Sistema de Gestión de Biblioteca - Documentación Técnica

Tabla de Contenidos

Introducción

Este proyecto implementa un **Sistema de Gestión de Biblioteca** desarrollado en C++ que permite gestionar libros, usuarios y préstamos. El sistema está diseñado aplicando conceptos fundamentales de la programación orientada a objetos (POO) y estructuras de datos.

Objetivos del Proyecto

- Aplicar conceptos de POO (clases, objetos, encapsulación)
- Implementar estructuras de datos básicas (arrays, enumeraciones)
- Utilizar plantillas (templates) para crear código reutilizable
- Gestionar memoria de forma eficiente usando arrays estáticos

---

Fundamentos Teóricos

1. Programación Orientada a Objetos (POO)

1.1 Encapsulación

La **encapsulación** es uno de los pilares fundamentales de la POO que consiste en:

- **Ocultación de datos**: Los atributos de una clase se declaran como `private`
- **Acceso controlado**: Se proporcionan métodos públicos (getters/setters) para acceder a los datos
- **Protección de la integridad**: Los datos solo pueden ser modificados a través de métodos controlados

Implementación en el código:

```
class Libro {
private:
    int id_libro;           // Datos ocultos
    std::string nombre_libro;
    std::string autor_libro;
```

```
    Genero genero_libro;

public:
    int getId() const { return id_libro; }     // Acceso controlado
    void setId(int id) { id_libro = id; }      // Modificación controlada
};
```

1.2 Abstracción

La **abstracción** permite modelar entidades del mundo real como clases, enfocándose en las características esenciales:

- **Clase Libro**: Representa un libro con sus propiedades fundamentales
- **Clase Usuario**: Modela un usuario de la biblioteca con su información personal y libros
- **Clase Biblioteca**: Abstrae el concepto de una biblioteca como contenedor de libros

2. Enumeraciones (enum class)

Las **enumeraciones** proporcionan una forma de definir constantes con nombre, mejorando la legibilidad del código:

```
enum class Genero {
    Accion, Drama, Comedia, Romantico, Aventura, Fantasia
};
```

3. Plantillas (Templates)

Las **plantillas** permiten escribir código genérico que funciona con diferentes tipos de datos:

```
template <typename T>
class Biblioteca {
    // Puede almacenar cualquier tipo T
};
```

Beneficios:

- **Reutilización**: Un solo código para múltiples tipos
- **Type Safety**: Verificación de tipos en tiempo de compilación
- **Eficiencia**: No hay overhead de runtime

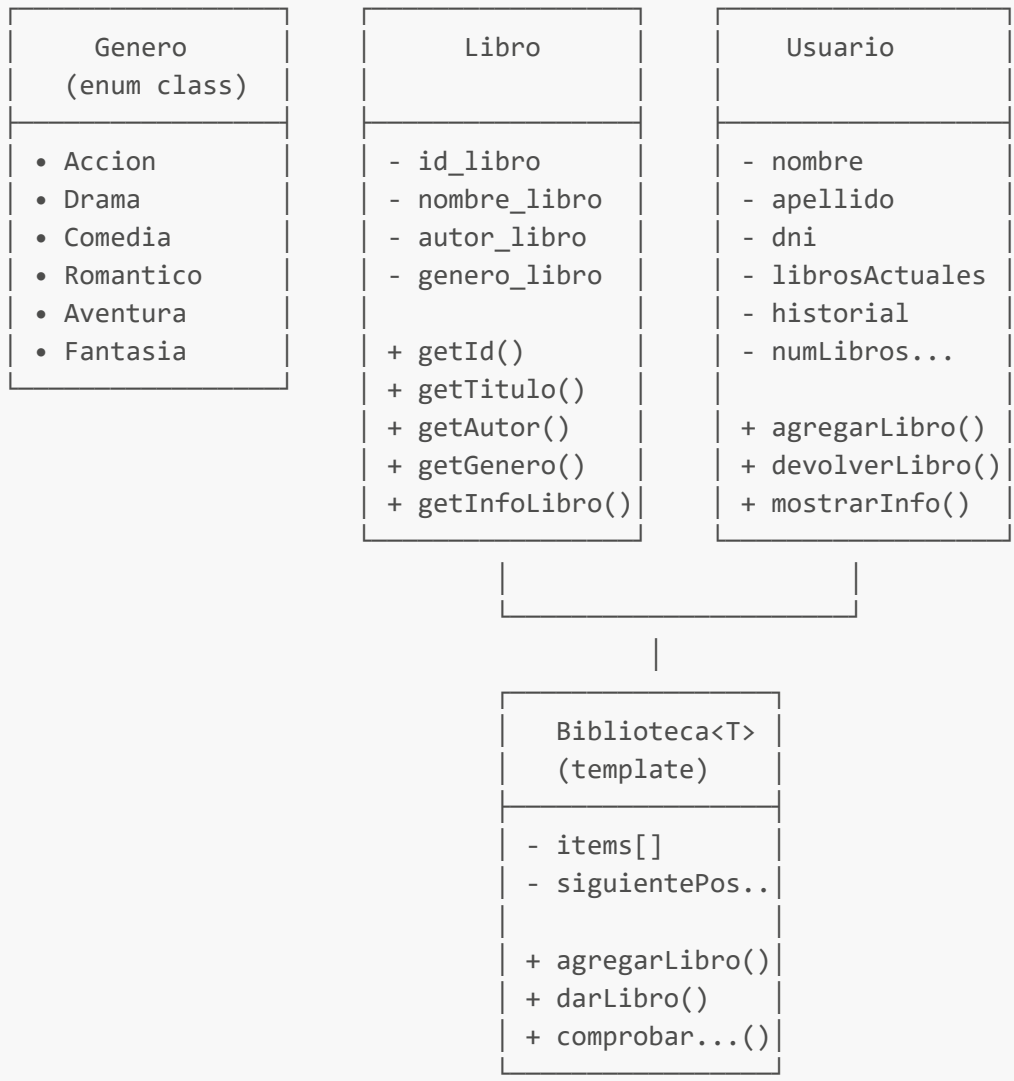
4. Gestión de Memoria con Arrays Estáticos

