```
// Proyecto: Sistema de Gestión de Libros Electrónicos
// Versión consola sin base de datos (datos en memoria)
// Lenguaje: Go (Golang)
// Se aplican: Encapsulación, manejo de errores, interfaces y comentarios claros
// Autor: Carlos Maruri

package main

import (
    "bufio"
    "errors"
    "fmt"
    "os"
    "strconv"
    "strings"
)

// Estructura Libro: representa un libro electrónico con campos encapsulados (privados)
type Libro struct {
    ID       int
    titulo    string // campo privado
    autor     string // campo privado
    categoria string // campo privado
    anio      int    // campo privado
}

// Métodos públicos (Getters)
func (l Libro) Titulo() string    { return l.titulo }
func (l Libro) Autor() string     { return l.autor }
```

```
func (l Libro) Categoria() string { return l.categoria }
func (l Libro) Anio() int        { return l.anio }

// Métodos públicos (Setters)
func (l *Libro) SetTitulo(t string)    { l.titulo = t }
func (l *Libro) SetAutor(a string)     { l.autor = a }
func (l *Libro) SetCategoria(c string) { l.categoria = c }
func (l *Libro) SetAnio(a int)         { l.anio = a }

// Interfaz que define el contrato para gestionar libros (CRUD)
type ServicioLibro interface {
    Listar() []Libro
    Buscar(titulo, autor string) []Libro
    Agregar(libro Libro) error
    Actualizar(id int, libro Libro) error
    Eliminar(id int) error
}

// Implementación en memoria de ServicioLibro
// Simula almacenamiento temporal de libros
type MemoriaLibros struct {
    datos    []Libro // slice para almacenar libros
    ultimoID int     // para asignar IDs autoincrementales
}

// Listar devuelve todos los libros almacenados
func (m *MemoriaLibros) Listar() []Libro {
    return m.datos
}
```

```
// Buscar filtra libros por título y autor (insensible a mayúsculas)
func (m *MemoriaLibros) Buscar(titulo, autor string) []Libro {
    var resultado []Libro
    for _, l := range m.datos {
        if strings.Contains(strings.ToLower(l.titulo), strings.ToLower(titulo)) &&
            strings.Contains(strings.ToLower(l.autor), strings.ToLower(autor)) {
            rs := l
            resultado = append(resultado, rs)
        }
    }
    return resultado
}

// Agregar valida y añade un nuevo libro al sistema
func (m *MemoriaLibros) Agregar(libro Libro) error {
    if libro.titulo == "" || libro.autor == "" {
        return errors.New("título y autor son obligatorios")
    }
    if libro.anio < 0 || libro.anio > 2100 {
        return errors.New("año inválido")
    }
    m.ultimoID++
    libro.ID = m.ultimoID
    m.datos = append(m.datos, libro)
    return nil
}

// Actualizar busca un libro por ID y reemplaza sus datos
```

```
func (m *MemoriaLibros) Actualizar(id int, libro Libro) error {
    if libro.titulo == "" || libro.autor == "" {
        return errors.New("título y autor no pueden estar vacíos")
    }
    for i := range m.datos {
        if m.datos[i].ID == id {
            libro.ID = id // mantener el ID original
            m.datos[i] = libro
            return nil
        }
    }
    return errors.New("libro no encontrado")
}

// Eliminar remueve un libro del slice por su ID
func (m *MemoriaLibros) Eliminar(id int) error {
    for i := range m.datos {
        if m.datos[i].ID == id {
            m.datos = append(m.datos[:i], m.datos[i+1:]...)
            return nil
        }
    }
    return errors.New("libro no encontrado")
}

// Función principal que muestra el menú y responde a las acciones del usuario
func main() {
    libros := &MemoriaLibros{} // instancia en memoria
    scanner := bufio.NewScanner(os.Stdin)
```

```
for {
    // Mostrar opciones del menú
    fmt.Println("\n----- GESTIÓN DE LIBROS ELECTRÓNICOS -----")
    fmt.Println("1. Listar libros")
    fmt.Println("2. Buscar libros")
    fmt.Println("3. Registrar nuevo libro")
    fmt.Println("4. Editar libro")
    fmt.Println("5. Eliminar libro")
    fmt.Println("0. Salir")
    fmt.Print("Seleccione una opción: ")
    scanner.Scan()
    opcion := scanner.Text()

    switch opcion {
    case "1": // listar todos
        for _, l := range libros.Listar() {
            fmt.Printf("[%d] %s - %s (%s, %d)\n", l.ID, l.Titulo(), l.Autor(), l.Categoria(), l.Anio())
        }
    case "2": // búsqueda
        fmt.Print("Título: ")
        scanner.Scan()
        t := scanner.Text()
        fmt.Print("Autor: ")
        scanner.Scan()
        a := scanner.Text()
        res := libros.Buscar(t, a)
        if len(res) == 0 {
            fmt.Println("No se encontraron resultados.")
```

```
    } else {
        for _, l := range res {
            fmt.Printf("[%d] %s - %s (%s, %d)\n", l.ID, l.Titulo(), l.Autor(), l.Categoria(), l.Anio())
        }
    }
case "3": // nuevo libro
    libro := leerLibro(scanner)
    if err := libros.Agregar(libro); err != nil {
        fmt.Println("Error:", err)
    } else {
        fmt.Println("Libro registrado correctamente.")
    }
case "4": // editar libro existente
    fmt.Print("ID a editar: ")
    scanner.Scan()
    id, _ := strconv.Atoi(scanner.Text())
    editado := leerLibro(scanner)
    if err := libros.Actualizar(id, editado); err != nil {
        fmt.Println("Error:", err)
    } else {
        fmt.Println("Libro actualizado.")
    }
case "5": // eliminar libro
    fmt.Print("ID a eliminar: ")
    scanner.Scan()
    id, _ := strconv.Atoi(scanner.Text())
    fmt.Print("¿Está seguro? (s/n): ")
    scanner.Scan()
    if strings.ToLower(scanner.Text()) == "s" {
```

```
            if err := libros.Eliminar(id); err != nil {
                fmt.Println("Error:", err)
            } else {
                fmt.Println("Libro eliminado.")
            }
        } else {
            fmt.Println("Operación cancelada.")
        }
    case "0":
        fmt.Println("Saliendo...")
        return
    default:
        fmt.Println("Opción inválida")
    }
  }
}

// leerLibro: solicita los datos de un libro desde consola y los encapsula en un struct
func leerLibro(scanner *bufio.Scanner) Libro {
  fmt.Print("Título: ")
  scanner.Scan()
  t := scanner.Text()
  fmt.Print("Autor: ")
  scanner.Scan()
  a := scanner.Text()
  fmt.Print("Categoría: ")
  scanner.Scan()
  c := scanner.Text()
  fmt.Print("Año: ")
```

```
    scanner.Scan()
    año, _ := strconv.Atoi(scanner.Text())

    // Usar setters para encapsulación
    libro := Libro{}
    libro.SetTitulo(t)
    libro.SetAutor(a)
    libro.SetCategoria(c)
    libro.SetAnio(año)

    return libro
}
```