Laboratorio

Ingeniería de Software

@SOYPABLOG – 21 DE FEBRERO DE 2023

Anteriormente...

Recursión de cola

`clojure.core`

Seqs

Lazy seqs

Diseño

Requerimientos

«Un requerimiento de *software* es una característica que se debe exhibir por el *software* desarrollado para resolver un problema del mundo real».

SWEBOK, 2014

Definición de requerimientos

- Los requerimientos de un producto de software son las descripciones de lo que hará, los servicios que proporcionará y las restricciones de su operación.
  Sommerville I., 2011

- Un requerimiento o necesidad es lo que el cliente o un usuario desean que haga el software para resolver un problema.
  Ibargüengoitia G; *Oktaba, H.*

Sistema Único de Gestión Bibliotecaria (SUGBI)

- Existen dos tipos de usuarios de la biblioteca, los miembros y los bibliotecarios.
- Los usuarios pueden iniciar sesión en el sistema utilizando su correo electrónico y una contraseña.
- Los miembros pueden llevarse libros en préstamo.
- Los miembros y los bibliotecarios pueden buscar libros por título o por autor.
- Los bibliotecarios pueden bloquear y desbloquear miembros (por ejemplo, cuando se retrasan en la devolución de un libro).
- Los bibliotecarios pueden visualizar los libros que actualmente se han prestado a un miembro.
- Existen varios ejemplares de un libro.
- Cada libro pertenece a una biblioteca física.

Sistema Único de Gestión Bibliotecaria (SUGBI)

Requerimientos

- Existen dos tipos de usuarios de la biblioteca, los miembros y los bibliotecarios.
- Los usuarios pueden iniciar sesión en el sistema utilizando su correo electrónico y una contraseña.
- Los miembros pueden llevarse libros en préstamo.
- Los miembros y los bibliotecarios pueden buscar libros por título o por autor.
- Los bibliotecarios pueden bloquear y desbloquear miembros (por ejemplo, cuando se retrasan en la devolución de un libro).
- Los bibliotecarios pueden visualizar los libros que actualmente se han prestado a un miembro.
- Existen varios ejemplares de un libro.
- Cada libro pertenece a una biblioteca física.

Principales clases necesarias

- `Library`.
- `Book`.
- `BookItem`.
- `BookLending`.
- `Member`.
- `Librarian`.
- `User`.
- `Catalog`.
- `Author`.

UML

Unified Modeling Language

- El lenguaje de modelado unificado (UML) es un lenguaje de modelado visual de propósito general que se utiliza para especificar, visualizar, construir y documentar los artefactos que posee un sistema de software.

