

# **Memoria Actividad Colaborativa 1: Implantación del sistema de una biblioteca**

Gerardo Diéguez Serpa  
Daniel González Garrote  
Laura Flores La Lueta

# Índice

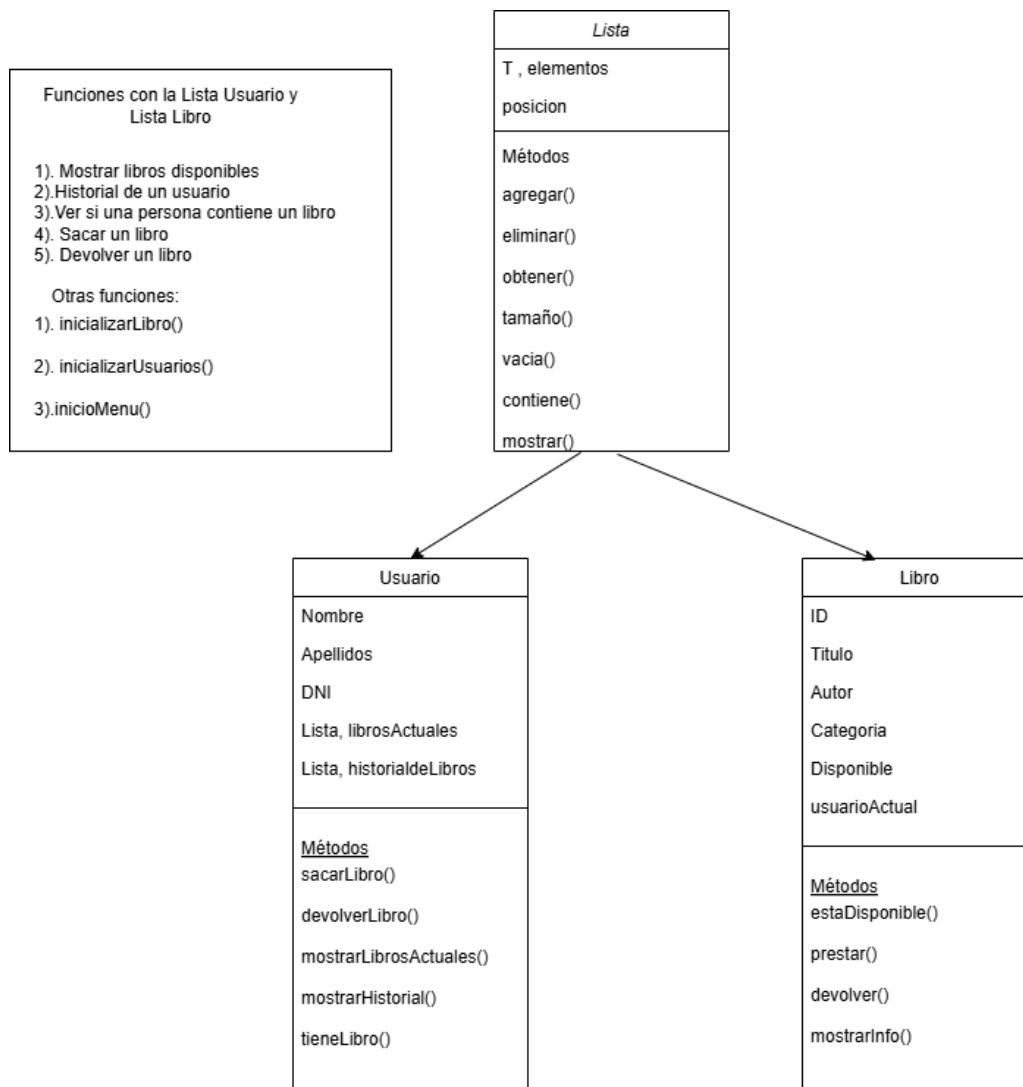
Introducción	2
Creación de clase Lista	3
Creación de clase Libro	7
Creación de clase Usuario	9
Funciones del programa	12
Inicialización de Libros	12
Inicialización de Usuarios	12
Función 1: Libros disponibles	13
Función 2: Historial de un usuario	15
Estructura general para las funciones 3,4,5	16
Funcionalidad y User Experience (UX)	18
Ejecución del programa	20
Conclusión	23

# Introducción

El presente proyecto tiene como objetivo el desarrollo de un sistema de gestión de una biblioteca, diseñado para ser implementado por cualquier persona física.

El desarrollo de este proyecto abarca el lenguaje de programación C++ , con la implementación de los contenidos de la primera unidad del tema 1, siendo estos: tipos de datos abstractos, introducción a programación orientada a objetos, creación de estructuras o plantillas, implementación de funciones y métodos, creación de clases, etc.

