}
public int buscarReg(String codigo){
int n=-1;
for(int i = 0; i<regiones.size();i++){
    if(regiones.get(i).getNombreregion().equalsIgnoreCase(codigo)){
        n=1;
    }
}
return n;
}

public int buscarDep(String nombre, String region){
int n=-1;
for(int i = 0; i<departamentos.size();i++){

    if(region.equalsIgnoreCase(departamentos.get(i).getcRegion())){
        if(departamentos.get(i).getNombre().equalsIgnoreCase(nombre)){
        n=1;
        }
    }
}
return n;
}
public int buscarMun(String nombre, String depa){
int n=-1;
for(int i = 0; i<municipios.size();i++){
    if(municipios.get(i).getDepartamento().equalsIgnoreCase(depa)){
        if(municipios.get(i).getNombre().equalsIgnoreCase(nombre)){
        n=1;
        }
    }
}
return n;
}

public boolean insRegion (Region region){
    regiones.add(region);
    return true;
}
public boolean insDepartamento(Departamento departamento){

    if(buscarDep(departamento.getNombre(),departamento.getcRegion())==-1){
        departamentos.add(departamento);
        return true;
    }else{
        return false;
    }
}
public boolean insMunicipio (Municipio municipio){
    if(buscarMun(municipio.getNombre(),municipio.getDepartamento())==-1){
        municipios.add(municipio);
        return true;
    }else{
        return false;
    }
}
```

**La primera función, buscarReg, busca una región por su nombre, representado por el parámetro codigo, en la lista regiones. Devuelve 1 si la región se encuentra en la lista y -1 si no se encuentra.**

**La segunda función, buscarDep, busca un departamento por su nombre, representado por el parámetro nombre, y por la región a la que pertenece, representada por el parámetro region, en la lista departamentos. Devuelve 1 si el departamento se encuentra en la lista y -1 si no se encuentra.**

**La tercera función, buscarMun, busca un municipio por su nombre, representado por el parámetro nombre, y por el departamento al que pertenece, representado por el parámetro depa, en la lista municipios. Devuelve 1 si el municipio se encuentra en la lista y -1 si no se encuentra.**

**Las tres funciones anteriores son funciones de búsqueda que recorren las listas en busca de objetos que coincidan con los criterios de búsqueda.**

**Las siguientes tres funciones, insRegion, insDepartamento e insMunicipio, son funciones de inserción que agregan nuevos objetos a las listas regiones, departamentos y municipios, respectivamente. La función insRegion agrega una nueva región, mientras que la función insDepartamento agrega un nuevo departamento si no existe otro departamento con el mismo nombre y la misma región. La función insMunicipio agrega un nuevo municipio si no existe otro municipio con el mismo nombre en el mismo departamento.**

```
private boolean validarContraseña(String contraseña) {
    boolean tieneNumeros = false;
    boolean tieneCaracteresEspeciales = false;
    boolean tieneMayusculas = false;
    boolean tieneMinusculas = false;

        for (int i = 0; i < contraseña.length(); i++) {
            char c = contraseña.charAt(i);
        if (Character.isDigit(c)) {
```

```
            tieneNumeros = true;

            }else if (Character.isUpperCase(c)) {
            tieneMayusculas = true;

            }else if (Character.isLowerCase(c)) {
            tieneMinusculas = true;

            } else {
            tieneCaracteresEspeciales = true;
        }
    }
    return tieneNumeros && tieneCaracteresEspeciales && tieneMayusculas && tieneMinusculas;
}
```

**define una función llamada validarContraseña que toma una cadena de caracteres contraseña como parámetro y verifica si cumple con ciertos requisitos de seguridad. La función devuelve true si la contraseña es válida y false si no lo es.**

**La función utiliza cuatro variables booleanas para realizar un seguimiento de si la contraseña contiene dígitos, caracteres especiales, mayúsculas y minúsculas. Luego, un bucle for recorre cada carácter de la cadena contraseña y verifica si es un dígito, una letra mayúscula, una letra minúscula o un carácter especial. Si el carácter actual cumple con uno de estos criterios, la variable booleana correspondiente se establece en true.**

**Al final del bucle, se verifica si las cuatro variables booleanas se han establecido en true. Si es así, significa que la contraseña cumple con los requisitos de seguridad y se devuelve true. Si no, significa que la contraseña no cumple con los requisitos y se devuelve false.**