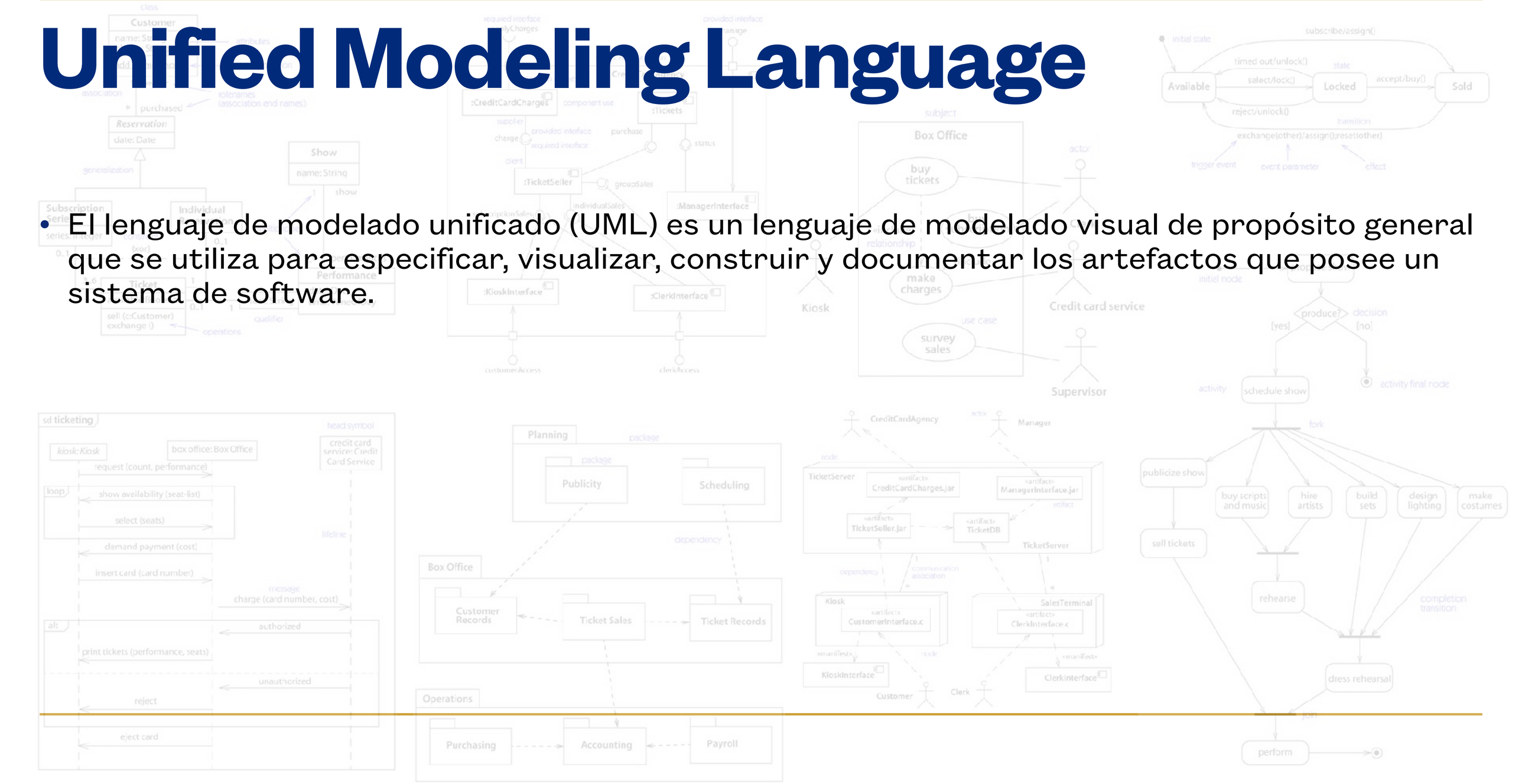
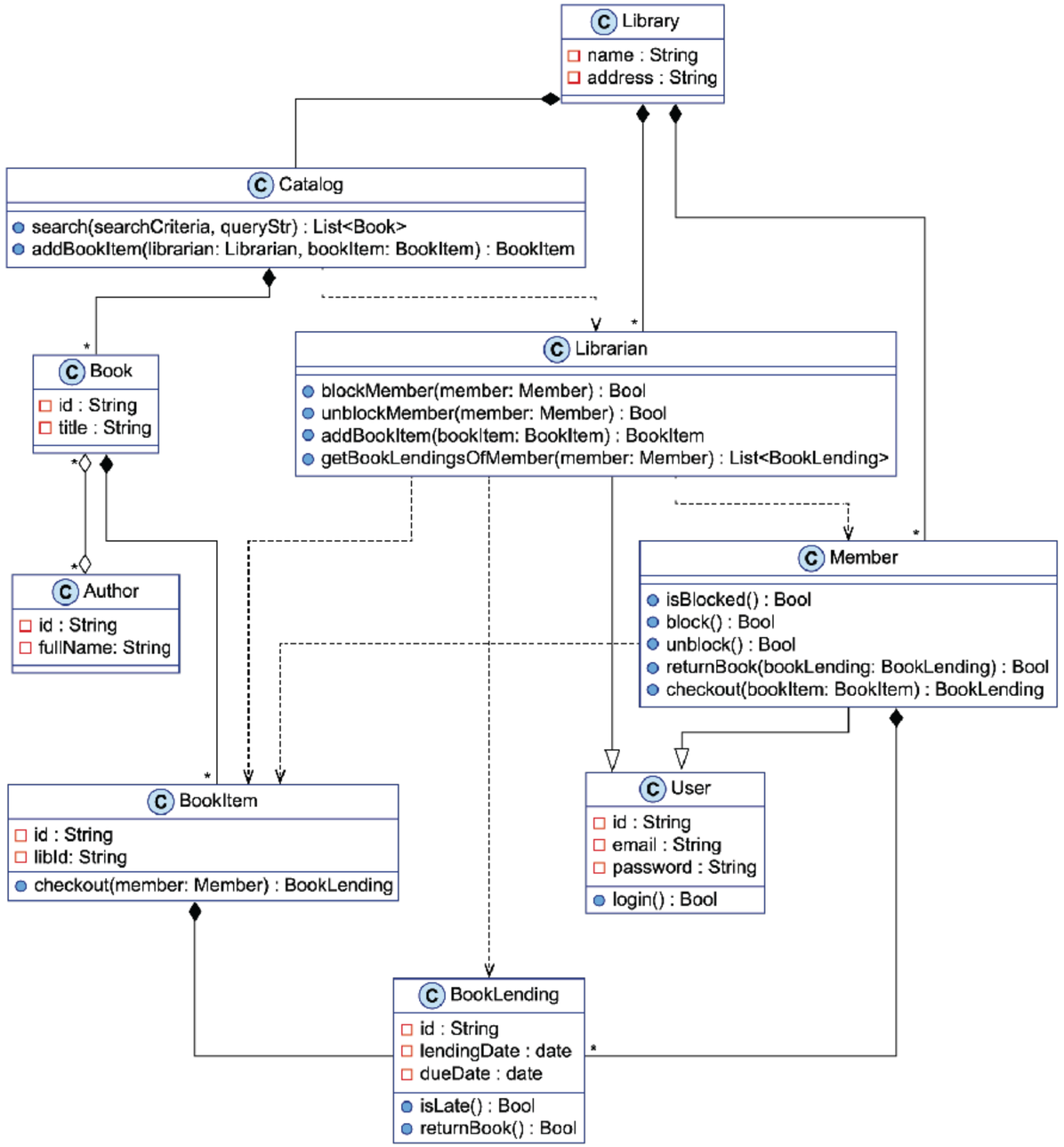


