UNIVERSIDAD MAYOR DE SAN ÁNDRES

CARRERA DE INFORMÁTICA

PROGRAMACIÓN II INF-121

PROYECTO FINAL

“BIBLIOTECA ACADÉMICA”

Integrantes:

- **Gomez Blanco Cash Israel** **C.I : 11093913**
- **Agner Asistiri Yoel Americo** **C.I : 13553236**
- **Rodriguez Mendez Gianina Madelein** **C.I : 9110875**
- **Echenique Mayta Lizeth Hortencia** **C.I : 12864297**
- **Mamani Quispe Gustavo Giovanni** **C.I : 9082100**
- **Peredo Mamani Devon Richard** **C.I :15744531**

Docente: Lic. Rosalia Lopez

LA PAZ - BOLIVIA 2025

BIBLIOTECA ACADEMICA

BIBLIOTECA ACADEMICA

1. INTRODUCCION

Este proyecto resuelve el querer encontrar libros para un tema en específico o para un tema de interés. Tenemos el deber de hacer un sistema que nos permita prestar y devolver libros de interés ya que ese el problema que queremos resolver "La devolución de libros" esto se llevara mediante registros del usuario que quiere pedir el préstamo respectivo del libro. Las herramientas para la elaboración de nuestro proyecto será **IntelliJ IDEA**, la cual ofrece un entorno óptimo para programar en **Java** aplicando los principios de la **Programación Orientada a Objetos (POO).**

2. OBJETIVOS

Desarrollar un programa que ayude al préstamo y devolución de libros de la carrera de Informática, para un mejor orden y no estar desactualizados a la hora de brindar los ejemplares que brindan una amplia información a los estudiantes, para que no estén con la duda de algún tema de interés propio.

3. OBJETIVOS ESPECIFICOS

- Diseñar e implementar un sistema que permita gestionar el registro de libros, usuarios, préstamos y devoluciones, aplicando los principios de la Programación Orientada a Objetos (POO).

- Organizar y mantener actualizada la información de los libros disponibles para la carrera de Informática, facilitando su búsqueda y control.

- Automatizar el proceso de préstamo y devolución de libros, asegurando un registro claro y preciso para mejorar la administración de la biblioteca.

4. ANALISIS DEL PROYECTO

El proyecto consiste en el desarrollo de un perfecto código y también un uso sencillo del mismo, para poder hacer un programa que te facilite el préstamo de un libro, también tener una mayor organización con los usuarios/estudiantes que quieran ver si el libro que desean leer está disponible o no y poder devolverlos mediante un mejor registro.

El sistema será desarrollado **Programación Orientada a Objetos (POO)** en Java, y se implementará a través del entorno de desarrollo **IntelliJ IDEA**. De esta manera todo se automatizará y se simplificará rutinas diarias de una biblioteca ofreciendo información más actualizada sobre la disponibilidad de los libros.

4.1 REQUISITOS FUNCIONALES

- El sistema debe permitir registrar libros con información como: código, título, autor, categoría, año de publicación y estado.

- El sistema debe permitir registrar usuarios con sus datos básicos: nombre, identificación, carrera, correo.

- El sistema debe permitir realizar préstamos de libros actualizando automáticamente el estado del libro a "prestado".

- El sistema debe permitir registrar devoluciones y actualizar el estado del libro a "disponible".

- El sistema debe permitir buscar libros disponibles por filtros como título, autor, categoría o palabras clave.

4.2 REQUISITOS NO FUNCIONALES

- El sistema debe contar con una interfaz gráfica simple, intuitiva y fácil de usar para los bibliotecarios.

- El sistema debe estar desarrollado bajo principios de buena práctica de programación (POO) para garantizar su mantenibilidad y escalabilidad.

- El sistema debe ofrecer respuestas rápidas en la gestión de consultas y operaciones de registro.

4.3 CASOS DE USO

Libro de registro:

El bibliotecario podrá ingresar nuevos libros al sistema, proporcionando información como código, nombre, autor, categoría, año de publicación y Bogsatus. Esta funcionalidad le permite actualizar el inventario de la biblioteca.