Para el desarrollo de esta actividad, se realizó el siguiente esquema para proyectar la idea principal del sistema de gestión de la biblioteca:



En este diagrama, se establecen las diferentes clases que utilizaremos en el desarrollo del sistema de gestión de la biblioteca, que a lo largo del documento se explicará el funcionamiento de ellas mismas. La clase Lista está compuesta por la estructura de una lista genérica. Hemos realizado esta clase con la ayuda del ejercicio “PilaGen1”, creando una plantilla con **template<typename T>**.

La clase Usuario está compuesta por diferentes funciones que comparten con la clase lista, ya que haremos referencia en algunos métodos como **contiene()**, **agregar()**, entre otros.

Por último, la clase Usuario tiene unas funcionalidades similares a clase Libro, pero en esta clase se hará referencia a métodos como **tieneLibro()**, **mostrarHistorial()**, etc.

Por otro lado, en el diagrama anterior se reflejaron las funcionalidades externas a las clases, es decir, las funciones que se van a realizar en la biblioteca, y que se establecerán en el menú. Estas funciones relacionan las listas de usuarios y libros para poder realizar los métodos y juntar ambas informaciones, para luego tener comparativa y poder establecer si se ha sacado un libro , si se ha devuelto, etc.

## Creación de clase Lista

### Atributos

La clase Lista está compuesta por los siguientes atributos, entre los que destacamos la creación de la plantilla con el uso de templates, teniendo en cuenta que “T” se utiliza para poder usar cualquier tipo de dato.

```
//Plantilla para listas genéricas
template<typename T>
class Lista{
    private:
        T elementos [50];
        int posición;
```

En la parte **privada**, definimos los elementos, establecidos por un valor determinado, ya que, al tratarse de una lista genérica, debe contener un valor asociado por defecto. También definimos una variable posición que utilizaremos en las funciones de esta clase para realizar condicionales, bucles, entre otros.

## Métodos

```

public:
    Lista() : posicion(0) {}
    //Función Añadir elemento
    void agregar(T elemento){
        if (posicion < 50){
            elementos[posicion] = elemento;
            posicion++;
        }else
        {
            std::cout<<"La lista está completa, no se puede añadir nada más" <<std::endl;
        }
    }
}

```

Dentro de la parte **pública**, encontramos el constructor de la clase y las diferentes funciones que vamos a usar, en concreto tenemos la función *agregar()*, cuya finalidad principal es añadir un elemento a la lista.

Para ello se emplea un condicional en el que se evalúa si la posición es menor a 50, que es el valor establecido por defecto, y si se cumple, se añadirá, del otro modo, se indicará por pantalla que la lista está completa.

```

//Sacar elemento de la lista
void eliminar(T elemento){
    for (int i = 0; i < posicion; i++)
    {
        if (elementos[i] == elemento){
            for (int k = i; k < posicion - 1; k++)
            {
                elementos[k] = elementos[k + 1];
            }
            posicion--;
        }
    }
}

```

El método **eliminar()** consiste en eliminar un elemento de la lista, esto se realiza mediante un bucle que recorre la posición, y en el que mediante un condicional, se evalúa si los elementos recorridos en el bucle son iguales a los elementos que queremos eliminar. Dentro se realiza un segundo bucle, con el fin de mover cada

elemento una posición hacia la izquierda, para que de esta forma se cubra el hueco que ha quedado por la eliminación del elemento anterior, de esta manera no se deja ningún espacio libre. En este método se utiliza el concepto de **anidación de bucles**: uso de bucles dentro de otros.

```
//Obtener elemento por índice
T& obtener(int indice)
{
    if (indice >= 0 && indice < posicion)
    {
        return elementos[indice];
    }
    return elementos[0];
}
```

El método **obtener()** tiene la funcionalidad de obtener el índice de un elemento. Esto se realiza mediante condicionales, en los que se evalúa si el índice es mayor o igual a 0 y el índice es menor que la posición. Si esto se cumple, devolverá los elementos de ese índice, y si no se cumple, devuelve el primer índice en la posición de los elementos [0].

```
//Tamaño de la lista
int tamano()
{
    return posicion;
}
```

El método **tamano()**, se utiliza para que nos devuelva la posición, haciendo referencia a un *return*, de esta manera, si nos preguntamos “**¿Cuáles son los elementos de la lista?**”, podemos llamar al método tamaño y nos devolverá con el valor de la posición.

```
//Ver si está vacía la lista
bool vacia()
{
    return posicion == 0;
}
```

El método **vacia()** devuelve *true* si no hay ningún elemento en la posición.

```
//Limpian la lista
void limpiar()
{
    posicion = 0;
}
```