```
{
    if(Ventanas.usuarios.Admin.Logica.tamañoR()>0){

    }else{
        Ventanas.usuarios.Admin.Logica.Ingresar();
    }
     //ASIGANCION VARIABLES ----------------------------------------------------
     int a = 0, b = 0;
     String contraseña = InsertarContraseña.getText();
     String correo = InsertarCorreoElectronico.getText();
     if(contraseña.isEmpty()){ a = 1; }
     if(correo.isEmpty()){ b = 1; }
     // VERIFICANDO CASILLAS VACIAS ---------------------------------------------
     if(a!=1 && b!=1){
         // ACCESO LOGIN ADMIN ---------------------------------------------------
         if(correo.equalsIgnoreCase("hola")&&
                 contraseña.equals("hola")){
                 System.out.println("Personal ");
                         opcionescliente cliente = new opcionescliente();
                         cliente.setVisible(true);
                         cliente.setCorreoU(correo);
                         String rol = "PERSONAL";
                         cliente.setRol(rol);
                         cliente.nombre();
                         this.dispose();

         }
         else if(correo.equalsIgnoreCase("ipc1_202200100@ipc1delivery.com")&&
                 contraseña.equals("202200100"))
         {
             // ABRIENDO PAGINA ADMIN ---------------------------------
             opciones2 admin = new opciones2();
             this.dispose();
             admin.nombre(correo);
             admin.setVisible(true);
             admin.setCorreoU(correo);


         }else
         {
             // ACCESO LOGIN USUARIOS----------------------------------------------
             int login = Logicas.login(contraseña, correo);
             Ventanas.usuarios.Admin.Logica.Ingresar();
             if(login >=0){
                 switch(Logicas.getrol(login)){

                     case "Usuario Individual":
                         System.out.println("individual ");
                         opcionescliente cliente = new opcionescliente();
                         cliente.setVisible(true);
                         cliente.setCorreoU(correo);
                         String rol = "Usuario individual";
                         cliente.setRol(rol);
                         cliente.nombre();
                         this.dispose();
                         break;

                     case "Usuario Empresarial":
                         System.out.println("empresarial");
                         opcionescliente empresarial = new opcionescliente();
                         empresarial.setVisible(true);
                         empresarial.setCorreoU(correo);
                         String rols = "Usuario empresarial";
                         empresarial.setRol(rols);
                         empresarial.nombre();
                         this.dispose();

                         break;
```

```
                case "Kiosko":
                    System.out.println("kiosco");
                    elegirkiosco dis = new elegirkiosco();
                    dis.iniciar();
                    dis.setCorreoU(correo);
                    dis.setVisible(true);
                    this.dispose();
                    break;

                }
            }else{
                JOptionPane.showMessageDialog(this, "Credenciales incorrectas");
            }
        }
    }else{
        JOptionPane.showMessageDialog(this, "Hay campos vacios");
    }
    // ------------------------------------------------------------------
}
```

**El código se encarga de validar si los campos de contraseña y correo electrónico están llenos y, si no lo están, verificar si el usuario ha ingresado credenciales de administrador o de usuario individual / empresarial / kiosko.**

**Primero, el código verifica si ya hay usuarios registrados . Si hay, no hace nada. Si no hay usuarios registrados, la función "Ingresar" se llama desde la lógica de administrador para ingresar los usuarios de prueba.**

**Luego, el código obtiene la información ingresada en los campos de contraseña y correo electrónico y verifica si ambos campos están vacíos. Si uno o ambos campos están vacíos, se muestra un mensaje de error.**

**Si ambos campos están llenos, el código verifica si el usuario ha ingresado credenciales de administrador o de usuario individual / empresarial / kiosko. Si el usuario ingresa las credenciales de administrador, se abre una interfaz de opciones de administrador. Si el usuario ingresa credenciales de usuario individual / empresarial / kiosko, se abre una interfaz de opciones correspondiente a ese tipo de usuario.**

**Si las credenciales ingresadas no son correctas, se muestra un mensaje de error.**

