Universidad San Carlos de Guatemala

Facultad de Ingeniería

Ingeniería en Ciencias y Sistemas

PROYECTO #1

Manual Técnico

Sistema Biblioteca

Introducción a la Programación y Computación 1 Sección A

David Antonio Meza Silva

202500708

1.Datos Generales

- **Nombre del Programa**: IPC1_Proyecto1_202500708
- **Lenguaje:** Java 25
- **Entorno en donde se desarrolló:** Apache NetBeans IDE 28

2. Descripción del programa:

El sistema de Biblioteca esta construido en base al paradigma de Programación a Objetos (POO) en el lenguaje de programación Java. Se utilizó una arquitectura de capas para la separación de lógica y responsabilidades en diferentes clases:

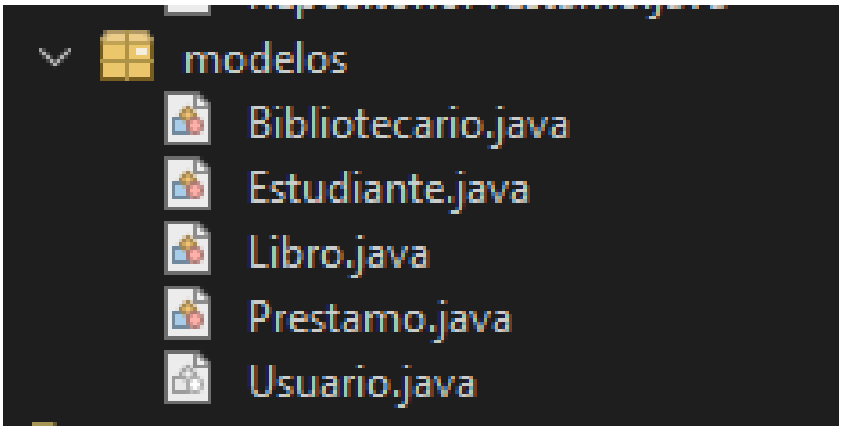
- Modelos: Clases que se encargan de definir los atributos de los usuarios en el sistema.
- Repositorios: Clases que se encargan de la gestión y almacenamiento de datos de los objetos.
- Controladores: Clases encargadas de manejar el flujo del programa.
- Ventanas/Interfaces Gráficas: Encargadas de realizar la parte gráfica del sistema.

3. Arquitectura/Paquetes

El programa se organiza en diferentes paquetes

3.1. Modelos:

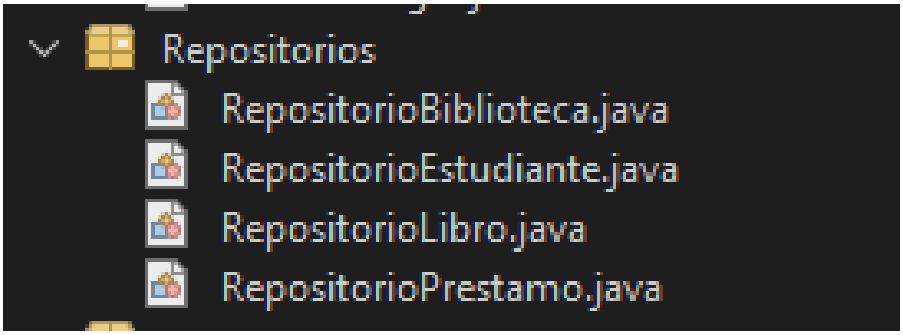
- **Usuario (Abstracto):** Se encarga de realizar la abstracción de los atributos de un usuario en el sistema. Posee atributos comunes como CUI, nombre, correo y credenciales.
- **Estudiante y Bibliotecario:** Heredan atributos de Usuario, y agregan otros atributos según el rol (Carnet, Carrera, ID de empleado, turno, salario).
- **Libro:** Se encarga de definir atributos como ISBN, Titulo y Autores para los Libros.
- **Préstamo:** Clase que se encarga de la gestión de fechas y calculo de multas.