El método **limpiar()**, lo que hace es dejar la posición a 0 , y por tanto esto hace que la lista quede “ limpia ”.

```
//Buscar elemento
bool contiene(T& elemento){
    for (int i = 0; i < posicion; i++){
        if (elementos[i] == elemento){
            return true;
        }
    }
    return false;
}
```

El método **contiene()** revisa uno a uno los elementos guardados, y sobre eso, se hace una condición para saber si alguno de los elementos guardados es igual al que le pasamos. De esta manera sabremos si lo contiene o no.

```
//Mostrar todos los elementos
void mostrar()
{
    for (int i = 0; i < posicion; i++)
    {
        cout << elementos[i] << endl;
    }
    cout << endl;
}
```

El método **mostrar()** recorre todos los elementos guardados y los muestra uno a uno.

# Creación de clase Libro

## Atributos

Los atributos de la clase se declaran como privados, siendo estos los siguientes:

```
int id;
string titulo;
string autor;
string categoria;
bool disponible;
string usuarioActual;
```

## Métodos

Funciones asignadas a la clase:

- **Constructor por defecto:**

Permite la creación de un objeto sin pasar información al momento de crearlo.

```
Libro() : id(0), titulo(""), autor(""), categoria(""), disponible(true), usuarioActual("") {}
```

- **Constructor con parámetros:**

Crea el objeto ya inicializado con datos.

```
Libro(int id, string titulo, string autor, string categoria)
      : id(id), titulo(titulo), autor(autor), categoria(categoría),
        disponible(true), usuarioActual("") {}
```

- **Getters:**

Acceso a los atributos.

```
int getId() const{return id;}
string getTitulo() const{return titulo;}
string getAutor() const{return autor;}
string getCategoría() const{return categoria;}
bool getDisponible() const{return disponible;}
string getUsuarioActual() const{return usuarioActual;}
```

- **bool estaDisponible()**

Comprueba si un libro se encuentra disponible.

```
bool estaDisponible() {  
    |   return disponible;  
    } }
```

*Nota: Esta función se puede hacer también con setDisponible*

- **void prestar(string dni)**

Permite asignar un libro a una persona.

- Atributo **bool disponible** pasa a ser *false*. Ya no se encuentra disponible.
- Se le asigna al atributo **string usuarioActual** el valor de **dni**.

```
void prestar(string dni) {  
    |   disponible = false;  
    |   usuarioActual = dni;  
    } }
```

- **void devolver()**

Caso contrario de **void prestar(string dni)**, se de asigna el libro a la persona.

- Atributo **bool disponible** pasa a ser *true*. Se encuentra disponible.
- Atributo **string usuarioActual** pasa a tener valor vacío.

```
void devolver() {  
    |   disponible = true;  
    |   usuarioActual = "";  
    } }
```

- **void mostrarInfo()**

Este método es necesario para mostrar los datos en la lista.

Muestra los atributos de la clase Libro. En el caso de **disponible**, si es verdadero, imprime por pantalla “Disponible”, en caso contrario imprime “No está disponible el libro actual, el usuario **usuarioActual** tiene el libro.”

```
void mostrarInfo()
{
    cout << "->Título: " << titulo << " || Autor: " << autor << " || Categoría: " << categoría << " || Disponible" << disponible << endl;
    if (!disponible) {
        cout << "No está disponible el usuario: " << usuarioActual << "tiene el libro" << endl;
    }
}
```

## Creación de clase Usuario

### Atributos

Los atributos privados, que solo son accesibles desde su clase son los siguientes:

```
class Usuario {
private:
    //Declaramos las variables de la clase
    string nombre;
    string apellidos;
    string dni;
    Lista<string> librosActuales;
    Lista<string> historialdeLibros;
```

Como atributos tenemos dos listas que son de tipo string que usa el usuario llamadas **librosActuales** e **historialdeLibros**.

### Métodos

Son las funciones que nos permiten realizar cosas con las clases.

- **Constructor por defecto:**

Inicializa todas las variables por defecto como cadenas vacías.

```
public:
    // Constructor por defecto con valores vacíos
    Usuario() : nombre(""), apellidos(""), dni("") {}
```

- **Constructor con parámetros:**

Inicializa cada atributo de la clase con los valores que correspondan.

```
// Constructor con parámetros
Usuario(string nombre, string apellidos, string dni)
    : nombre(nombre), apellidos(apellidos), dni(dni) {}
```

- **Getters:**

Devuelve los valores de los atributos de las clases.

```
string getNombre() const { return nombre; }
string getApellido() const { return apellidos; }
string getDNI() const { return dni; }
```

- **void sacarLibro():**

Cuando se hace uso de esta función es porque el usuario va a sacar un libro de la biblioteca. Se añade el título del libro, al usuario que le corresponde a la lista **librosActuales**.

```
//Función sacarLibro
void sacarLibro(string tituloLibro) {
    librosActuales.agregar(tituloLibro);
}
```

