Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Introducción a la Programación de Computadoras 1
Primer Semestre 2023

Manual de programador

Nombre: Jose Javier Bonilla Salazar
Carnet: 202200035

índice

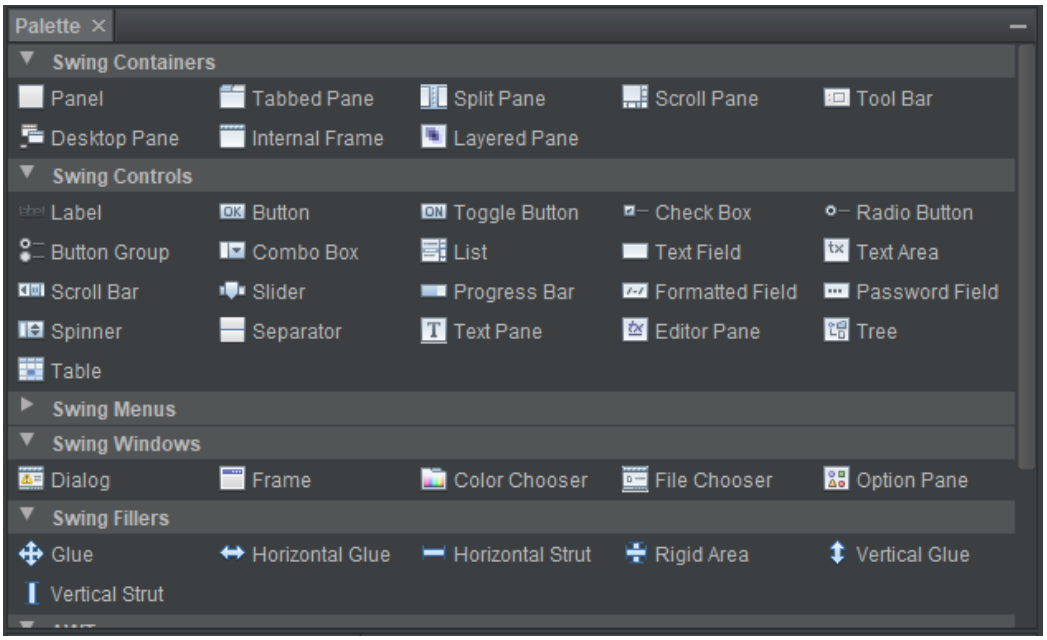
1. Librerías

```
import java.util.ArrayList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;
```

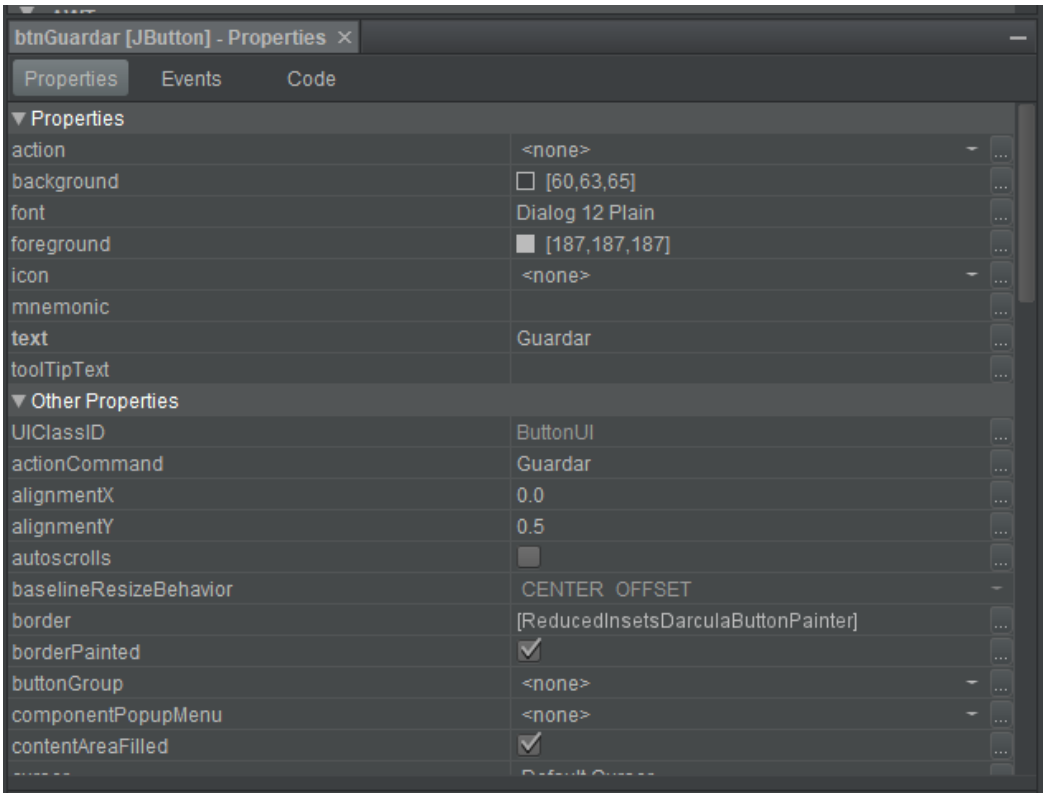
A continuación, se muestran las clases que darán funcionamiento a nuestra aplicación, la primera servirá para el uso de arreglos dinámicos.
La segunda y tercera servirán para el manejo de expresiones regulares.
La cuarta y quinta servirán como opciones adicionales de diseño de la librería “swing”.