3.2. Repositorios:

Capa que sirve para el acceso y manipulación de datos.

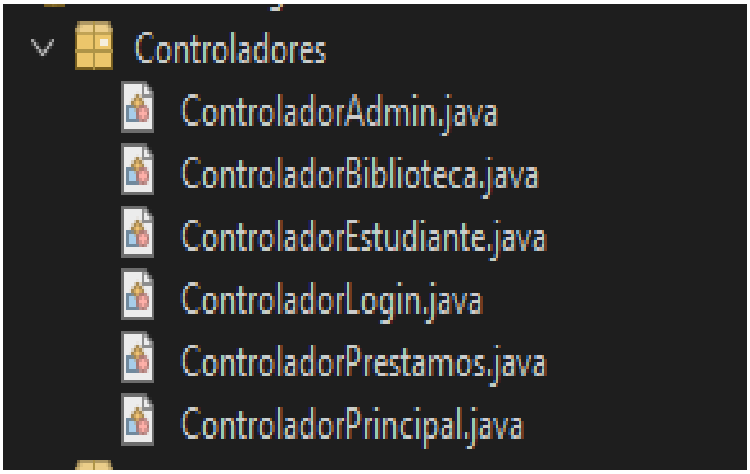
- **RepositorioLibro,Estudiante,Biblioteca, Préstamos:** Se encargan de gestionar los arreglos (Libro[],Estudiante[], etc.), e incluyen métodos para agregar, buscar, eliminar datos, etc.. (utilizan “null” para establecer su posición inicial).
- **Arreglos:** Para darle listas a la interfaz, se realizan “copias limpias” de los arreglos que eliminan los espacios para evitar errores en las tablas visuales.



3.3.Controladores:

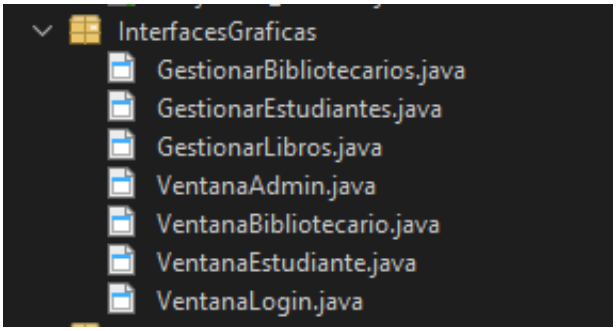
Intermediarios entre la interfaz gráfica y los repositorios.

- **ControladorPrincipal:** Se encarga de el manejo de vistas e inicio de controladores.
- **ControladorAdmin:** Controlador más complejo, se encarga de gestionar la lógica de ordenamiento, carga masiva y guardar datos.
- **ControladorLogin:** Se encarga de la lógica para iniciar sesión según el usuario que va a ingresar.
- **ControladorPrestamos:** Se encarga de la lógica de préstamos para el estudiante y los libros.
- **Controlador Estudiante y Biblioteca:** Se encarga de obtener el Estudiante/Bibliotecario actual y sus atributos.



3.4. InterfacesGraficas

Contiene los JFrame, Jdialog, Jtables, botones y etc. para la visualización del programa según el rol de usuario (Admin, Estudiante, Bibliotecario), además de pequeños métodos complementarios para la obtención y display de datos.



4. Algoritmos Implementados:

4.1. Recursividad:

Se implementó un método de recursividad para la Búsqueda de libros dentro del Repositorio de Libros, donde el método: buscarLibroRecursivo(String ISBN, int índice) recorre el arreglo, y según el caso:

1. Índice >= al tamaño del arreglo de libros, significa que no se encontró libro y retorna null.
2. Libros[índice].getISBN, coincide, retorna el objeto.

Luego de esto se llama a si mismo, utilizando un índice + 1.Con este método, se permite el conteo de cuantos libros coinciden con un filtro de búsqueda antes de crear el arreglo de resultados donde se devolverán estos mismo.