- **void devolverLibro():**

Esta función realiza la acción de devolver un libro a la biblioteca. De la lista **librosActuales** del usuario, en la que se encuentra el título del libro que actualmente tiene, se elimina ese título de la lista puesto que lo está devolviendo a la biblioteca, y se añade a la lista **historialdeLibros** si ese libro no estaba ya en esa lista porque lo haya cogido de nuevo y lo está devolviendo.

```
//Función devolverLibro
void devolverLibro(string tituloLibro) {
    librosActuales.eliminar(tituloLibro);
    // Solo se añade al historial si no estaba ya
    if (!historialdeLibros.contiene(tituloLibro))
        historialdeLibros.agregar(tituloLibro);
}
```

- **void mostrarLibrosActuales():**

Muestra los libros actuales que tiene un usuario. En el caso de que esté vacía, no muestra nada.

```
//Función mostrarLibrosActuales del usuario
void mostrarLibrosActuales() {
    cout << "LIBROS ACTUALES DE: " << nombre << " " << apellidos << endl;
    if (librosActuales.vacia()) {
        cout << " No tiene libros" << endl;
    } else {
        for (int i = 0; i < librosActuales.tamano(); i++)
            cout << " - " << librosActuales.obtener(i) << endl;
    }
}
```

- **void mostrarHistorial():**

Muestra el historial de un usuario de todos los libros que ha devuelto a la biblioteca.

```
//Función mostrarHistorial de libros de un usuario
void mostrarHistorial() {
    cout << "HISTORIAL DE LIBROS DE: " << nombre << " " << apellidos << endl;
    if (historialdeLibros.vacia()) {
        cout << " No ha devuelto ningun libro" << endl;
    } else {
        for (int i = 0; i < historialdeLibros.tamano(); i++)
            cout << " - " << historialdeLibros.obtener(i) << endl;
    }
}
```

- **bool tieneLibro():**

Comprueba si el usuario tiene actualmente un libro en concreto en préstamo.

```
//Función tiene libro , booleana si tiene , no tiene
bool tieneLibro(string tituloLibro) {
    return librosActuales.contiene(tituloLibro);
}
```

# Funciones del programa

## Inicialización de Libros

```
//Función para inicializar los libros
void inicializarLibro(Lista<Libro>& libros)
{
    //Libros de acción
    libros.agregar(elemento: Libro(id: 1, título: "El código Da Vinci", autor: "Dan Brown", categoría: "Acción"));
    libros.agregar(elemento: Libro(id: 2, título: "La chica del tren", autor: "Paula Hawkins", categoría: "Acción"));
    libros.agregar(elemento: Libro(id: 3, título: "Inferno", autor: "Dan Brown", categoría: "Acción"));
    libros.agregar(elemento: Libro(id: 4, título: "El juego de Ender", autor: "Orson Scott Card", categoría: "Acción"));
    //Libros de aventuras
    libros.agregar(elemento: Libro(id: 5, título: "El hobbit", autor: "J.R.R. Tolkien", categoría: "Aventuras"));
    libros.agregar(elemento: Libro(id: 6, título: "La isla del tesoro", autor: "R.L. Stevenson", categoría: "Aventuras"));
    libros.agregar(elemento: Libro(id: 7, título: "Las aventuras de Tom Sawyer", autor: "Mark Twain", categoría: "Aventuras"));
    libros.agregar(elemento: Libro(id: 8, título: "Viaje al centro de la Tierra", autor: "Julio Verne", categoría: "Aventuras"));
    //Libros de drama
    libros.agregar(elemento: Libro(id: 9, título: "Cien años de soledad", autor: "García Marquez", categoría: "Drama"));
    libros.agregar(elemento: Libro(id: 10, título: "La casa de los espíritus", autor: "Isabel Allende", categoría: "Drama"));
    libros.agregar(elemento: Libro(id: 11, título: "Crimen y castigo", autor: "Dostoyevski", categoría: "Drama"));
    libros.agregar(elemento: Libro(id: 12, título: "Los miserables", autor: "Victor Hugo", categoría: "Drama"));
    //Libros de comedia
    libros.agregar(elemento: Libro(id: 13, título: "El diario de Bridget Jones", autor: "Helen Fielding", categoría: "Comedia"));
    libros.agregar(elemento: Libro(id: 14, título: "Tres hombres en un bote", autor: "Jerome K. Jerome", categoría: "Comedia"));
    libros.agregar(elemento: Libro(id: 15, título: "La tía Julia y el escribidor", autor: "Mario Vargas Llosa", categoría: "Comedia"));
    libros.agregar(elemento: Libro(id: 16, título: "Historias de cronopios", autor: "Julio Cortázar", categoría: "Comedia"));
    //Libros de romanticismo
    libros.agregar(elemento: Libro(id: 17, título: "Orgullo y prejuicio", autor: "Jane Austen", categoría: "Romanticismo"));
    libros.agregar(elemento: Libro(id: 18, título: "Cumbres borrascosas", autor: "Emily Bronte", categoría: "Romanticismo"));
    libros.agregar(elemento: Libro(id: 19, título: "Jane Eyre", autor: "Charlotte Bronte", categoría: "Romanticismo"));
    libros.agregar(elemento: Libro(id: 20, título: "Romeo y Julieta", autor: "Shakespeare", categoría: "Romanticismo"));
}
```