1.1. Java swing

Esta librería tiene la función de crear interfaces gráficas, para un uso mas amigable con el usuario.



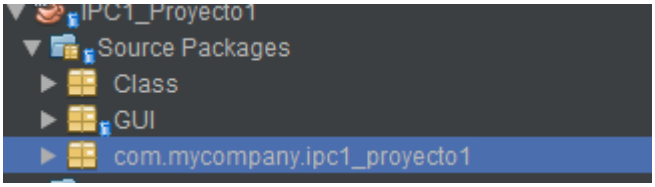
Como se aprecia en la imagen se puede ver cierta cantidad de objetos los cuales se pueden usar para generar una interfaz gráfica.



En esta parte de la imagen se puede ver las propiedades que se pueden modificar del objeto con el cual este trabajando.

2. División de la aplicación

Esta aplicación se dividió en tres carpetas con el propósito de tener un código mas ordenado y fácil de leer.



2.1. Clase arranque

Se encuentra en la carpeta com.mycompany.ipc1_proyecto1, como indica el nombre de la sección esta clase sirve para el arranque de la aplicación además de generar las variables globales que se usaran en el resto del código.

```
public class IPC1_Proyecto1 {

    public static ArrayList<Users> usuarios = new ArrayList<Users>();
    public static ArrayList<KioskoC> kioskos = new ArrayList<KioskoC>();
    public static ArrayList<Regiones> regiones = new ArrayList<Regiones>();
    public static ArrayList<Tarjetas> tarjetas = new ArrayList<Tarjetas>();


    public ArrayList<Users> getArreglo() {
        return usuarios;
    }

   public ArrayList<KioskoC> getArregloK() {
       return kioskos;
   }

   public ArrayList<Regiones> getArregloR(){
       return regiones;
   }

   public ArrayList<Tarjetas> getArregloT(){
       return tarjetas;
   }
```

Dentro de esta clase se puede observar cómo se crean los cuatro arreglos que se utilizaran para guardar la información necesaria de los usuarios, además cuenta con cuatro métodos los cuales retornan el arreglo para poder usarlo en las demás vistas o clases.

```
    //Datos quemados
    Users admin = new Users("ipc1_202200035@ipc1delivery.com","Jose Javier","Bonilla Salazar",

    Regiones reg1 = new Regiones("Metropolitana","M",35,25);
    Regiones reg2 = new Regiones("Norte","NT",68.50,45.55);
    Regiones reg3 = new Regiones("Nororiente","NO",58.68,35.48);
    Regiones reg4 = new Regiones("Suroriente","SO",38.68,32.48);
    Regiones reg5 = new Regiones("Suroccidente","SOC",34,29);
    Regiones reg6 = new Regiones("Noroccidente","NOC",44.50,40);

    usuarios.add(admin);

    regiones.add(reg1);
    regiones.add(reg2);
    regiones.add(reg3);
    regiones.add(reg4);
    regiones.add(reg5);
    regiones.add(reg6);

    Login login = new Login();
    login.setVisible(true);
```

También cuenta con un área de datos quemados, en esta área primero se llama a la clase “Users” y se crea al usuario administrador” además se llama a la clase “Regiones” para crear a las primeras seis regiones. Posterior a eso, se procede a agregar cada objeto a su respectivo arreglo por medio de la función “add()”.

Al final se puede observar que las dos últimas línea de código se utilizan para llamar a la vista “Login” la cual servirá para ingresar sesión.