Registre al usuario:

El sistema le permite registrar nuevos usuarios, guardar sus datos personales, como nombre completo, identificación, carrera y E -RAW. Estos usuarios podrán proporcionar préstamos de libros y regresar.

Haga un préstamo:

El bibliotecario puede registrar un préstamo en nombre de un usuario pregrabado. Al hacer un préstamo, la condición del libro cambia a los "prestatarios" y la fecha relevante está registrada.

Recuesa de registro:

El sistema le permite determinar la devolución de libros ocupados y actualizar su estado a "disponible" para que otros usuarios puedan volver a solicitarlos.

Libro de búsqueda:

El sistema permite que los libros se registren a través de filtros como nombre, autor, categoría o palabras clave. Esta funcionalidad facilitará a los usuarios y bibliotecarios que encuentren los libros disponibles de manera rápida y precisa.

5. DISEÑO DEL SISTEMA

5.1 CLASES Y JERARQUÍAS

A continuación se presenta el diseño del sistema en términos de clases y su jerarquía.

Clase	Atributos principales	Métodos clave
Biblioteca	usuarios, libros, prestamos	agregarUsuario(), agregarLibro(), realizarPrestamo(), devolverLibro()
Usuario	nombre, idUsuario, email	mostrarPrestamos(), getNombre(), getIdUsuario()
Libro	titulo, autor, isbn, estado	isDisponible(), setDisponible(), getTitulo()
Prestamo	usuario, libro, fechaPrestamo, fechaDevolucion	getUsuario(), getLibro(), getFechaPrestamo()

5.2 RELACIONES ENTRE CLASES

Herencia:

El sistema no tiene muchas subclases explícitas en el modelo principal, pero sí implementa

interfaces como Repositorio<T> con clases que las implementan (por ejemplo, RepositorioLibros y RepositorioUsuarios). Además, el patrón Decorator se aplica en las clases relacionadas con ILibro y sus decoradores (LibroBasico, LibroDecorator, LibroConResumen, LibroEdicionEspecial).

Composición / Agregación:

La clase Biblioteca contiene listas de Usuario, Libro y Prestamo (composición).

Prestamo tiene referencias a Usuario y Libro (composición).

La interfaz Repositorio<T> y sus implementaciones usan listas internas para almacenar elementos.

Interacción entre clases:

Biblioteca actúa como controlador central que administra las colecciones y operaciones de préstamo.

La interfaz gráfica BibliotecaUI interactúa con el controlador para registrar usuarios, libros y gestionar préstamos/devoluciones.

El patrón **Observer** se utiliza para notificar a los usuarios sobre eventos relevantes.

El patrón Decorator permite extender la funcionalidad de objetos ILibro sin modificar su estructura base.

6. DESARROLLO

El sistema está compuesto por varias clases como Biblioteca, Usuario, Libro, Prestamo, repositorios genéricos y una interfaz gráfica para interacción.

Se aplicaron conceptos como:

Clases y subclases:

Clases que representan entidades reales y controladores de negocio.

Uso de genéricos:

En la interfaz Repositorio<T>, que permite manejar listas de objetos genéricos (usuarios, libros, etc.) de forma reutilizable.

Métodos sobrescritos:

Sobrescritura del método toString() en clases como Usuario, Libro y Prestamo para mostrar datos de forma clara.

Validaciones:

Se valida la disponibilidad del libro antes de realizar un préstamo y la existencia de usuario y libro antes de registrar acciones.

Persistencia con archivos (.ser):

Serialización y deserialización de listas para guardar y cargar datos usando la clase Persistencia.

Excepciones personalizadas:

No se definieron explícitamente excepciones personalizadas en el código compartido, pero se manejan casos de error con mensajes y control de flujo.