Realizada por **inicializarLibro()**: Crea todos los libros de la biblioteca, en concreto, veinte libros divididos en cuatro categorías (acción, aventuras, drama, comedia, romanticismo). Estos libros mediante el método **agregar()**, mencionado anteriormente en la clase Lista, agrega los libros con sus respectivos atributos a la lista libros.

## Inicialización de Usuarios

```
//Función para inicializar los usuarios
void inicializarUsuarios(Lista<Usuario>& usuarios)
{
    usuarios.agregar(elemento: Usuario(nombre: "Daniel", apellidos: "Gonzalez Garrote", dni: "12345632A"));
    usuarios.agregar(elemento: Usuario(nombre: "Gerardo", apellidos: "Diequez Serpa", dni: "12348732X"));
    usuarios.agregar(elemento: Usuario(nombre: "Laura", apellidos: "Flores Lalveta", dni: "12345321B"));
    usuarios.agregar(elemento: Usuario(nombre: "Domingo", apellidos: "Garcia Mendez", dni: "12752632P"));
    usuarios.agregar(elemento: Usuario(nombre: "Paco", apellidos: "Gonzalez Sanz", dni: "12345632A"));
    usuarios.agregar(elemento: Usuario(nombre: "Maria", apellidos: "Gutierrez Labrador", dni: "12345632A"));
}
```

Realizado por **inicializarUsuarios()**, tiene un funcionamiento similar a la inicialización de los libros, pero esta vez, a diferencia del método anterior, se llama a la lista usuarios, para posteriormente agregar los usuarios creados con los diferentes atributos a la lista usuarios.

### *Función 1: Libros disponibles*

```
//Función 1: Mostrar libros disponibles
void mostrarLibrosDisponibles(Lista<Libro>& libros)
{
    cout << "---Libros disponibles---" << endl;
    bool disponible = false;
    string categoriaActual = "";
    //Recorremos todos los libros
    for ( int i = 0; i<libros.tamano(); i++)
    {
        //Comprobamos si está disponible
        if (libros.obtener(i).getDisponible())
        {
            //Si está disponible, y la categoría es distinta entonces mostramos la actual
            if (libros.obtener(i).getCategoria() != categoriaActual)
            {
                categoriaActual = libros.obtener(i).getCategoria();
                cout << "\n ----" << categoriaActual << "----" << endl;
            }
            libros.obtener(i).mostrarInfo();
            disponible = true;
        }
    }
    //Si no está disponible
    if (!disponible)
    {
        cout << "Libros no disponibles" << endl;
    }
}
```

**mostrarLibrosDisponibles(libros)**: Muestra por pantalla toda la información sobre los libros disponibles en la biblioteca.

**void mostrarLibrosDisponibles(Lista<Libro>& libros)**: Recorre la lista de libros y los muestra uno por uno por pantalla. Cada vez que la categoría cambia, se imprime un encabezado indicando la nueva categoría actual del libro. Cada libro llama al método de la clase libro **mostrarInfo()** para mostrar sus datos.

```
void mostrarLibrosDisponibles(Lista<Libro>& libros)
{
    cout << "---Libros disponibles---" << endl;
    bool disponible = false;
    string categoriaActual = "";
    //Recorremos todos los libros
    for ( int i = 0; i<libros.tamano(); i++)
    {
        //Comprobamos si está disponible
        if (libros.obtener(i).getDisponible())
        {
            //Si está disponible, y la categoria es distinta entonces mostramos la actual
            if (libros.obtener(i).getCategoria() != categoriaActual)
            {
                categoriaActual = libros.obtener(i).getCategoria();
                cout << "\n ---" << categoriaActual << "----" << endl;
            }
            libros.obtener(i).mostrarInfo();
            disponible = true;
        }
    }
    //Si no está disponible
    if (!disponible)
    {
        cout << "Libros no disponibles" << endl;
    }
}
```