Diagrama de clases

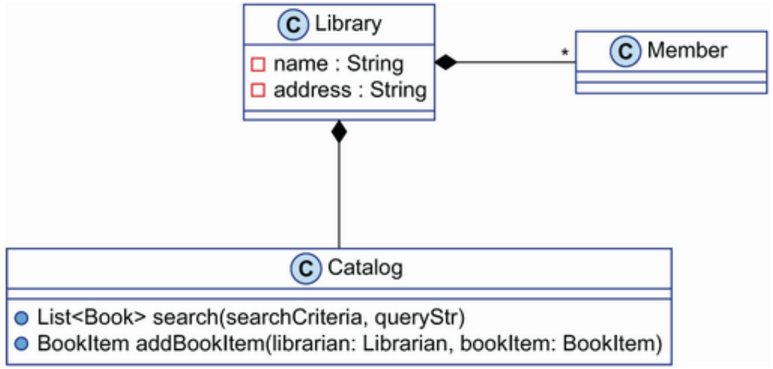

Library
name : String
address : String
Catalog
search(searchCriteria, queryStr) : List<Book>
addBookItem(librarian: Librarian, bookItem: BookItem) : BookItem
Book
id : String
title : String
Librarian
blockMember(member: Member) : Bool
unblockMember(member: Member) : Bool
addBookItem(bookItem: BookItem) : BookItem
getBookLendingsOfMember(member: Member) : List<BookLending>
Author
id : String
fullName: String
Member
isBlocked() : Bool
block() : Bool
unblock() : Bool
returnBook(bookLending: BookLending) : Bool
checkout(bookItem: BookItem) : BookLending
BookItem
id : String
libId: String
checkout(member: Member) : BookLending
User
id : String
email : String
password : String
login() : Bool
BookLending
id : String
lendingDate : date
dueDate : date
isLate() : Bool
returnBook() : Bool

Tipos de relaciones

- Composición.
- Asociación.
- Herencia.
- Dependencia.

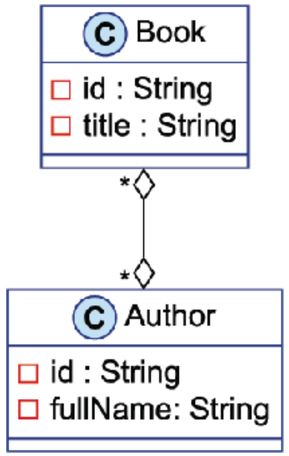
Tipos de relaciones

- Composición. Sucede cuando un objeto no puede existir sin algún otro. Cuando un objeto muere, el otro lo hace también.