El proyecto utiliza **arrays estáticos** para gestionar las colecciones:

- **Ventajas**: Gestión automática de memoria, acceso rápido
- **Limitaciones**: Tamaño fijo definido en tiempo de compilación
- **Implementación**: `static const int MAX_LIBROS = 20;`

Arquitectura del Sistema

Diagrama de Clases



Relaciones entre Clases

1. **Composición**: `Usuario` contiene arrays de objetos `Libro`
2. **Agregación**: `Biblioteca` puede contener objetos de tipo `T` (genérico)
3. **Uso**: `Libro` utiliza la enumeración `Genero`

Implementación del Código

1. Clase Libro

Propósito: Representar un libro con sus atributos fundamentales.

Características implementadas:

- **Constructores**: Constructor por defecto y constructor parametrizado

- **Encapsulación**: Atributos privados con métodos de acceso públicos
- **Métodos de información**: `getInfoLibro()` para mostrar los datos del libro

Conceptos aplicados:

- Inicialización de miembros usando lista de inicialización
- Métodos const para operaciones de solo lectura
- Separación de interfaz e implementación

2. Clase Usuario

Propósito: Gestionar la información de usuarios y sus libros prestados.

Funcionalidades implementadas:

- **Gestión de préstamos**: Agregar y devolver libros
- **Historial**: Mantener registro de libros leídos
- **Límites**: Control de límite máximo de libros (MAX_LIBROS = 20)
- **Consultas**: Verificar si un usuario tiene un libro específico

Algoritmos utilizados:

- **Búsqueda lineal**: Para encontrar libros por ID
- **Gestión de arrays**: Agregar/eliminar elementos manteniendo orden

3. Plantilla Biblioteca

Propósito: Contenedor genérico para gestionar cualquier tipo de objeto.

Implementación de template:

```
template <typename T>
class Biblioteca {
    // Implementación genérica
};
```

Métodos implementados:

- `agregarLibro()`: Añade un elemento al contenedor
- `darLibro()`: Transfiere un libro del inventario al usuario
- `comprobarLibroDisponible()`: Verifica disponibilidad

4. Gestión de Enumeraciones

La función auxiliar `generoToString()` convierte valores enum a strings:

- **Patrón Switch**: Mapeo exhaustivo de todos los casos
- **Valor por defecto**: Manejo de casos no contemplados
- **Función inline**: Optimización de rendimiento

Manual de Usuario

Funcionalidades Disponibles

1. Gestión de Libros

- **Crear libros**: Instanciar objetos con título, ID, autor y género
- **Consultar información**: Ver todos los datos de un libro
- **Categorización**: Organizar por géneros predefinidos

