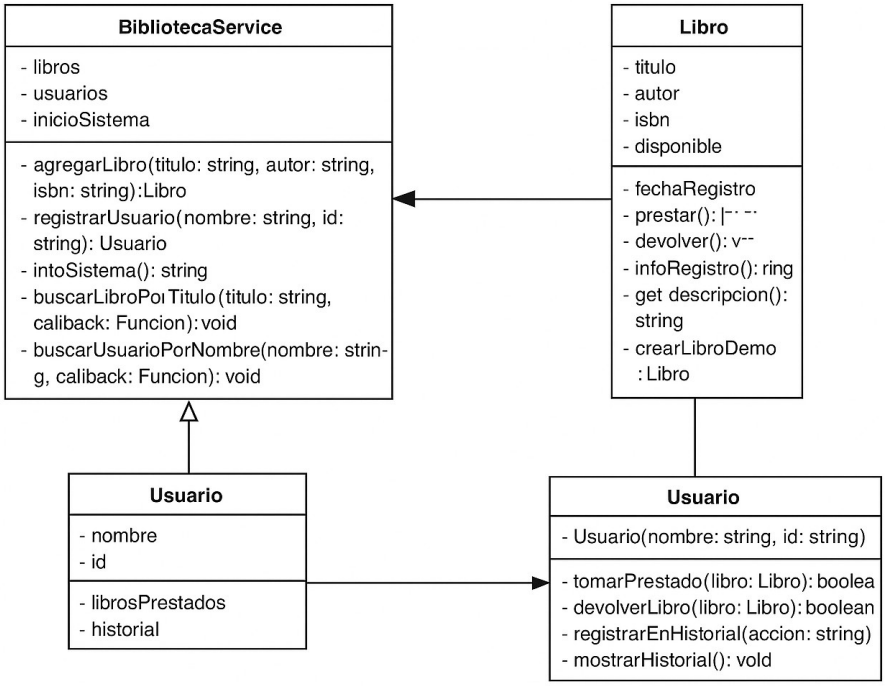
Práctica 9: Clases, Módulos y Funciones (Tradicionales vs Flecha) en JavaScript



1. **Crea el proyecto y configura package.json:**

mkdir biblioteca-js
cd biblioteca-js
npm init -y

2. Edita package.json para usar módulos ES: Establecer las bases del proyecto Node.js con soporte para módulos ES6.

- **"type": "module"**: Habilita la sintaxis de importación/exportación ES6
- Scripts preconfigurados para ejecución **(start)** y pruebas **(test)**
- Permite usar **import/export** en lugar de **require/module.exports**

```
"type": "module",
  "scripts": {
    "start": "node src/app.js",
    "test": "node src/test.js"
  }
```

3. Estructura del proyecto: Descarga el archivo adjunto y crea la estructura mostrada del proyecto

```
biblioteca-js/
├── package.json
├── src/
│   ├── entidades/
│   │   ├── Libro.js
│   │   └── Usuario.js
│   ├── servicios/
│   │   └── BibliotecaService.js
│   ├── helpers/
│   │   └── Funciones.js
│   └── app.js
```

4. Funciones.js

Función	Tipo	Qué hace	Comentarios
`formatearFechaTradicional`	Tradicional	Formatea fecha en español	Tiene su `this` propio (aunque no se usa)
`formatearFechaFlecha`	Flecha	Hace lo mismo que la tradicional	Sintaxis más compacta, sin `this`
`mostrarInfoTradicional`	Tradicional	Devuelve fecha/hora actual	Puede usar `this` si se requiere
`mostrarInfoFlecha`	Flecha	Solo llama a la tradicional	Arrow function como wrapper

5. **Clase Libro (Libro.js)**

Elemento	¿Qué hace?	Observaciones
`constructor`	Inicializa un libro con datos y fecha	Correcto
`prestar()`	Método normal, cambia disponibilidad	Buena práctica
`devolver` (flecha)	Método flecha como propiedad	No recomendado, pero funciona
`infoRegistro()`	Usa helper importado para fecha	Correcto
`descripcion`	Getter para mostrar título y autor	Correcto
`crearLibroDemo` (static flecha)	Crea libro de ejemplo sin instanciar antes	Funciona