En este caso, la búsqueda recursiva por ISBN tiene una complejidad aproximada O(n), donde n es la cantidad de libros en el arreglo.

```java
private Libro buscarLibroRecursivo(String ISBN, int indice) {
    // Se regresa valor vacio si se llega el final y no encuentra coincidencia.
    if (indice >= libros.length) {
        return null;
    }

    Libro actual = libros[indice];

    // Si hay libro en esta posición y coincide con el ISBN, se devuelve
    if (actual != null && ISBN.equals(actual.getISBN())) {
        return actual;
    }

    //buscar en la siguiente posición
    return buscarLibroRecursivo(ISBN, indice + 1);
}
```

4.2. Ordenamiento:

Se implementaron 5 algoritmos diferentes en la clase ControladorAdmin para ordenar los reportes de libros según diferentes criterios:

1. Burbuja (Bubble Sort): Utilizado para ordenar por ISBN. Compara pares adyacentes y los intercambia si están en desorden.
2. Selección (Selection Sort): Utilizado para ordenar por Título. Busca el elemento menor (alfabéticamente) y lo coloca al inicio.
3. Inserción (Insertion Sort): Utilizado para ordenar por Autor. Construye el arreglo ordenado desplazando elementos mayores a la derecha.
4. QuickSort: Utilizado para ordenar por Editorial. Utiliza un pivote para particionar el arreglo en menores y mayores, ordenando recursivamente las sub-listas.
5. MergeSort: Utilizado para ordenar por Año de Publicación. Divide el arreglo en mitades recursivamente y luego las mezcla ("merge") en orden.

```
private void burbujaPorISBN(Libro[] lista) {
    for (int i = 0; i < lista.length - 1; i++) {
        boolean huboCambio = false;

        for (int j = 0; j < lista.length - 1 - i; j++) {
            String isbn1 = lista[j].getISBN();
            String isbn2 = lista[j + 1].getISBN();

            if (isbn1.compareToIgnoreCase(isbn2) > 0) {
                intercambiar(lista, j, j + 1);
                huboCambio = true;
            }
        }

        if (!huboCambio) {
            break;
        }
    }
}
```

5.Persistencia/Manejo de Archivos

5.1. Carga Masiva (CSV)

El sistema utiliza BufferedReader y FileReader para la lectura de los archivos de CSV. Estos se contienen los datos separados por comas, los cuales son leidos línea por línea (utilizando Split) para separar los campos.

Se utiliza un try-catch para evitar que el programa falle/colapse en caso de que el archivo CSV no contenga los datos correctos (números en lugar de letras, y viceversa).

```
public String cargarLibrosCSV(File archivo) {
    int exitosos = 0;
    int errores = 0;
        StringBuilder reporte = new StringBuilder();
        reporte.append("=== REPORTE DE CARGA DE LIBROS ===\n");

    try (BufferedReader br = new BufferedReader(new FileReader(archivo))) {
        String linea;
            boolean primeraLinea = true;
            int numeroLinea = 0;

        while ((linea = br.readLine()) != null) {
            numeroLinea++;
            if (primeraLinea) {
                primeraLinea = false;
                continue;
            }
            if (linea.trim().isEmpty()) {
                continue;
            }

            try {
                String[] datos = linea.split(",");
                if (datos.length < 9) {
                    reporte.append("Linea ").append(numeroLinea).append(": Datos incompletos\n");
                        errores++;
                        continue;
                }
```