## Función 2: Historial de un usuario

```

//Función 2: Historial de un usuario
void historialUsuario(Lista<Usuario>& usuarios)
{
    //Declaramos las variables que vamos a usar en el metodo
    string dni;
    cout << "---Escribe el dni del usuario---" << endl;
    //Recogemos el dato escrito por el usuario
    cin >> dni;
    //Declaramos una variable a -1
    int indice = -1;
    //Bucle para recorrer la lista de libros desde la posición 0
    for ( int i = 0; i<usuarios.tamano(); i++)
    {
        //Si el DNI de la posicion i , es igual al dni que queremos
        if (usuarios.obtener(i).getDNI() == dni)
        {
            indice = i;
            break;
        }
    }
    if (indice == -1)
    {
        cout << "No hemos encontrado ningun usuario con ese DNI" << endl;
    }
    //Mostramos los libros actuales del usuario y mostramos el historial
    usuarios.obtener(indice).mostrarLibrosActuales();
    usuarios.obtener(indice).mostrarHistorial();
}

```

Realizada por **historialUsuario**, muestra todo el historial de libros devueltos de un usuario.

Se recoge en los usuarios, el índice y mostramos los libros actuales y el historial de los libros asociados a ese usuario. Conociendo el DNI del usuario, tenemos acceso a la información de los libros que tiene actualmente y a su historial.

## Estructura general para las funciones 3,4,5

```

//Declaramos las variables que vamos a usar en el metodo
string dni;
string tituloLibro;
cout << "----Escribe el dni del usuario---" << endl;
cin >> dni;
cin.ignore(); //Lo ponemos para quitar caracteres no deseados, y aseguramos que se lea bien
cout << "----Escribe el título del libro--" << endl;
getline(&cin, &tituloLibro); //Se utila para que podamos leer los espacios en blanco
//Iniciamos en -1 , que será para indicar que no se ha encontrado
int indiceUsuario = -1;
//Bucle para recorrer la lista de libros desde la posición 0
for ( int i = 0; i<usuarios.tamano(); i++)
{
    //Si el DNI de la posicion i , es igual al dni que queremos
    if (usuarios.obtener(i).getDNI() == dni)
    {
        indiceUsuario = i;
        break;
    }
}
//Iniciamos en -1 , que será para indicar que no se ha encontrado
int indiceLibro = -1;
//Bucle para recorrer la lista de libros desde la posición 0
for ( int i = 0; i<libros.tamano(); i++)
{
    //Si el título de la posicion i , es igual al título que queremos
    if (libros.obtener(i).getTitulo() == tituloLibro)
    {
        indiceLibro = i;
        break;
    }
}

```

En esta imagen se puede observar la estructura que se ha utilizado para las diferentes funciones 3, 4, 5. Se ha realizado de la misma manera para todas ellas, y se ha cambiado los condicionales para que se lleven a cabo las funcionalidades requeridas para cada una.

A continuación se explicarán los diferentes condicionales que se han añadido a las diferentes funciones:

### Función 3 verificarLibroEnUsuario():

```
//Comprobamos si es información existe
if (usuarios.obtener(indiceUsuario).tieneLibro( & tituloLibro))
{
    cout << usuarios.obtener(indiceUsuario).getNombre() << " si tiene el libro: " << tituloLibro << endl;
} else
{
    cout << usuarios.obtener(indiceUsuario).getNombre() << " no tiene el libro: " << tituloLibro << endl;
}
```

Una vez recogida la información sobre el título del libro y el DNI del usuario, se comprueba en este condicional que la información ofrecida por el usuario es correcta, y por tanto se confirma si tiene el libro ese usuario, o de otro modo no lo tiene.

En esta función , se relacionan ambas listas usuario y libro , de tal manera que en los parámetros de la función tiene la siguiente estructura **Lista<Libro>& libros**, **Lista<Usuario>& usuarios**. Esta estructura contiene “&”, utilizada para hacer referencias.

#### **Función 4 sacarLibro():**

```
//Obtenemos el libro en la posición indiceLibro
libros.obtener(indiceLibro).prestar( & dni);
//Obtenemos el libro en la posición indiceUsuario
usuarios.obtener(indiceUsuario).sacarLibro( & tituloLibro);

cout << "---SE HA SACADO EL LIBRO---" << endl;
cout << "El usuario: " << usuarios.obtener(indiceUsuario).getNombre() <<
    "se ha llevado el libro " << libros.obtener(indiceLibro).getTitulo() << endl;
```

En esta función la diferencia principal que podemos observar es que se obtiene **indiceLibro** y se añade la función **prestar()**. Dentro se introduce el DNI del usuario.

De esta manera, podemos conocer qué usuario tiene prestado el libro. Para el usuario, se obtiene **indiceUsuario** y se llama a la función **sacarLibro()**, donde se introduce el título del libro seleccionado.