```
public boolean referencias (int abrir)
{
    boolean si= false;

    if(abrir==0)
    {
        int ventana;
        ventana=JOptionPane.showConfirmDialog(this, "Desea agregar sus datos de facturacion");

        if(ventana==JOptionPane.NO_OPTION)
        {
            JOptionPane.showMessageDialog(this, "No agrego datos de facturacion");

        }else if(ventana==JOptionPane.YES_OPTION)
        {
            registrofactura o = new registrofactura();
            o.setVisible(true);
            o.setCorreoU(correo);
            o.botones(false);

        }else{
            JOptionPane.showMessageDialog(this, "No agrego datos de facturacion");
        }
        si=false;

    }else
    {
         Set<String>nombresRegion = new HashSet<>();
         refers.removeAllElements();
            for(int a = 0; a<Logics.tamaño();a++)
            {
                String nombre = Logics.completa(a).getCodigo();
                String parent = Logics.completa(a).getUsuario();
```

```
                if(parent.equalsIgnoreCase(correo))
                {
                    if(!nombresRegion.contains(nombre))
                    {
                        refers.addElement(nombre);
                        nombresRegion.add(nombre);
                    }
                }
            }
            si= true;
        }
        return si;
    }
```

**Si el valor del parámetro "abrir" es igual a 0, se muestra un cuadro de diálogo con una pregunta para agregar datos de facturación. Si la respuesta es "NO", se muestra un mensaje de que no se agregaron datos de facturación y se establece el valor de retorno a "false". Si la respuesta es "SI", se muestra otra ventana llamada "registrofactura" y se establece el valor de retorno a "false".**

**Si el valor del parámetro "abrir" es diferente de 0, se crea un conjunto llamado "nombresRegion" y se eliminan todos los elementos del vector "refers". Luego, se recorre un ciclo "for" a través de los datos almacenados en la variable "Logics" y se verifica si el usuario asociado con los datos es igual al correo almacenado en la variable "correo". Si es así, se verifica si el nombre no está ya en el conjunto "nombresRegion", y si es así, se agrega el nombre al vector "refers" y se agrega el nombre al conjunto "nombresRegion". Finalmente, se establece el valor de retorno a "true".**

```
public void pagoTarjeta (int abrir)
{
    registrotarjetas o = new registrotarjetas();

    if(abrir==0)
    {
        int ventana;
        ventana=JOptionPane.showConfirmDialog(this, "Desea agregar tarjeta de credito/debito");

        if(ventana==JOptionPane.NO_OPTION)
        {
            JOptionPane.showMessageDialog(this, "No agrego tarjeta nueva");
        }else if(ventana==JOptionPane.YES_OPTION)
        {
            System.out.println("Agregando tarjetas");
            o.setVisible(true);
            o.setCorreoU(correo);
            o.boton(true);
            o.iniciarTarjeta();
            o.cambio();

        }else{
            JOptionPane.showMessageDialog(this, "No agrego tarjetas nuevas");

        }

    }else
    {
        o.setVisible(true);
        o.setCorreoU(correo);
        o.iniciarTarjeta();
        o.boton(false);
    }
}
//// ------------------------------------------------------------------------
```

**Este código implementa un método llamado pagoTarjeta que recibe un parámetro entero llamado abrir. La función muestra una ventana de diálogo con opciones para agregar una tarjeta de crédito o débito.**

**Si el parámetro abrir es igual a 0, se muestra un mensaje de confirmación para agregar una nueva tarjeta de crédito o débito. Si se selecciona la opción "Sí", se abre una nueva ventana de registro de tarjeta de crédito/débito (registrotarjetas) donde se puede agregar una tarjeta. Si se selecciona la opción "No", se muestra un mensaje de que no se agregó una tarjeta.**

**Si el parámetro abrir es diferente de 0, se abre una nueva ventana de registro de tarjeta de crédito/débito (registrotarjetas) con los campos de registro precargados con los detalles de la tarjeta existente. En este caso, el botón de guardar está deshabilitado y solo se puede ver la información de la tarjeta.**