7. APLICACIÓN DE PATRONES DE DISEÑO

7.1 PATRÓN(ES) APLICADOS

Patrón	Rol en el sistema	Clases involucradas
Singleton	Garantiza una única instancia para la gestión centralizada de la Biblioteca	Biblioteca
Factory Method	(Opcional, si se implementa para creación de objetos)	No implementado explícitamente en código
Strategy	(No implementado en el código compartido)	-
Observer	Para notificar cambios y eventos de forma desacoplada	Notificador, Observador, UsuarioObservador
Decorator	Para agregar funcionalidades adicionales a objetos ILibro sin modificar la clase base	ILibro, LibroBasico, LibroDecorator, LibroConResumen, LibroEdicionEspecial

7.2 JUSTIFICACIÓN

Singleton:

Se eligió para centralizar y controlar el acceso a la instancia única de la clase Biblioteca, que administra usuarios, libros y préstamos, evitando inconsistencias.

Observer:

Permite notificar a los usuarios registrados (observadores) sobre eventos importantes en la biblioteca, como la devolución de libros o alertas, de forma desacoplada.

Decorator:

Facilita la extensión dinámica de funcionalidades en objetos libro (como agregar resumen

o marcar como edición especial) sin modificar la clase base, respetando el principio abierto/cerrado.

7.3 DIAGRAMA Y EJEMPLO DE USO

Código ejemplo para Observer:	Código ejemplo para Decorator:

```
interface Observador {
   actualizar(String mensaje);
}
class UsuarioObservador implements Observador {
   private String nombre;
UsuarioObservador(String nombre) {
      this.nombre = nombre;
   }
actualizar(String mensaje) {
      print("Notificación para " + nombre + ": " + mensaje);
   }
}

class Notificador {
   private List<Observador> observadores = new ArrayList<>();

agregarObservador(Observador o) {
      observadores.add(o);
   }
eliminarObservador(Observador o) {
      observadores.remove(o);
   }
notificar(String mensaje) {
      for (Observador o : observadores) {
         o.actualizar(mensaje);
      }
```

```
interface ILibro {
   String getDescripcion();
}
class LibroBasico implements ILibro {
   private String titulo;
LibroBasico(String titulo) {
      this.titulo = titulo;
   }
String getDescripcion() {
      return "Libro: " + titulo;
   }
}
class LibroDecorator implements ILibro {
   protected ILibro libro;
LibroDecorator(ILibro libro) {
      this.libro = libro;
   }
}
class LibroConResumen extends LibroDecorator {
LibroConResumen(ILibro libro) {
      super(libro);
   }
String getDescripcion() {
      return libro.getDescripcion() + " + incluye resumen";
   }
}
```

```
    }
}
```

```
class LibroEdicionEspecial extends LibroDecorator {
  LibroEdicionEspecial(ILibro libro) {
    super(libro);
  }
String getDescripcion() {
    return libro.getDescripcion() + " (Edición Especial)";
  }
}
```

Observador.java 1

C: > Users > Liz > OneDrive > Escritorio > GRUPOPROY > Pr

```java
package observador;

public interface Observador {
    void actualizar(String mensaje);
}

```

ILibro.java 1

C: > Users > Liz > OneDrive > Escritorio > GRUPO

```java
package decorador;

public interface ILibro {
    String getDescripcion();
}

```

8. PERSISTENCIA DE DATOS

Descripción del formato usado

El proyecto implementa persistencia de datos utilizando archivos de texto plano en formato CSV (Comma-Separated Values) . Este formato fue seleccionado por su simplicidad, portabilidad y facilidad de procesamiento, permitiendo una gestión eficiente de la información sin requerir bases de datos externas. Los archivos CSV permiten almacenar datos tabulares de manera estructurada, facilitando tanto la lectura manual como el procesamiento automatizado.

Clase encargada de lectura/escritura

La clase principal Biblioteca contiene dos métodos fundamentales para la gestión de la persistencia:

Método cargarDatos()

Este método se ejecuta automáticamente al iniciar la aplicación y se encarga de:

- Verificar la existencia de los archivos de datos
- Leer secuencialmente cada archivo CSV

- Parsear la información y reconstruir los objetos en memoria
- Manejar posibles errores de lectura o formato
- Método guardarDatos()

Se ejecuta al finalizar la aplicación y realiza:

- Serialización de todos los objetos en memoria
- Escritura ordenada en archivos CSV correspondientes
- Mantenimiento de la integridad de los datos entre sesiones

Estructura de archivos y ejemplos

Archivo recursos.csv

Almacena todos los recursos disponibles en la biblioteca, diferenciando entre libros y recursos digitales mediante un identificador de tipo al final de cada línea.