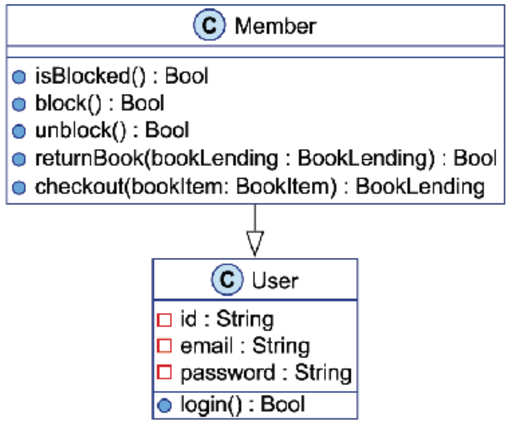
Tipos de relaciones

- Asociación. Se da cuando algún objeto depende de otro. Sin embargo, pueden existir de forma independiente.



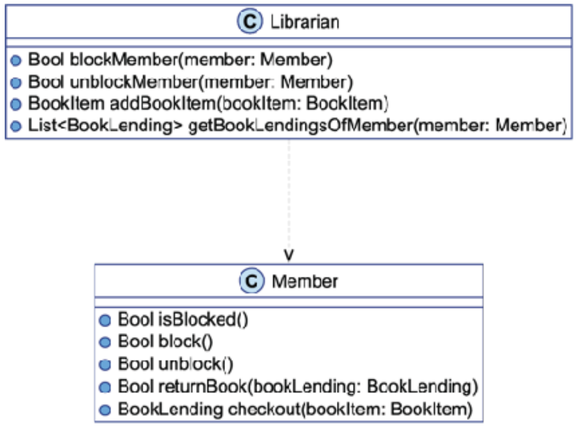
Tipos de relaciones

- Herencia. Muestra la relación entre objetos generales y sus especializaciones, mejor conocidos como super clases y sub clases.



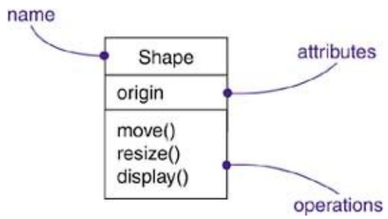
Tipos de relaciones

- Dependencia. Muestra la relación de uso entre objetos.