```
public int envio(String opcion){
 int x = 0;
    switch (opcion){
        case "Estandar":
            showTxt.setText("Entrega de 3 a 5 días habiles");
            x= 1;
            break;
        case "Especial":
            showTxt.setText("Entrega inmediata 1 o 2 días habiles");
            x=2;
            break;
    }
    return x;
}
```

**El método envio toma como entrada una cadena de texto opcion que representa la opción de envío seleccionada por el usuario. El método utiliza una estructura de control switch para evaluar la opción seleccionada y, según el caso, establecer el tiempo de entrega y devolver un valor entero correspondiente.**

**Si la opción seleccionada es "Estandar", se establece el mensaje "Entrega de 3 a 5 días hábiles" en un componente de la interfaz de usuario llamado showTxt y se devuelve el valor entero 1. Si la opción seleccionada es "Especial", se establece el mensaje "Entrega inmediata 1 o 2 días hábiles" en el mismo componente de la interfaz de usuario y se devuelve el valor entero 2.**

```
public void Cotizar(){
    String dep = destinoDEP.getSelectedItem().toString();
    destinoM(dep);
    System.out.println("COTIZAR M: "+municipio.getSelectedItem().toString());
    String opcion = btnEnvio.getSelectedItem().toString();
    int eleccion = envio(opcion);
    String dpt = destinoD.getSelectedItem().toString();
    double x =precio(eleccion, dpt);
    setPrecioCotizar(x);
}
```

**En la función Cotizar() se obtiene el departamento y municipio de destino seleccionados por el usuario, y se llama a la función destinoM(dep) para obtener y mostrar en pantalla los municipios correspondientes al departamento seleccionado.**

**Luego, se obtiene la opción de envío seleccionada por el usuario, y se llama a la función envio(opcion) para obtener el tiempo de entrega estimado en días hábiles y asignar una variable numérica correspondiente.**

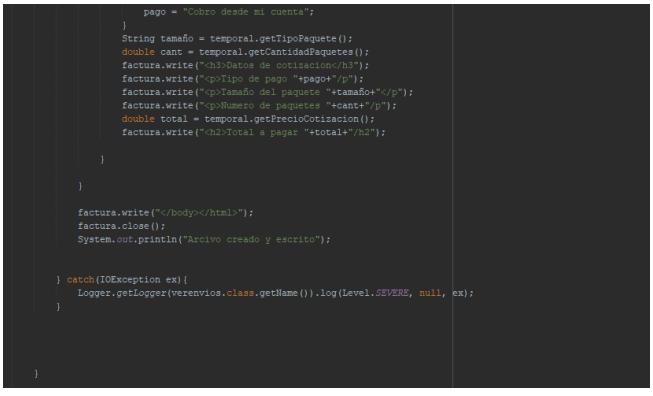
**Se obtiene el departamento de destino seleccionado y se llama a la función precio(eleccion, dpt) para obtener el precio del envío, el cual se asigna a la variable x.**

**Finalmente, se llama a la función setPrecioCotizar(x) para mostrar el precio obtenido en pantalla.**

```
public String agregado (){
    String tipo = "";
    //Comprobando que haya seleccionado una opcion
    if(rdEntrega.isSelected() || rdTarjeta.isSelected())
    {   // Comprobando que no hayan datos vacios
        if(accionEnvio())
        {
            boolean s1, s2;
            if(rdTarjeta.isSelected())
            {

                //Comprobando tarjetas
                s1=true;
                if(Logics.tamaño()==0){
                    s2=false;
                    JOptionPane.showMessageDialog(this, "Agregue datos de facturacion");
                    System.out.println("No puede facturar");
                }else{
                    s2=referencias(1);
                }
                if(s1==true && s2==true){
                    tipo = "TARJETA";
                    setPago(tipo);
                    btnValidar.setEnabled(true);
                    mastr.setEnabled(true);
                    setAñadir(0);
                    JOptionPane.showMessageDialog(this, "Valide su tarjeta");
                }