- LIB001,Don Quijote de la Mancha,Miguel de Cervantes,1605,950,Libro
- LIB002,Cien años de soledad,Gabriel García Márquez,1967,450,Libro
- DIG001,Introducción a la Programación,Juan Pérez,2020,PDF,3.2,RecursoDigital
- DIG002,Manual de Bases de Datos,Maria López,2019,EPUB,5.7,RecursoDigital

Archivo usuarios.csv

Contiene la información de todos los usuarios registrados en el sistema, identificando su tipo (Estudiante o Docente).

Ejemplo de contenido:

- USR001,Juan Pérez García,Estudiante
- USR002,María Gómez Fernández,Docente
- USR003,Carlos Ruiz Martínez,Estudiante
- USR004,Ana López Sánchez,Docente

Archivo prestamos.csv

Registra todos los préstamos realizados, incluyendo su estado actual (activo/inactivo) para mantener el historial completo.

- LIB001,USR001,01/04/2025,15/04/2025,activo
- DIG001,USR002,05/04/2025,20/04/2025,inactivo
- LIB002,USR003,10/04/2025,25/04/2025,activo

Ventajas del sistema de persistencia implementado

- Simplicidad: No requiere instalación de sistemas de bases de datos adicionales
- Portabilidad: Los archivos CSV pueden ser leídos por múltiples aplicaciones

- Transparencia: El usuario final no necesita conocimientos técnicos para entender los datos
- Mantenimiento: Fácil de auditar y modificar manualmente si es necesario
- Eficiencia: Carga y guardado rápido para volúmenes de datos moderados

Consideraciones técnicas

- Los archivos se crean automáticamente si no existen
- Se implementa manejo de excepciones para prevenir errores de lectura/escritura
- La codificación de caracteres se maneja en UTF-8 para soportar tildes y caracteres especiales
- Se incluyen validaciones para asegurar la integridad de los datos cargados

9. PRUEBAS Y VALIDACIÓN

Casos de prueba realizados

Se ejecutaron las siguientes pruebas manuales para validar el correcto funcionamiento del sistema:

CASO	ACCION REALIZADA	RESULTADO ESPERADO	RESULTADO OBTENIDO

1	Registrar un nuevo libro	Se agrega a la lista de recursos	Correcto
2	Registrar un usuario	Se agrega a la lista de usuarios	Correcto
3	Prestar un recurso existente	Se crea un préstamo activo	Correcto
4	Intentar registrar un recurso con ID duplicado	Mensaje de error	Correcto
5	Devolver un recurso prestado	El préstamo se marca como inactivo	Correcto
6	Guardar y cargar datos	Los datos persisten entre sesiones	Correcto

Capturas de pantalla

ILibro.java 1 ●

C: > Users > Liz > OneDrive > Escritorio > GRUPOPROY > ProyBibliotecaAcademica > ProyBibl

```java
package decorador;

public interface ILibro {
    String getDescripcion();
}
public class LibroBasico implements ILibro {
    private String titulo;

    public LibroBasico(String titulo) {
        this.titulo = titulo;
    }

    @Override
    public String getDescripcion() {
        return "Libro: " + titulo;
    }
}
public class LibroConResumen extends LibroDecorator {
    public LibroConResumen(ILibro libro) {
        super(libro);
    }

    @Override
    public String getDescripcion() {
        return libro.getDescripcion() + " + incluye resumen";
    }
}
public abstract class LibroDecorator implements ILibro {
    protected ILibro libro;

    public LibroDecorator(ILibro libro) {
        this.libro = libro;
    }
}

public class LibroEdicionEspecial extends LibroDecorator {
    public LibroEdicionEspecial(ILibro libro) {
```

