

ALDO SAÚL VÁSQUEZ MOREIRA

CARNET: 202109754

FACULTAD DE INGENIERÍA

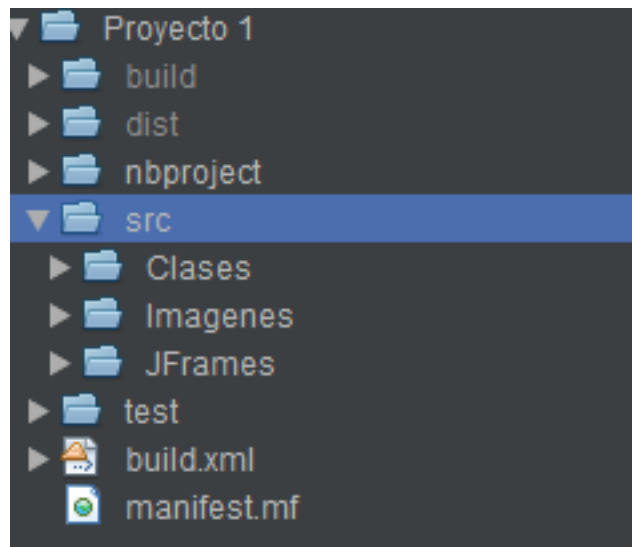
ESCUELA DE CIENCIAS Y SISTEMAS

LABORATORIO DE INTRODUCCIÓN A LA PROGRAMACIÓN 1



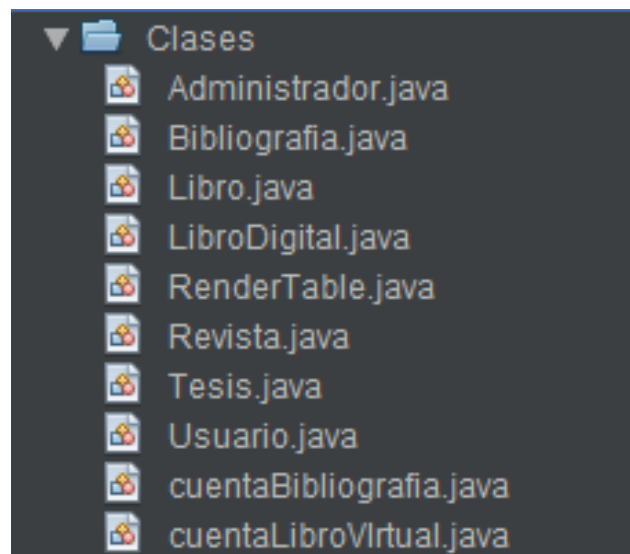
MANUAL TÉCNICO

El proyecto del sistema de administración de la Universidad de San Carlos de Guatemala tiene la siguiente estructura:



Principalmente, nos enfocaremos en el paquete src y los paquetes que esta contiene pues en estas se desarrolló todo el sistema.

- **Paquete Clases:** En este se encuentran todos los modelos de datos necesarios para el desarrollo del sistema. Se cuenta con una clase para cada uno de los tipos de bibliografía, así como para los usuarios.



Se debe resaltar que en las clases se hizo del patrón Singleton. Esto con el fin de evitar que se realiza más de una instancia de estas clases, permitiendo así, que se conservaran los datos manipulados mientras el programa se encuentra en ejecución


```
package Clases;

public class Usuario {

    public int contador = -1;
    public String[] id = new String[10];
    public String[] nombre = new String[10];
    public String[] apellido = new String[10];
    public String[] user = new String[10];
    public String[] rol = new String[10];
    public String[] password = new String[10];
    public boolean usuarioConInstanciado = false;

    private static Usuario instancia;

    public static Usuario getInstancia() {
        if (instancia == null) {
            instancia = new Usuario();
        }
        return instancia;
    }
}
```

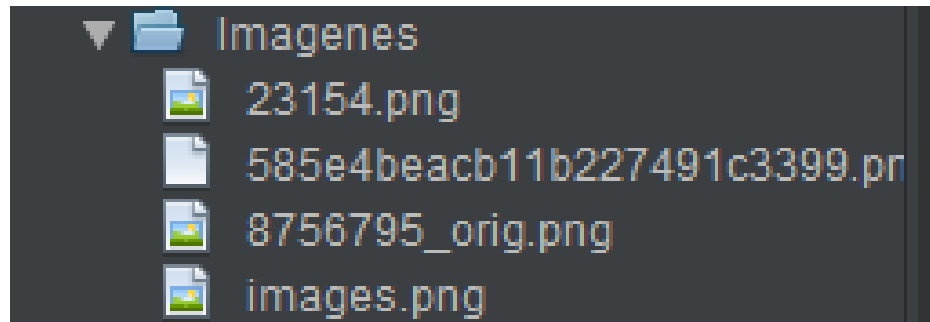


Se puede observar que los atributos de la clase son arreglos; se realizó de esta forma ya que el sistema es bastante elaborado, dirigido para que grandes cantidades de usuarios y bibliografía sea manipulado. Por lo tanto, al no contar con una base de datos se almacena la información en arreglos.