2.2. Clases

Como su nombre lo indica en esta carpeta solo se guardarán las clases que se utilizaran en la aplicación.

2.2.1. Instanciar clases

```
Users usuario = new Users();
```

Para instanciar una clase primero se debe llamar a la clase y se le debe asignar un nombre.

```
import Class.Users;
```

También se debe importar la clase en la pestaña que se esta utilizando.

2.2.2. Manejo de listas

La clase “Listas” tiene como función principal manejar las funcione de las listas.

```
        public String obtnerNombre(String user){
        for (Users u : lstUsuario) {
            if(u.getCorreo().equals(user) ) {
                String usuario = u.getNombre();
                return usuario;
            }
        }
        return "";
    }
```

En este código se puede observar como se utiliza el “for each” para verificar si existe el correo en el arreglo “lstUsuario” si existe regresar el nombre del usuario y sino existe no regresara nada.

```
public String validarUsuario(String user, String password){
    for (Users u : lstUsuario) {
        if(u.getCorreo().equals(user) && u.getContraseña().equals(password) && u.getRol().equals("Administrador")) {
            return "admin";
        }
        else if(u.getCorreo().equals(user) && u.getContraseña().equals(password) && u.getRol().equals("Individual")) {
            return "individual";
        }
        else if(u.getCorreo().equals(user) && u.getContraseña().equals(password) && u.getRol().equals("Kiosko")) {
            return "kiosko";
        }
    }
    return "incorrecto";
}
```

Se emplea nuevamente el uso de “for each” para verificar que el usuario, contraseña y rol coincidan con los que se ingresaron en los arreglos.

2.2.3. Mantener información en clases

```
protected String correo;
protected String nombre;
protected String apellido;
protected String contraseña;
protected String dpi;
protected String fechaNacimiento;
protected String genero;
protected String nacionalidad;
protected String alias;
protected String telefono;
protected String rol;
protected String img;

public String getCorreo() {
    return correo;
}

public void setCorreo(String correo) {
    this.correo = correo;
}
```

Se puede observar que se crean variables con la palabra reservada “protected” esto con la finalidad de agregar protección a la aplicación, ya que de este modo dichas variables no pueden ser accedidas desde otra clase.

Para poder guardar datos y mostrarlos de dichos objetos se deben de crear dos métodos “get” y “set” con los cuales se podrá acceder a los objetos.

2.2.4. Constructores

```java
public Users() {
    this.correo = "";
    this.nombre = "";
    this.apellido ="";
    this.contraseña = "";
    this.dpi = "";
    this.fechaNacimiento = "";
    this.genero = "";
    this.nacionalidad = "";
    this.alias = "";
    this.telefono = "";
    this.rol = "";
    this.img = "";
}

public Users(String nombre, String apellido, String correo) {
    this.correo = correo;
    this.nombre = nombre;
    this.apellido = apellido;
}

public Users(String correo, String nombre, String apellido, String contraseña, String dpi, String fechaNacimiento,
        String genero, String nacionalidad, String alias, String telefono, String rol, String img) {
    this.correo = correo;
    this.nombre = nombre;
    this.apellido = apellido;
    this.contraseña = contraseña;
    this.dpi = dpi;
    this.fechaNacimiento = fechaNacimiento;
    this.genero = genero;
    this.nacionalidad = nacionalidad;
    this.alias = alias;
    this.telefono = telefono;
    this.rol = rol;
    this.img = img;
}
```

Los constructores sirve para almacenar la información de los objetos, según la necesidad se pueden crear constructores que no reciban ningún parámetro como es el primero de la foto, o el segundo que solo recibe ciertos parámetros o el ultimo que recibe todos los parámetros de la clase.

2.2.5. Mantener información de vista en vista

```java
public class UsuarioActual {

    public static String correo;

    public static String getCorreo() {
        return correo;
    }

    public static void setCorreo(String correo) {
        UsuarioActual.correo = correo;
    }
```

La clase “UsuarioActual” tiene como propósito el mantener información y compartirla de una ventana a otra. Para lograr esto se debe de crear su objeto y métodos de manera pública y estática, al hacer esto la información que se almaceno en una vista se podrá utilizar en otra vista.