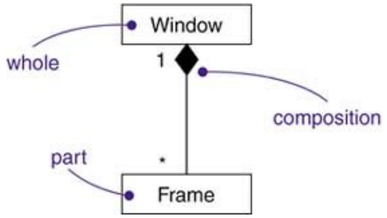


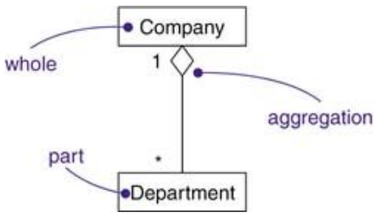
UML *Cheatsheet*

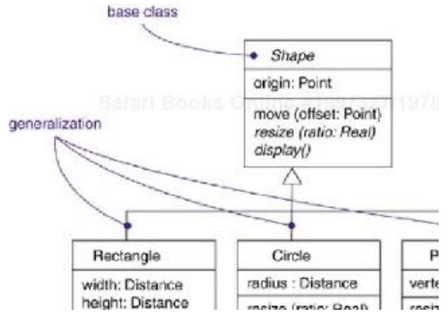
Clases

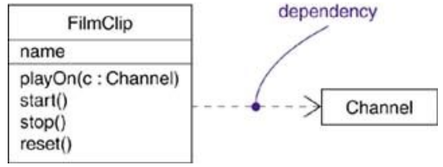


Relaciones

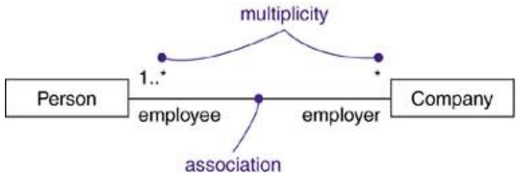


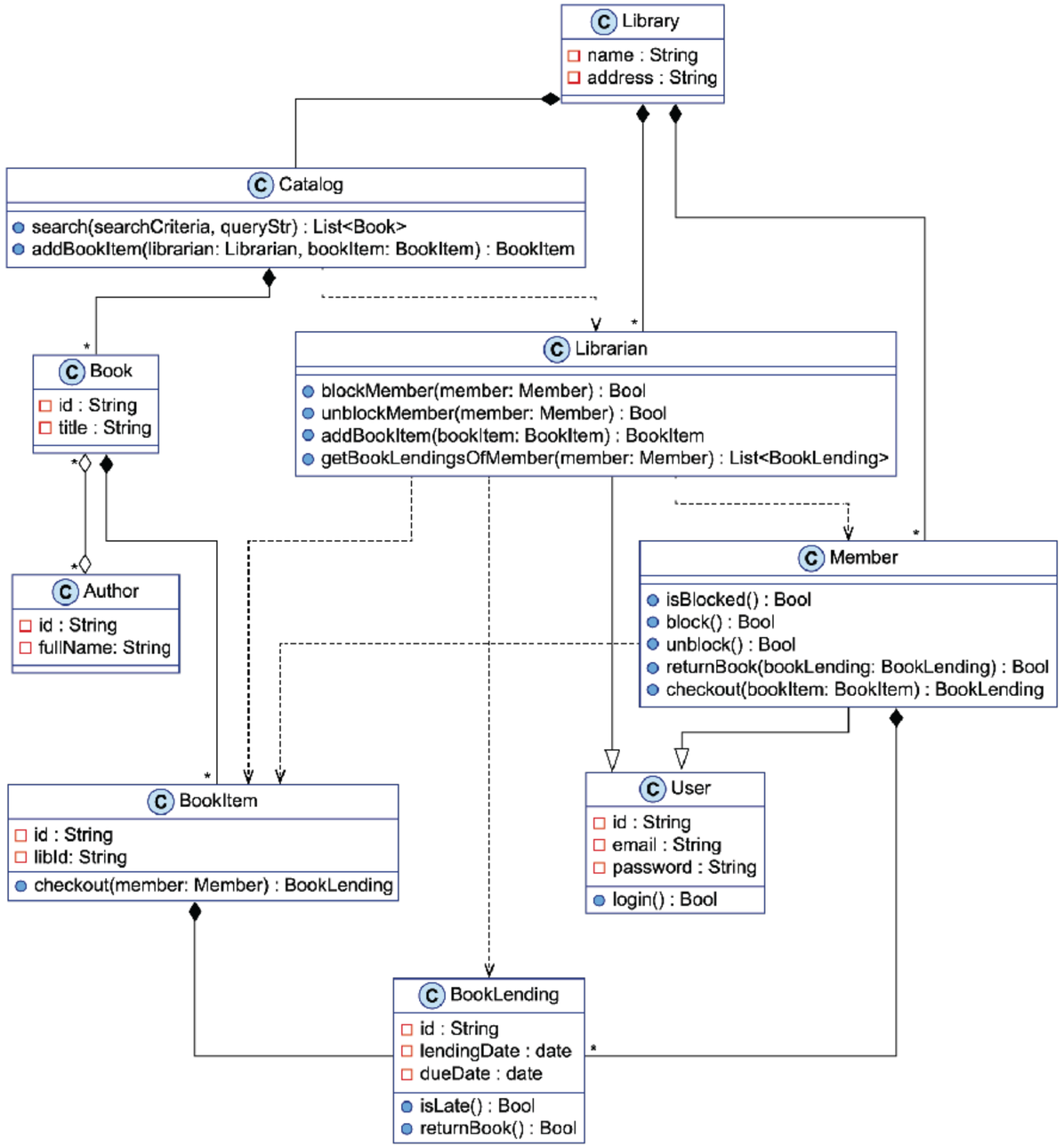






Multiplicidad



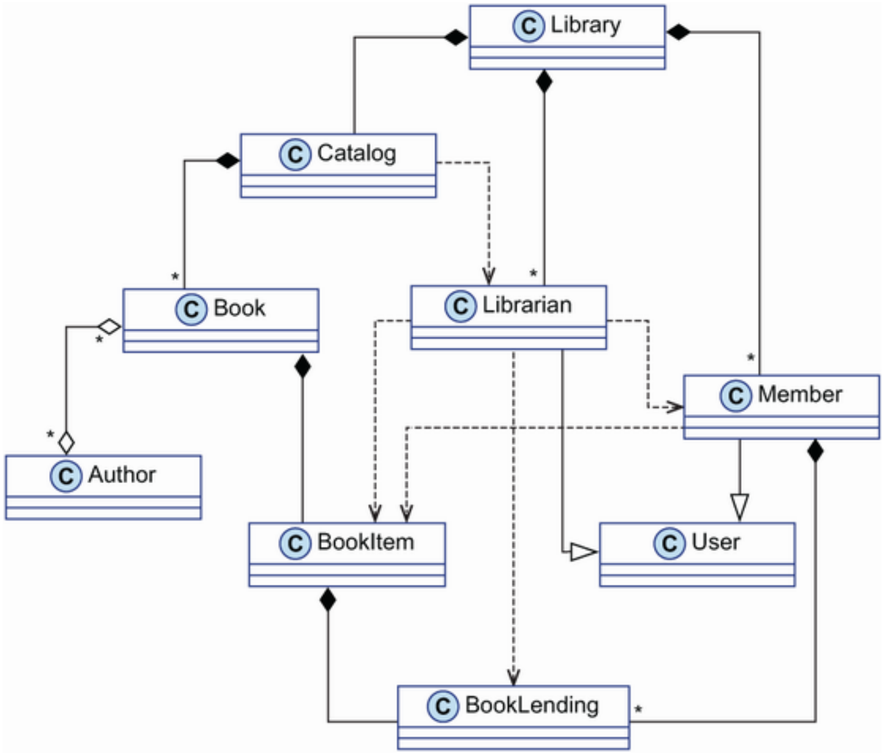

Library
name : String
address : String
Catalog
search(searchCriteria, queryStr) : List<Book>
addBookItem(librarian: Librarian, bookItem: BookItem) : BookItem
Book
id : String
title : String
Librarian
blockMember(member: Member) : Bool
unblockMember(member: Member) : Bool
addBookItem(bookItem: BookItem) : BookItem
getBookLendingsOfMember(member: Member) : List<BookLending>
Author
id : String