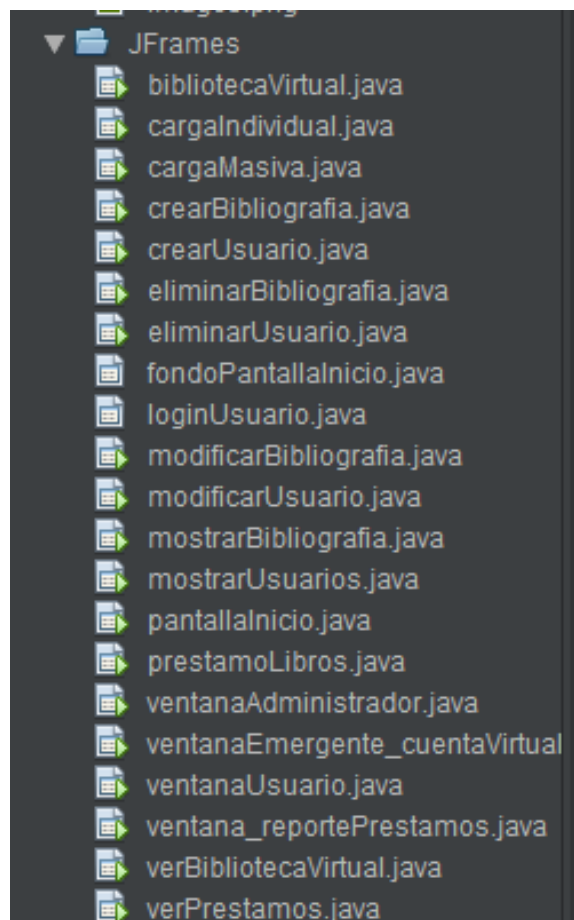
Este mismo patrón se aplicó en el resto de clases para el fin mencionado anteriormente.

Asimismo, la clase RenderTabla se creó para poder ingresar botones en las tablas. Botones como botón prestar, botón devolver, etc.

- **Paquete imágenes:** En este paquete se encuentran las imágenes que se utilizaron para mejorar estéticamente a la interfaz de usuario para hacerla lo más amigable posible.



- **Paquete JFrames:** En este paquete se encuentran todos los archivos destinados a la parte visual del proyecto haciendo uso de JAVA Swing.



- Peculiaridades del proyecto:

- Clase Split: Para realizar la carga masiva fue necesario utilizar la clase Split. Esto con el fin de separar los datos contenidos en la cadena de texto.

```
String[] cadenaTexto = areaTexto.getText().split(";");  
String[] saltoTexto = areaTexto.getText().split("\n");
```

En este se validó el separador establecido “;” así como los saltos de línea presentes entre cada una de las bibliografías a ingresar.

- Clase JOptionPane: Al ser un proyecto gráfico se deben presentar diversas alertas al usuario para hacer del sistema lo más amigable posible.

```
JOptionPane.showMessageDialog
```

- Patrón Singleton: El patrón singleton, o singleton pattern en inglés, pertenece a la categoría de patrones creativos dentro del grupo de los patrones de diseño. También se le conoce simplemente como “singleton”. El propósito de este patrón es evitar que sea creado más de un objeto por clase. Esto se logra creando el objeto deseado en una clase y recuperándolo como una instancia estática. El singleton es uno de los patrones más simples, pero más poderosos en el desarrollo de software.
- Clase File: Se hizo uso de la clase File para crear los reportes en formato HTML.

```
File reporteExistenciaLibros = new File("reporteExistenciaLibros_administrador.html");  
try {  
    BufferedWriter bw = new BufferedWriter(new FileWriter(reporteExistenciaLibros));  
    bw.write(reporte);  
    bw.close();  
    JOptionPane.showMessageDialog(null, "Reporte de existencia de libros realizado correctamente",  
    } catch (Exception e) {  
    }  
}
```