6. **Clase Usuario (Usuario.js)**

Parte	Función	Tipo	Comentario
`constructor`	Crear usuario, listas vacías	—	Correcto
`tomarPrestado()`	Prestar libro y registrar acción	Método tradicional	Muy adecuado
`devolverLibro`	Devolver libro usando arrow function	Método flecha	Funciona, pero no ideal por memoria
`registrarEnHistorial()`	Guardar acciones	Tradicional	Correcto
`mostrarHistorial()`	Mostrar historial en consola	Tradicional con flecha interna	Correcto

7. BibliotecaService (BibliotecaService.js)

Función	Tipo	Qué hace	Comentario
`constructor`	—	Inicializa listas y fecha de inicio	Correcto
`agregarLibro()`	Método tradicional	Crea y agrega un libro	Buena práctica
`registrarUsuario`	Arrow function	Crea y agrega un usuario	Funciona, aunque no óptimo
`infoSistema()`	Método tradicional	Muestra fecha formateada	Usa helper externo
`buscarLibroPorTitulo()`	Callback tradicional	Busca libros por título	Ejemplo de callback clásico
`buscarUsuarioPorNombre()`	Callback arrow	Busca usuarios por nombre	Arrow para sintaxis corta

8. **Archivo Principal (app.js)**

Propósito: Demostrar el uso integrado de todos los componentes.

Flujo:

1. Se muestran datos del sistema y fecha actual.
2. Se crea la biblioteca.
3. Se agregan un libro y un usuario.
4. Se agrega un libro demo mediante un método estático.
5. Se buscan libros y usuarios con dos tipos de callbacks.
6. El usuario presta un libro y luego lo devuelve.
7. Se imprime su historial de acciones.

9. **Ejecución del proyecto**

npm start *# Ejecuta app.js*
node src/test.js *# Ejecuta las pruebas*

Ejercicio para entrega:

Ampliar el sistema de biblioteca implementando un módulo de préstamos y multas, utilizando funciones tradicionales y arrow functions en los contextos adecuados. Sube un reporte con el funcionamiento y adjunta tu enlace del codigo en github

Parte 1: Modificar la Clase Libro

1.Añade un método flecha:

- **diasPrestamo()**: Calcula los días base de préstamo (15 días si está prestado, 0 si está disponible).

2.Añade un método tradicional:

- **estaDisponible()**: Devuelve **true** o **false** según la disponibilidad del libro.

3.Añade un método estático con arrow function:

- **validarISBN(isbn)**: Valida que el ISBN tenga exactamente 6 dígitos numéricos (usa una expresión regular).

Parte 2: Crear la Clase Prestamo

1.Propiedades:

2.**libro**, **usuario**, **fechaPrestamo**, **fechaDevolucion**, **multa**.

3.Métodos:

- Tradicional:
  - **registrarDevolucion()**: Establece **fechaDevolucion**, calcula la multa y marca el libro como disponible.
- Arrow function:
  - **calcularMulta()**: Calcula multa de $2 por día de retraso (solo si hay devolución).
- Función tradicional privada (convención **_nombre**):
  - **_diasTranscurridos()**: Calcula días entre préstamo y devolución.
- Método con arrow function interna:

- **infoPrestamo()**: Retorna un string formateado con los detalles del préstamo, usando una arrow function interna para mostrar el estado.

Parte 3: Ampliar BibliotecaService

1.Nuevas propiedades:

- Añade un array **prestamos** al constructor.

2.Métodos:

- Arrow function:
  - **registrarPrestamo(libroId, usuarioId)**: Registra un nuevo préstamo si el libro y usuario existen.
- Tradicional con callback:
  - **buscarPrestamosPorUsuario(usuarioId, callback)**: Filtra préstamos por ID de usuario y usa un callback tradicional para procesar resultados.
- Arrow function con reduce:
  - **calcularMultasPendientes()**: Suma todas las multas no pagadas.

Parte 4: Implementar en app.js

1.Simula un préstamo:

- Usa **registrarPrestamo** (arrow function) para prestar un libro.
- Muestra los detalles con **infoPrestamo()**.

2.Simula una devolución después de "20 días":

- Usa **setTimeout** para simular el paso del tiempo.
- Llama a **registrarDevolucion()** (tradicional) y muestra el nuevo estado.

3.Busca préstamos por usuario:

- Usa **buscarPrestamosPorUsuario** con un callback tradicional para mostrar resultados.

4.Calcula multas totales:

- Usa **calcularMultasPendientes()** (arrow function) para mostrar el total.