fullName: String
Member
isBlocked() : Bool
block() : Bool
unblock() : Bool
returnBook(bookLending: BookLending) : Bool
checkout(bookItem: BookItem) : BookLending
BookItem
id : String
libId: String
checkout(member: Member) : BookLending
User
id : String
email : String
password : String
login() : Bool
BookLending
id : String
lendingDate : date
dueDate : date
isLate() : Bool
returnBook() : Bool

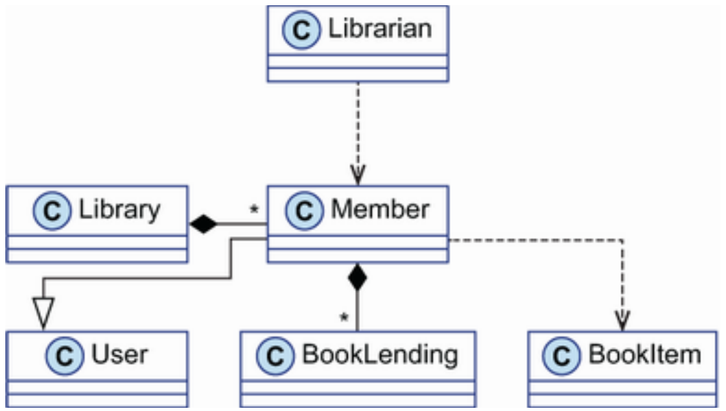
¿Por qué los sistemas en orientación a objetos son tan complejos?

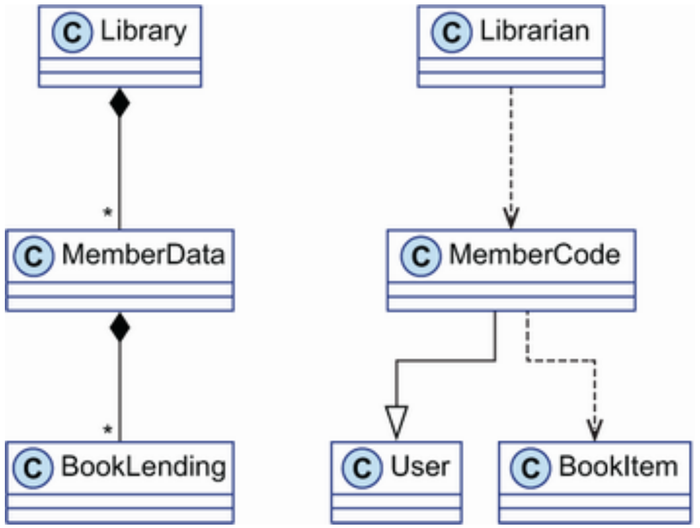
- Se mezcla la información y el comportamiento.
- Los objetos son mutables.
- La información está encapsulada dentro de los objetos (atributos).
- El comportamiento está encapsulado dentro de las clases (métodos).