#### **Función 5 devolverLibro():**

```
//Obtenemos el libro en la posición indiceLibro e indicamos el metodo devolver
libros.obtener(indiceLibro).devolver();
//Obtenemos el libro en la posición indiceUsuario y devolvemos el libro con el titulo
usuarios.obtener(indiceUsuario).devolverLibro(& tituloLibro);

cout << "---SE HA DEVUELTO EL LIBRO--" << endl;
cout << "El usuario: " << usuarios.obtener(indiceUsuario).getNombre() <<
"ha devuelto el libro " << libros.obtener(indiceLibro).getTitulo() << endl;
```

En esta función se ha obtenido **indiceLibro**, que será el que se devolverá. También se obtiene **indiceUsuario**, y se llama a la función **devolverLibro()**, donde se refleja el título del libro que se quiere devolver. Esta función al igual que las mencionadas anteriormente, llama a las dos listas para ser usadas en la función y poder hacer referencias a “**usuarios**” y “**libros**”.

## Funcionalidad y User Experience (UX)

En primer lugar para saber cómo el usuario actúa con nuestra interfaz de gestión de biblioteca, necesitamos saber qué es lo que ve el usuario, para ello necesitamos la función **inicioMenu()**:

```
//Función para mostrar el menú por pantalla
void inicioMenu()
{
    cout << "-----MENU DE GESTION DE BIBLIOTECA-----" << endl;
    cout << "||| 1. Mostrar libros disponibles      |||" << endl;
    cout << "||| 2. Historial de un usuario       |||" << endl;
    cout << "||| 3. Ver si una persona tiene un libro|||" << endl;
    cout << "||| 4. Sacar un libro                 |||" << endl;
    cout << "||| 5. Devolver un libro              |||" << endl;
    cout << "||| 0. Salir                         |||" << endl;
    cout << "-----" << endl;
}
```

Esta función permite al usuario ver por pantalla el menú de la biblioteca. Para ello se hace referencia en el *main*, llamando a la función de la siguiente manera: **inicioMenu()**.

Una vez mostrado el menú por pantalla, llega la siguiente parte: referenciar cada función en un *switch case*, donde el usuario elige qué opción quiere realizar.

```

int main(){
    do
    {
        //Inicializamos el menú, llamando a la función inicioMenu()
        inicioMenu();
        cin >> opcion;
        switch (opcion)
        {
            //Primer caso: llamamos a la primera función ( mostrarLibrosDisponibles )
            case 1:
                mostrarLibrosDisponibles(&libros);
                break;
            //Segundo caso: llamamos a la primera función ( historialUsuario )
            case 2:
                historialUsuario(&usuarios);
                break;
            //Tercer caso: llamamos a la primera función ( verificarLibroEnUsuario )
            case 3:
                verificarLibroEnUsuario(&libros, &usuarios);
                break;
            //Cuarto caso: llamamos a la primera función ( sacarLibro )
            case 4:
                sacarLibro(&libros, &usuarios);
                break;
            //Quinto caso: llamamos a la primera función ( devolverLibro )
            case 5:
                devolverLibro(&libros, &usuarios);
                break;
            //Si pulsa 0, saldrá del bucle
            case 0:
                cout<<"Saliendo de la biblioteca, HASTA OTRO DIA DE LECTURA !" << endl;
                break;
            //Si se pulsa otra opción que no existe, se muestra el siguiente mensaje
            default:
                cout<<"La opción que has escogido, no es válida, prueba otra vez"<<endl;
        }
        //Mientras sea distinto de 0 el bucle continua
    }while(opcion != 0);
}

```

Se realiza un bucle *do while* para que se pueda salir de la ejecución del programa cuando el usuario pulse el número 0. Si es un número distinto, el programa seguirá funcionando.

## Ejecución del programa

```
-----MENU DE GESTION DE BIBLIOTECA-----
||| 1. Mostrar libros disponibles      || |
||| 2. Historial de un usuario      ||
||| 3. Ver si una persona tiene un libro|||
||| 4. Sacar un libro               ||
||| 5. Devolver un libro            ||
||| 0. Salir                         ||
-----
1

---Libros disponibles---

----Accion---
->Titulo: El codigo Da Vinci || Autor: Dan Brown || Categoria: Accion || Disponible1
->Titulo: La chica del tren || Autor: Paula Hawkins || Categoria: Accion || Disponible1
->Titulo: Inferno || Autor: Dan Brown || Categoria: Accion || Disponible1
->Titulo: El juego de Ender || Autor: Orson Scott Card || Categoria: Accion || Disponible1

----Aventuras---
->Titulo: El hobbit || Autor: J.R.R. Tolkien || Categoria: Aventuras || Disponible1
->Titulo: La isla del tesoro || Autor: R.L. Stevenson || Categoria: Aventuras || Disponible1
->Titulo: Las aventuras de Tom Sawyer || Autor: Mark Twain || Categoria: Aventuras || Disponible1
->Titulo: Viaje al centro de la Tierra || Autor: Julio Verne || Categoria: Aventuras || Disponible1
```