2.2.6. Herencia

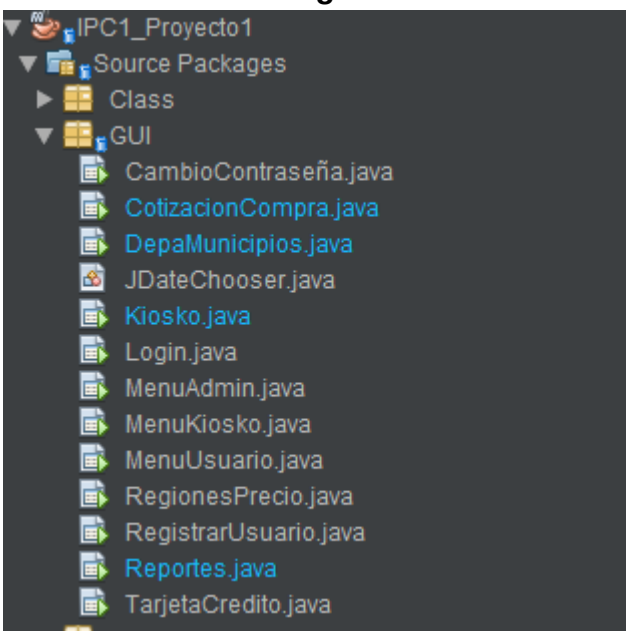
```java
public class Tarjetas extends Users{
    protected String numTarjeta;

    public String getNumTarjeta() {
        return numTarjeta;
    }

    public void setNumTarjeta(String numTarjeta) {
        this.numTarjeta = numTarjeta;
    }
```

Como se puede observar en la primer línea de código se coloca la palabra reservada “extends” la cual sirve para heredar los métodos, objetos y atributos de otra clase, en este caso de “Users” la cual será la clase padre de “Tarjetas”. En este caso solo se deberá de crear el objeto nuevo que se desea usar.

```java
    public Tarjetas() {
        this.correo = "";
        this.nombre = "";
        this.apellido ="";
        this.contraseña = "";
        this.dpi = "";
        this.fechaNacimiento = "";
        this.genero = "";
        this.nacionalidad = "";
        this.alias = "";
        this.telefono = "";
        this.rol = "";
        this.img = "";
        this.numTarjeta = "";
    }

    public Tarjetas(String numTarjeta) {
        this.numTarjeta = numTarjeta;
    }

    public Tarjetas(String numTarjeta, String nombre, String apellido, String correo) {
        super(nombre, apellido, correo);
        this.numTarjeta = numTarjeta;
    }

    public Tarjetas(String numTarjeta, String correo, String nombre, String apellido, String contraseña, String dpi, String f
        super(correo, nombre, apellido, contraseña, dpi, fechaNacimiento, genero, nacionalidad, alias, telefono, rol, img);
        this.numTarjeta = numTarjeta;
    }
```

Se puede observar que en el primer constructor no recibe ningún parámetro, en el segundo solo se recibe el parámetro de la clase actual, en el tercer constructor se reciben solo tres objetos de su clase padre además del de la clase actual, se utiliza la palabra reservada “super” para heredar todos los objetos de la clase padre. Y en el ultimo constructor se heredan todos los objetos.

2.3. Interfaz grafica



En dicha carpeta se guardan todos los “jFrame” que se utilizan para crear las ventanas que utilizara el usuario.

En este caso solo se utiliza el “jlabel” que sirve para mostrar texto quemado o dinámico.

“JTextField” se utilizará para que el usuario ingrese texto.

“Jbutton” para que el usuario realice acciones.

“Jlist” para que el usuario vea las opciones que tiene para seleccionar algo.

“JTable” para mostrar los datos de un arreglo al usuario.

2.3.1. Obtener datos de un JTextField

```
String user = username.getText();
String password1 = password.getText();
```

Se deberá de crear una variable del tipo que se necesite, y se debe igualar al JTextField y utilizar el método “getText()” el cual obtendrá el texto de este.

2.3.2. Asignar datos a un JTextFiel y JLabel

```
txtCorreo.setText("");
txtNombre.setText("");
txtApellido.setText("");
txtContraseña.setText("");
txtContraseña2.setText("");
txtDpi.setText("");
txtFecha.setText("");
txtNacionalidad.setText("");
txtAlias.setText("");
txtTelefono.setText("");
```

Se deberá de llamar al JTextFiel o JLabel y se debe de usar el método “setText()” el cual sirve para asignar texto, dentro de los paréntesis y dentro de comillas dobles se deberá colocar el texto deseado. Si se muestra una variable esta no debe ir encerrada por comillas.