Observador.java 1 ●

C: > Users > Liz > OneDrive > Escritorio > GRUPOPROY > ProyBibliotecaAcademica > ProyBibliotecaAcademica > sr

```java
package observador;

import java.util.ArrayList;
import java.util.List;

public class Notificador {
    private List<Observador> observadores = new ArrayList<>();

    public void agregarObservador(Observador o) {
        observadores.add(o);
    }

    public void eliminarObservador(Observador o) {
        observadores.remove(o);
    }

    public void notificar(String mensaje) {
        for (Observador o : observadores) {
            o.actualizar(mensaje);
        }
    }
}
public class UsuarioObservador implements Observador {
    private String nombre;

    public UsuarioObservador(String nombre) {
        this.nombre = nombre;
    }

    @Override
    public void actualizar(String mensaje) {
        System.out.println("Notificación para " + nombre + ": " + mensaje);
    }
```

Main.java

C: > Users > Liz > OneDrive > Escritorio > GRUPOPROY > ProyBibliotecaAcademica

```java
package modelo;

import javax.swing.*;

public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new BibliotecaUI();
            }
        });
    }
}
```

```java
package modelo;


import java.io.Serializable;
import java.util.Date;

public class Prestamo implements Serializable {
    private Usuario usuario;
    private Libro libro;
    private Date fechaPrestamo;
    private Date fechaDevolucion;

    public Prestamo(Usuario usuario, Libro libro, Date fechaPrestamo, Date fech
        this.usuario = usuario;
        this.libro = libro;
        this.fechaPrestamo = fechaPrestamo;
        this.fechaDevolucion = fechaDevolucion;
    }

    public Usuario getUsuario() {
        return usuario;
    }

    public Libro getLibro() {
        return libro;
    }

    public Date getFechaPrestamo() {
        return fechaPrestamo;
    }

    public Date getFechaDevolucion() {
        return fechaDevolucion;
    }

    @Override
    public String toString() {
```

```java
package modelo;

import java.util.ArrayList;
import java.util.List;

public class RepositorioLibros implements Repositorio<Libro> {
    private List<Libro> libros = new ArrayList<>();

    @Override
    public void guardar(Libro libro) {
        libros.add(libro);
    }

    @Override
    public List<Libro> listar() {
        return libros;
    }

    @Override
    public void modificar(Libro libro) {
        for (int i = 0; i < libros.size(); i++) {
            if (libros.get(i).getIsbn().equals(libro.getIsbn())) {
                libros.set(i, libro);
                break;
            }
```

```java
package modelo;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Date;

public class BibliotecaUI {
    private JFrame frame;
    private JTextArea areaUsuarios;
    private JTextArea areaLibros;
    private JTextArea areaPrestamos;
    private JTextField txtNombreUsuario, txtIdUsuario, txtEmail, txtTituloLibro, txtAutorLibro, txtIsbnLibro;

    public BibliotecaUI() {
        frame = new JFrame("Biblioteca Académica");
        frame.setSize(800, 600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout());

        // Panel de usuarios
        JPanel panelUsuarios = new JPanel(new GridLayout(3, 2));
        panelUsuarios.add(new JLabel("Nombre Usuario:"));
        txtNombreUsuario = new JTextField();
        panelUsuarios.add(txtNombreUsuario);

        panelUsuarios.add(new JLabel("ID Usuario:"));
        txtIdUsuario = new JTextField();
        panelUsuarios.add(txtIdUsuario);

        panelUsuarios.add(new JLabel("Email:"));
        txtEmail = new JTextField();
        panelUsuarios.add(txtEmail);

        JButton btnRegistrarUsuario = new JButton("Registrar Usuario");
```

10. CONCLUSION

El proyecto de **Biblioteca Académica en Java** implementa un sistema funcional para gestionar usuarios, libros y préstamos, utilizando principios de programación orientada a objetos, patrones de diseño (como Singleton y Factory), e interfaz gráfica con Swing. Se incluye, almacenamiento mediante archivos y serialización, y una arquitectura modular que facilita el mantenimiento y futuras ampliaciones.