5.2. Reporte HTML

Se utiliza PrintWriter y un método que genera el reporte en un html, donde se concatenan los datos de los objetos estudiante y Prestamos para su visualización web.

```
//==================================REPORTE HTML ====================================
  public String generarReporteHTML(String carnet, File archivo) {
      if (carnet == null || carnet.trim().isEmpty()) {
          return "Debe ingresar un carné.";
      }
      carnet = carnet.trim();

      Estudiante est = buscarEstudiantePorCarne(carnet);
      if (est == null) {
          return "No se encontró estudiante con ese carné.";
      }

      Prestamo[] activos = prestamos.prestamosActivosPorEstudiante(carnet);
      Prestamo[] historial = prestamos.historialPorEstudiante(carnet);

      try (PrintWriter pw = new PrintWriter(new FileWriter(archivo))) {
          pw.println("<!DOCTYPE html>");
          pw.println("<html><head><meta charset=\"UTF-8\"><title>Reporte de Estudiante</title>");
          pw.println("<style>");
          pw.println("table { border-collapse: collapse; width: 100%; }");
          pw.println("th, td { border: 1px solid #000; padding: 4px; text-align: left; }");
          pw.println("th { background-color: #ddd; }");
          pw.println("</style>");
          pw.println("</head><body>");

          pw.println("<h1>Información del Estudiante</h1>");
          pw.println("<p><b>Carné:</b> " + est.getCarnet() + "</p>");
          pw.println("<p><b>Nombre:</b> " + est.getNombre() + "</p>");
          pw.println("<p><b>Carrera:</b> " + est.getCarrera() + "</p>");
          pw.println("<p><b>Correo:</b> " + est.getCorreo() + "</p>");
          pw.println("<p><b>Estado:</b> " + est.getEstadoCivil() + "</p>");
```

6. Lógica Préstamos

Se Ubica dentro de en la clase ControladorPrestamos, y posee lo siguiente:

1. Validación de Stock: Antes de prestar, verifica libro.getCantidad() > 0.

2. Límite de Préstamos: Verifica que el estudiante tenga menos de 3 préstamos activos (estado == "ACTIVO").

3. Cálculo de Fechas: Se utiliza la API java.time, donde se define qué: Fecha Devolución Esperada = LocalDate.now().plusDays(7).

4. Cálculo de Multas: Al devolver, se comparan las fechas:
    - diasRetraso = ChronoUnit.DAYS.between(esperada, real)
    - Si hay retraso: Multa = diasRetraso * 2.00 (Q2.00 por día).

```java
package Repositorios;

    import modelos.Prestamo;

public class RepositorioPrestamo {
   private Prestamo[] prestamos;
    private int contador;

    public RepositorioPrestamo(int capacidadInicial) {
        this.prestamos = new Prestamo[capacidadInicial];
        this.contador = 0;
    }

//================================Utilidades==================================
    public void agregarPrestamo(Prestamo p) {
        if (p == null) {
            return;
        }
        if (contador >= prestamos.length) {
            return;
        }
        prestamos[contador] = p;
        contador++;
    }
```

```java
public Prestamo[] prestamosActivosPorEstudiante(String carnet) {
    if (carnet == null) return null;
        carnet = carnet.trim();
    if (carnet.isEmpty()) return null;

    int c = 0;
        for (int i = 0; i < contador; i++) {
            Prestamo p = prestamos[i];
            if (p != null
                && carnet.equals(p.getCarnetEstudiante())
                && "ACTIVO".equals(p.getEstado())) {
                c++;
            }
        }

    Prestamo[] resultado = new Prestamo[c];
        int idx = 0;
        for (int i = 0; i < contador; i++) {
            Prestamo p = prestamos[i];
            if (p != null
                && carnet.equals(p.getCarnetEstudiante())
                && "ACTIVO".equals(p.getEstado())) {
                resultado[idx++] = p;
            }
        }
    return resultado;
}

public Prestamo[] historialPorEstudiante(String carnet) {
    if (carnet == null) return null;
        carnet = carnet.trim();
    if (carnet.isEmpty()) return null;

    int c = 0;
        for (int i = 0; i < contador; i++) {
            Prestamo p = prestamos[i];
            if (p != null && carnet.equals(p.getCarnetEstudiante())) {
                c++;
            }
```