Cuando el usuario presiona la primera opción “**1. Mostrar libros disponibles**”, se muestra la información de todos los libros, reflejando su disponibilidad, y las diferentes categorías con las diferentes cabeceras como se explicaba en el bucle de la función **mostrarLibrosDisponibles**.

```
-----MENU DE GESTION DE BIBLIOTECA-----
||| 1. Mostrar libros disponibles      |||
||| 2. Historial de un usuario       |||
||| 3. Ver si una persona tiene un libro|||
||| 4. Sacar un libro                |||
||| 5. Devolver un libro             |||
||| 0. Salir                         |||
-----
```

2

---Escribe el dni del usuario---

12345632A

LIBROS ACTUALES DE: Daniel Gonzalez Garrote

No tiene libros

HISTORIAL DE LIBROS DE: Daniel Gonzalez Garrote

No ha devuelto ningun libro

Por otro lado, si el usuario intenta acceder a su historial de libros, mediante la opción **“2. Historial de un usuario”**, y no ha sacado ni devuelto previamente uno, el historial mostrará que no tiene ningún libro disponible.

```
-----MENU DE GESTION DE BIBLIOTECA-----
||| 1. Mostrar libros disponibles      |||
||| 2. Historial de un usuario       |||
||| 3. Ver si una persona tiene un libro|||
||| 4. Sacar un libro                |||
||| 5. Devolver un libro             |||
||| 0. Salir                         |||
-----
```

3

---Escribe el dni del usuario---

12345632A

---Escribe el titulo del libro--

*Los miserables*

---El libro no se ha encontrado--

Daniel si tiene el libro: Los miserables

En este caso, el usuario ha seleccionado la tercera opción “**3. Ver si una persona tiene un libro**”, donde se quería comprobar si el usuario, introduciendo DNI “12345632A” y libro “Los miserables”, lo tenía. En este caso, el usuario tenía el libro.

```
-----MENU DE GESTION DE BIBLIOTECA-----
||| 1. Mostrar libros disponibles      || |
||| 2. Historial de un usuario      ||
||| 3. Ver si una persona tiene un libro|||
||| 4. Sacar un libro                ||
||| 5. Devolver un libro              ||
||| 0. Salir                         ||
-----
4

---Escribe el dni del usuario---
12345632A

---Escribe el titulo del libro---
Los miserables

---SE HA SACADO EL LIBRO---
El usuario: Daniel se ha llevado el libro Los miserables
```

El usuario esta vez ha seleccionado la opción “**4. Sacar un libro**”. Para ello ha tenido que introducir su DNI y el título del libro que quería sacar. Como podemos observar, se ha llevado el libro exitosamente.

```
-----MENU DE GESTION DE BIBLIOTECA-----
||| 1. Mostrar libros disponibles      |||
||| 2. Historial de un usuario       |||
||| 3. Ver si una persona tiene un libro|||
||| 4. Sacar un libro                |||
||| 5. Devolver un libro             |||
||| 0. Salir                         |||
-----
5

---Escribe el dni del usuario---
12345632A

---Escribe el titulo del libro--
Los miserables

---SE HA DEVUELTO EL LIBRO--
El usuario: Danielha devuelto el libro Los miserables
```

Después de una larga semana de lectura, el usuario regresó a la biblioteca, y esta vez quería devolver el libro que había sacado.

Para ello pulsa la opción “**5. Devolver un libro**” y nuevamente introdujo por pantalla DNI y el título del libro que quería devolver. Y como podemos observar el libro se ha devuelto con éxito.

## Conclusión

A lo largo de este proyecto, hemos desarrollado un sistema de gestión de biblioteca funcional en C++, aplicando los conceptos fundamentales de la programación con estructuras lineales, como “templates” o “listas”, las cuales nos han permitido utilizarlas para almacenar objetos de tipo “Libro” y “Usuario”.

El proyecto ha reforzado el uso de clases con sus atributos privados, constructores, getters y métodos propios, otro de los principales aprendizajes que hemos tenido en este proyecto es el uso de “&”, ya que sin esto no habría sido posible guardar los datos correctamente, donde los cambios se perdían cuando se terminaba una función y provocaba que los datos no se guardaban correctamente.

**Enlace al repositorio de github:**  
<https://github.com/daaaaniel242/biblioteca>