Como:

- Intregracion de una Base de Datos
- Lectura de código por QR
- Notificaciones al correo electrónico
- Soporte multi usuario

Ventajas observadas en la estructura del sistema

- Modularidad: Cada clase tiene una responsabilidad clara (Usuario, Recurso, Prestamo).
- Extensibilidad: Fácil agregar nuevos tipos de recursos.
- Persistencia automática: Sin necesidad de intervención manual.

Qué se aplicó de los contenidos de la materia

- Uso de colecciones (ArrayList)
- Manejo de archivos (Persistencia)
- Programación orientada a objetos (POO)
- Implementación de herencia y polimorfismo
- Estructuras de control y validaciones

Qué se podría mejorar o ampliar

- Implementar una interfaz gráfica (por ejemplo, con JavaFX o Swing).
- Añadir búsqueda avanzada por autor, año, etc.
- Validaciones más robustas de entrada de datos.
- Uso de base de datos en lugar de archivos CSV.
- Implementar fechas con LocalDate para validaciones reales.

11. DISTRIBUCIÓN DE ROLES DEL EQUIPO

Mamani Quispe Gustavo Giovanni Desarrollo de clases modelo y persistencia de datos

Gomez Blanco Cash Israel Pruebas, documentación y reporte final

Agner Asistiri Yoel Americo	Desarrollo de diagramas UML
Rodriguez Mendez Gianina Madelein	Revisión de documentación y reporte final
Peredo Mamani Devon Richard	Apoyo en la codificacion
Echenique Mayta Lizeth Hortencia	Revisión de documentación y reporte final

12. ANEXOS

Código fuente organizado por paquetes

```
src/
├── bibliotecaAcademica/
│   ├── Main.java
│   ├── Biblioteca.java
│   ├── modelo/
│   │   ├── Recurso.java
│   │   ├── Libro.java
│   │   ├── RecursoDigital.java
│   │   ├── Usuario.java
│   │   └── Prestamo.java
│   ├── util/
│   │   └── FileManager.java
```