            }else
            {
                s1=true;
                setAñadir(5);
                //comprobando datos de facturacion
                if(Logics.tamaño()==0)
                {
                   s2= false;
                   JOptionPane.showMessageDialog(this, "Agregue datos de facturacion");
                    System.out.println("No puede facturar");
                }else{
                   s2=referencias(1);
                }
                if(s1==true && s2==true){
                    tipo = "EFECTIVO";
                    setPago(tipo);
                }
            }
        }
    }else{
        JOptionPane.showMessageDialog(this, "Seleccione tipo de pago");
    }

return tipo;
}
```

**El método agregado() verifica si el usuario ha seleccionado una opción de pago válida y si se han ingresado todos los datos necesarios para realizar la transacción. Si se han ingresado los datos necesarios, el método determina si el pago se realizará con tarjeta o en efectivo, y establece el tipo de pago correspondiente llamando al método setPago(). Si el pago se realizará con tarjeta, se verifica que se hayan ingresado los datos de facturación correspondientes llamando al método referencias(). Si todo está correcto, se habilita el botón de validación de tarjeta y se muestra un mensaje de confirmación. Si el pago se realizará en efectivo, se establece el tipo de pago y se retorna. Si el usuario no ha seleccionado una opción de pago válida, se muestra un mensaje de error. El método retorna el tipo de pago ("TARJETA" o "EFECTIVO") según corresponda.**

```
public void descargarFactura(String codigo){

    FileWriter factura;

    try
    {
        factura = new FileWriter("factura.html");
        factura.write("<html><head><title>FACTURA</title></head><body> \n");
        setNo();
        factura.write("<h1> FACTURA No. "+noFactura+"</h1>");

        for(int i = 0;i<Logic.tamaño();i++){
            Cotizacion temporal = Logic.completa(i);

            if(temporal.getCodigo().equals(codigo))
            {
                String alias = Logic.completa(i).getAlias();
                // datos de facturacion
                for(int j = 0; j<Logics.tamaño();j++){
                    if(Logics.completa(j).getCodigo().equals(alias)){
                        String nit = Logics.completa(j).getNit();
                        String nombre = Logics.completa(j).getNombrecom();
                        String direccion = Logics.completa(j).getDireccion();
                        factura.write("<h2> Datos personales</h2>");
                        factura.write("<p>NIT: "+nit+"</p>");
                        factura.write("<p>NOMBRE: "+nombre+"</p>");
                        factura.write("<p>DIRECCION: "+direccion+"</p>");
                    }
                }

                factura.write("<h2>Datos de compra</h2>");
                factura.write("<p> Codigo paquete "+codigo+"</p>");

                String origenDEP = temporal.getOrigenD();
                String origenMUN = temporal.getOrigenM();
                factura.write("<h3>ORIGEN</h3>");
                factura.write("<p>Departamento "+origenDEP+"/p");
                factura.write("<p>Municipio "+origenMUN+"</p");

                String destinoDEP = temporal.getDestinoD();
                String destinoMUN = temporal.getDestinoM();
                factura.write("<h3>DESTINO</h3>");
                factura.write("<p>Departamento "+destinoDEP+"/p");
                factura.write("<p>Municipio "+destinoMUN+"</p");

                String pago = temporal.getTipoPago();
                if(pago.equalsIgnoreCase("efectivo")){
                    pago = "Cobro contra entrega";
                }else{
```




**Este código define un método llamado "descargarFactura" que toma como parámetro un "codigo" que representa el código del paquete para el cual se va a descargar la factura. El método crea un archivo llamado "factura.html" y escribe en él información sobre el paquete y la factura.**

**Primero, se crea un objeto FileWriter para escribir en el archivo "factura.html". Luego, se escribe el encabezado HTML del documento, que incluye el título de la página y la etiqueta "body".**

**A continuación, se establece el número de factura, que se escribe en la página como un título. Después, se realiza un bucle que recorre todos los objetos de Cotizacion que se encuentran en un objeto de tipo Logic. Se busca el objeto de Cotizacion que corresponde al código del paquete que se pasó como parámetro al método.**

**Si se encuentra un objeto de Cotizacion con el código del paquete correspondiente, se busca el objeto de Facturacion que tiene el mismo alias que el objeto de Cotizacion. Los datos de facturación se escriben en la página, incluyendo el NIT, el nombre y la dirección del cliente.**

**Luego se escriben los datos del paquete, incluyendo el origen y destino (departamento y municipio), el tipo de pago, el tamaño del paquete, el número de paquetes y el total a pagar.**

**Finalmente, se escribe el cierre del documento HTML y se cierra el objeto FileWriter. Si se produce una excepción IOException durante el proceso, se registra en un objeto Logger.**