- Código repetido: Acciones que se realizaron con mayor frecuencia:

- Recorrer arreglos y validar datos:

```

String idUsuario = txt_id.getText();

for (int i = 0; i < Usuario.getInstance().contador; i++) {
    if (Usuario.getInstance().getId().equals(idUsuario)) {
        idExistente = i;
    }
}

if (idExistente == 0) {

    if (txt_confirmarPassword.getText().equals(txt_password.getText())) {
        JOptionPane.showMessageDialog(null, "Nuevo usuario: " + txt_usuario.getText(), "Usuario creado con éxito", JOptionPane.INFORMATION_MESSAGE);

        Usuario.getInstance().contador = Usuario.getInstance().contador + 1;
        Usuario.getInstance().setId(Usuario.getInstance().contador);
        Usuario.getInstance().nombre(Usuario.getInstance().contador);
        Usuario.getInstance().apellido(Usuario.getInstance().contador);
        Usuario.getInstance().correo(Usuario.getInstance().contador);
        Usuario.getInstance().confirmarPassword.getText() = txt_usuario.getText();
        Usuario.getInstance().confirmarPassword.getText() = (String)txt_confirmarPassword.getText();
        Usuario.getInstance().password(Usuario.getInstance().contador) = txt_password.getText();

        txt_id.setText("");
        txt_usuario.setText("");
        txt_apellido.setText("");
        txt_correo.setText("");
        txt_confirmarPassword.setText("");
        txt_confirmarPassword.setText("");
        txt_confirmarPassword.setText("");

        mostrarUsuario.getInstance().setVisible(false);
        mostrarAdministrador.getInstance().setVisible(true);

        System.out.println(Usuario.getInstance().contador);

        for (int i = 0; i < Usuario.getInstance().contador; i++) {
            System.out.println(Usuario.getInstance().getId(i));
            System.out.println(Usuario.getInstance().nombre(i));
            System.out.println(Usuario.getInstance().apellido(i));
            System.out.println(Usuario.getInstance().correo(i));
        }
    }
}

```

Tal como se muestra en la imagen, un código muy repetitivo fue el que se utilizó para recorrer los arreglos creados en las clases mencionadas anteriormente. Esto con el fin de validar ciertos parámetros que permitieran, ya sea, crear, eliminar, modificar e incluso mostrar datos, por ejemplo.

- Trasferir datos de un arreglo a una JTable: Haciendo uso de estructuras iterativas y de selección se asignaba un valor a un modelo de datos el cual luego sería transferido por medio del método setModel();

```

private void btn_verBibliografiaActionPerformed(java.awt.event.ActionEvent evt) {
    String[][] tablaLibros = new String[10][10];
    for (int i = 0; i < Libro.getInstance().contador; i++) {
        for (int j = 0; j < 10; j++) {
            tablaLibros[i][0] = Libro.getInstance().libro[i];
            tablaLibros[i][1] = Libro.getInstance().autor[i];
            tablaLibros[i][2] = Libro.getInstance().anoPublicacion[i];
            tablaLibros[i][3] = Libro.getInstance().titulo[i];
            tablaLibros[i][4] = Libro.getInstance().edicion[i];
            tablaLibros[i][5] = Libro.getInstance().palabrasClave[i];
            tablaLibros[i][6] = Libro.getInstance().descripcion[i];
            tablaLibros[i][7] = Libro.getInstance().temas[i];
            tablaLibros[i][8] = Libro.getInstance().copies[i].toString();
            tablaLibros[i][9] = Libro.getInstance().disponibilidad.toString();
        }
    }

    String[] encabezadoLibros = {"ID", "Autor", "Año Publicación", "Titulo", "Edición", "Palabras Clave", "Descripción", "Temas", "Copias", "Disponibilidad"};

    tbl_verBibliografia.setModel(new DefaultTableModel(
        tablaLibros,
        encabezadoLibros
    ));
}

```

- Creación de método filter: Se creó un método haciendo uso de la clase TableRowSorter, esto para cumplir con el requerimiento de filtración en las tablas.

```
private void filter(String query) {  
    TableRowSorter<TableModel> tr = new TableRowSorter<>(tbl_mostrarLibrosDigitales.getModel());  
    tbl_mostrarLibrosDigitales.setRowSorter(tr);  
    tr.setRowFilter(RowFilter.regexFilter(query));  
}
```

Eso fue un poco de las partes más importantes y bases para el desarrollo de este sistema.