│ └── factory/

│ └── RecursoFactory.java

Archivos de prueba

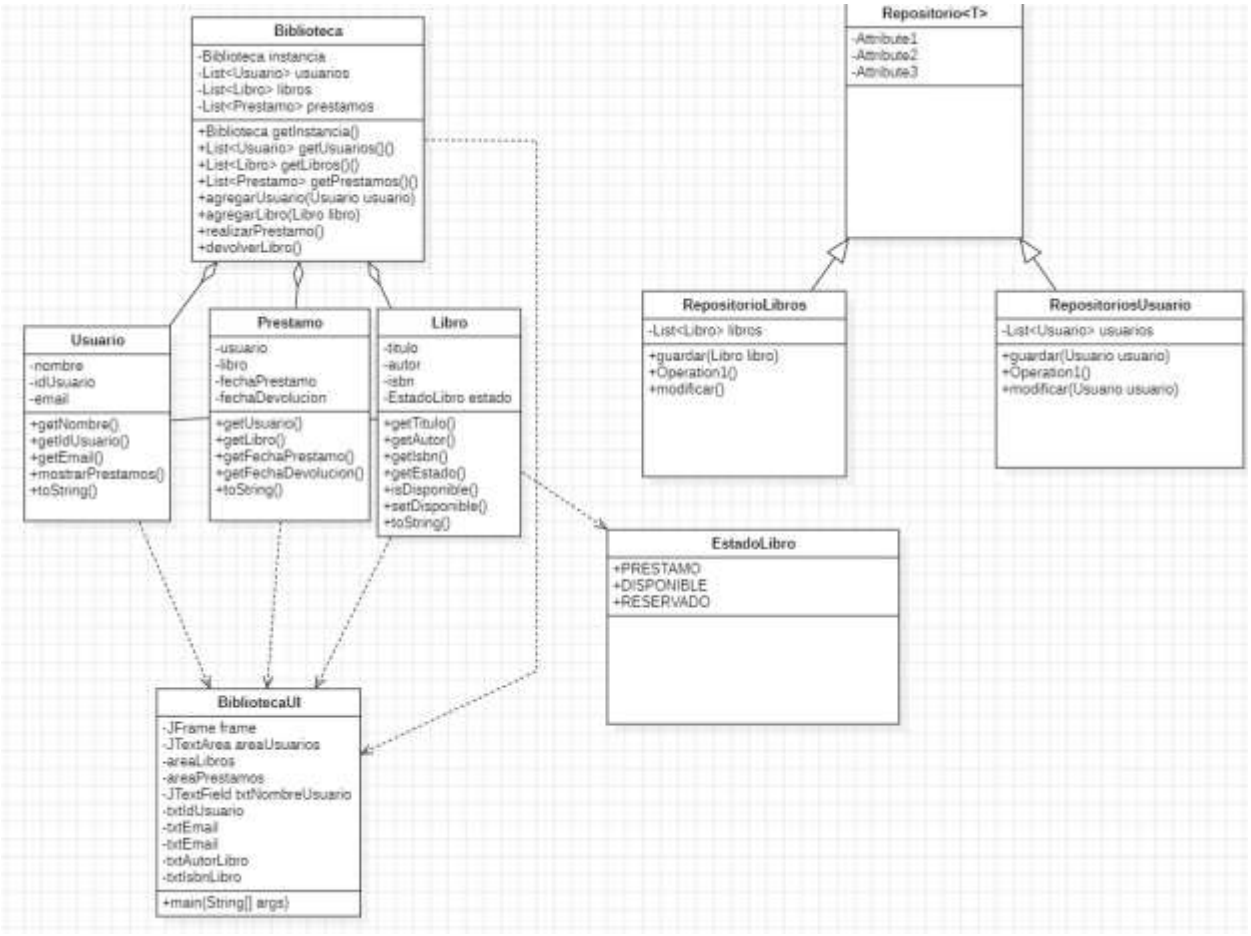
- recursos.csv
- usuarios.csv
- prestamos.csv
- Libro.csv
- recursoDigital.csv
- Biblioteca

Diagramas UML

Incluye:

Diagrama de clases mostrando herencia entre Recurso, Libro y RecursoDigital.

Diagrama de objetos para mostrar un préstamo activo.


Biblioteca
-Biblioteca instancia
-List<Usuario> usuarios
-List<Libro> libros
-List<Prestamo> prestamos
+Biblioteca getInstancia()
+List<Usuario> getUsuarios()()
+List<Libro> getLibros()()
+List<Prestamo> getPrestamos()()
+agregarUsuario(Usuario usuario)
+agregarLibro(Libro libro)
+realizarPrestamo()
+devolverLibro()
Repositorio<T>
-Atribute1
-Atribute2
-Atribute3
Usuario
-nombre
-idUsuario
-email
+getNombre()
+getIdUsuario()
+getEmail()
+mostrarPrestamos()
+toString()
Prestamo
-usuario
-libro
-fechaPrestamo
-fechaDevolucion
+getUsuario()
+getLibro()
+getFechaPrestamo()
+getFechaDevolucion()
+toString()
Libro
-titulo
-autor
-isbn
-EstadoLibro estado
+getTitulo()
+getAutor()
+getIsbn()
+getEstado()
+isDisponible()
+setDisponible()
+toString()
RepositorioLibros
-List<Libro> libros
+guardar(Libro libro)
+Operation1()
+modificar()
RepositoriosUsuario
-List<Usuario> usuarios
+guardar(Usuario usuario)
+Operation1()
+modificar(Usuario usuario)
EstadoLibro
+PRESTAMO
+DISPONIBLE
+RESERVADO
BibliotecaUI
-JFrame frame
-JTextArea areaUsuarios
-areaLibros
-areaPrestamos
-JTextField txtNombreUsuario
-txtIdUsuario
-txtEmail
-txtEmail
-txtAutorLibro
-txtIsbnLibro
+main(String[] args)