2.3.3. Mostar datos en un JList

```
ArrayList rol = new ArrayList();
ArrayList genero = new ArrayList();
DefaultListModel modelo = new DefaultListModel();
DefaultListModel modeloG = new DefaultListModel();
```

Para esto primero se debe de crear un arreglo y un modelo, el modelo servirá para guardar los datos que se asignaran al JList.

```
rol.add("Admin");
rol.add("Individual");
rol.add("Kiosko");

genero.add("F");
genero.add("M");

modelo.removeAllElements();
for (int i = 0; i < rol.size(); i++) {
    modelo.addElement(rol.get(i));
}

modeloG.removeAllElements();
for (int i = 0; i < genero.size(); i++) {
    modeloG.addElement(genero.get(i));
}

listRol.setModel(modelo);
jlistGenero.setModel(modeloG);
```

En este caso primero se guardan los datos necesarios en el arreglo y luego se utiliza el método “removeAllElement” en nuestro modelo esto con la finalidad de limpiar completamente el modelo.

Seguido se utiliza un ciclo for para recorrer nuestro arreglo y guardar en nuestro modelo los datos que necesitamos.

De ultimo se llama a nuestro “JList” y se usa el método “setModel” para asignar el modelo que necesitemos.

2.3.4. Validaciones de inicio de sesión

```
if("admin".equals(lst.validarUsuario(user, password1))) {
    currentUser.setCorreo(user);
    this.setVisible(false);
    MenuAdmin menu = new MenuAdmin();
    menu.setVisible(true);

}
else if("individual".equals(lst.validarUsuario(user, password1))) {
    currentUser.setCorreo(user);
    this.setVisible(false);
    MenuUsuario menu = new MenuUsuario();
    menu.setVisible(true);

}
else if("kiosko".equals(lst.validarUsuario(user, password1))) {
    currentUser.setCorreo(user);
    this.setVisible(false);
    MenuKiosko menu = new MenuKiosko();
    menu.setVisible(true);

}
else {
    JOptionPane.showMessageDialog(null, "Usuario y/o contraseña incorrecta, por favor revise
}
```

Para validar la sesion, se debe de obtener el correo y contraseña del login, en este caso se usara el método “validarUsuario” y se le pasan dos parámetros, los cuales son el correo y la contraseña.

```
public String validarUsuario(String user, String password){
    for (Users u : lstUsuario) {
        if(u.getCorreo().equals(user) && u.getContraseña().equals(password) && u.getRol().equals("Administrador")) {
            return "admin";
        }
        else if(u.getCorreo().equals(user) && u.getContraseña().equals(password) && u.getRol().equals("Individual")) {
            return "individual";
        }
        else if(u.getCorreo().equals(user) && u.getContraseña().equals(password) && u.getRol().equals("Kiosko")) {
            return "kiosko";
        }
    }
    return "incorrecto";
}
```

Este método se utilizará para recorrer la lista de usuarios, con el uso de la clase “Users”, se tendrá que verificar si el correo, contraseña y el tipo de rol coinciden, si ese es el caso regresara el tipo de rol que necesita y se le llevara a ese menú. En dado caso no coincide se le mostrara un mensaje de alerta.

2.3.5. Cambio de pestañas

```
this.setVisible(false);
MenuKiosko menu = new MenuKiosko();
menu.setVisible(true);
```

En este caso para cambiar la pestaña se utiliza el método “setVisible” según el dato booleano que se le asigne, mostrara o no la pestaña, con “false” se ocultara y con “true” se mostrara.
Al usar “this” estamos referenciando la pestaña en la cual estamos. Para llamar a otra pestaña primero se debe instanciar.

2.3.6. Guardar datos en un arreglo

```
String correo = currrenUser.getCorreo();
String nombre = obtenerNombre.obtnerNombre(correo);
String apellido = obtenerNombre.obtnerApellido(correo);

String tarjeta = txtTarjeta.getText();

tarjetas.setNumTarjeta(tarjeta);
tarjetas.setNombre(nombre);
tarjetas.setApellido(apellido);
tarjetas.setCorreo(correo);

(lst.getArregloT()).add(tarjetas);
```

Para guardar datos en un arreglo, primero se deben de guardar esos datos en la clase deseada y en los objetos que se utilizaran. Posterior a eso se debe de instancia la lista en la cual se guardaran los datos y se utilizar el método “add()” en el cual se le asignara el objeto.

2.3.7. Mostrar mensajes de alerta

```
JOptionPane.showMessageDialog(null, "Tarjeta agregada con exito");
```

Se utilizar la clase JOptionPane, el cual nos dejara mostrar mensajes al usuario, en el apartado de las comillas dobles es donde se agregara el mensaje que se desea mostrar.