2. Gestión de Usuarios

- **Registro de usuarios**: Crear perfiles con nombre, apellido y DNI
- **Préstamos**: Asignar hasta 20 libros por usuario
- **Devoluciones**: Procesar retorno de libros al inventario
- **Historial**: Consultar libros previamente leídos

3. Sistema de Biblioteca

- **Inventario**: Gestionar colección de hasta 20 libros
- **Disponibilidad**: Verificar si un libro está disponible
- **Préstamos**: Transferir libros del inventario a usuarios

Limitaciones del Sistema

1. **Capacidad máxima**: 20 libros por usuario y 20 libros en biblioteca
2. **Persistencia**: Los datos no se guardan al cerrar el programa
3. **Interfaz**: Solo interfaz de línea de comandos
4. **Búsqueda**: Solo búsqueda por ID numérico

Ejemplos de Uso

Ejemplo 1: Flujo Básico del Programa

```
// El programa actual ejecuta automáticamente:

1. Crear libros de ejemplo:
   - "El Hobbit" (ID: 1, Autor: J.R.R. Tolkien, Género: Fantasía)
   - "Cien años de soledad" (ID: 2, Autor: Gabriel García Márquez, Género: Drama)
   - "El Quijote" (ID: 3, Autor: Miguel de Cervantes, Género: Aventura)

2. Crear usuario de prueba:
   - Nombre: Juan Pérez
   - DNI: 12345678

3. Mostrar información del usuario (inicialmente sin libros)

4. Agregar libros al usuario:
   - Agregar "El Hobbit"
```

```
   - Agregar "Cien años de soledad"

5. Mostrar libros actuales del usuario

6. Devolver un libro (El Hobbit - ID: 1)

7. Mostrar estado final:
   - Libros actuales (solo "Cien años de soledad")
   - Historial (contiene "El Hobbit")
```

Ejemplo 2: Salida Esperada

```
Usuario: Juan Perez
DNI: 12345678
Libros actuales: 0
Historial de libros leídos: 0

Libro agregado a Juan Perez.
Libro agregado a Juan Perez.

Libros actuales de Juan:
ID: 1
Titulo: El Hobbit
Autor: J.R.R. Tolkien
Genero: Fantasia
------------------
ID: 2
Titulo: Cien anos de soledad
Autor: Gabriel Garcia Marquez
Genero: Drama
------------------

Libro devuelto correctamente.

Libros actuales de Juan:
ID: 2
Titulo: Cien anos de soledad
Autor: Gabriel Garcia Marquez
Genero: Drama
------------------

Historial de libros de Juan:
ID: 1
Titulo: El Hobbit
Autor: J.R.R. Tolkien
Genero: Fantasia
------------------
```

Ejemplo 3: Personalización del Código

Para modificar el comportamiento del programa:

```
// Agregar nuevos géneros
enum class Genero {
    // Géneros existentes...
    Terror,      // Nuevo género
    Biografia,   // Nuevo género
    Ciencia      // Nuevo género
};

// Crear libros personalizados
Libro miLibro("1984", 10, "George Orwell", Genero::Drama);

// Crear usuarios personalizados
Usuario miUsuario("Ana", "García", "87654321");

// Usar la biblioteca template
Biblioteca<Libro> miBiblioteca;
miBiblioteca.agregarLibro(miLibro);
```

Conclusiones

Este proyecto demuestra la aplicación práctica de conceptos fundamentales de C++ y programación orientada a objetos:

Conceptos POO Implementados

- ☑ **Encapsulación**: Datos privados con acceso controlado
- ☑ **Abstracción**: Modelado de entidades del mundo real
- ☑ **Reutilización**: Uso de templates para código genérico

Estructuras de Datos Utilizadas

- ☑ **Arrays estáticos**: Gestión eficiente de memoria
- ☑ **Enumeraciones**: Type safety y legibilidad
- ☑ **Templates**: Programación genérica

Buenas Prácticas Aplicadas

- ☑ **Const correctness**: Métodos const para operaciones de solo lectura
- ☑ **Inicialización**: Lista de inicialización en constructores
- ☑ **Separación de responsabilidades**: Cada clase tiene un propósito específico

Este sistema proporciona una base sólida para entender cómo se estructuran aplicaciones orientadas a objetos y puede ser extendido para incluir funcionalidades más avanzadas como persistencia de datos, interfaz gráfica, o sistemas de búsqueda más sofisticados.

Autores: Juan Labajo, Álvaro Hernández y Esteban Damián.
Fecha: Octubre 2025
Versión: 1.0