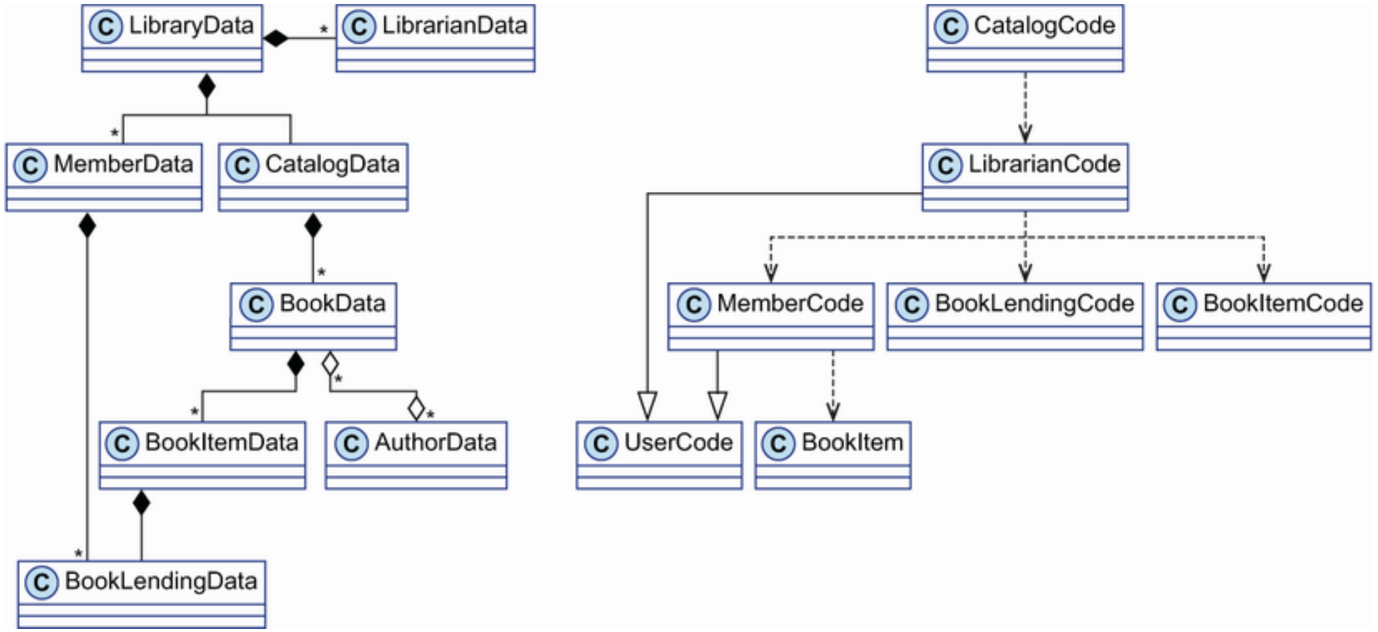
¿Por qué los sistemas en orientación a objetos son tan complejos?

- Se mezcla la información y el comportamiento.
- Los objetos son mutables.
- La información está encapsulada dentro de los objetos (atributos).
- El comportamiento está encapsulado dentro de las clases (métodos).


Library
Catalog
Book
Librarian
Member
Author
BookItem
User
BookLending
*


Librarian
Library
Member
User
BookLending
BookItem
*
*


Library
Librarian
MemberData
MemberCode
BookLending
User
BookItem
*
*


LibraryData
LibrarianData
MemberData
CatalogData
BookData
BookItemData
AuthorData
BookLendingData
CatalogCode
LibrarianCode
MemberCode
BookLendingCode
BookItemCode
UserCode
BookItem

¿Por qué los sistemas en orientación a objetos son tan complejos?

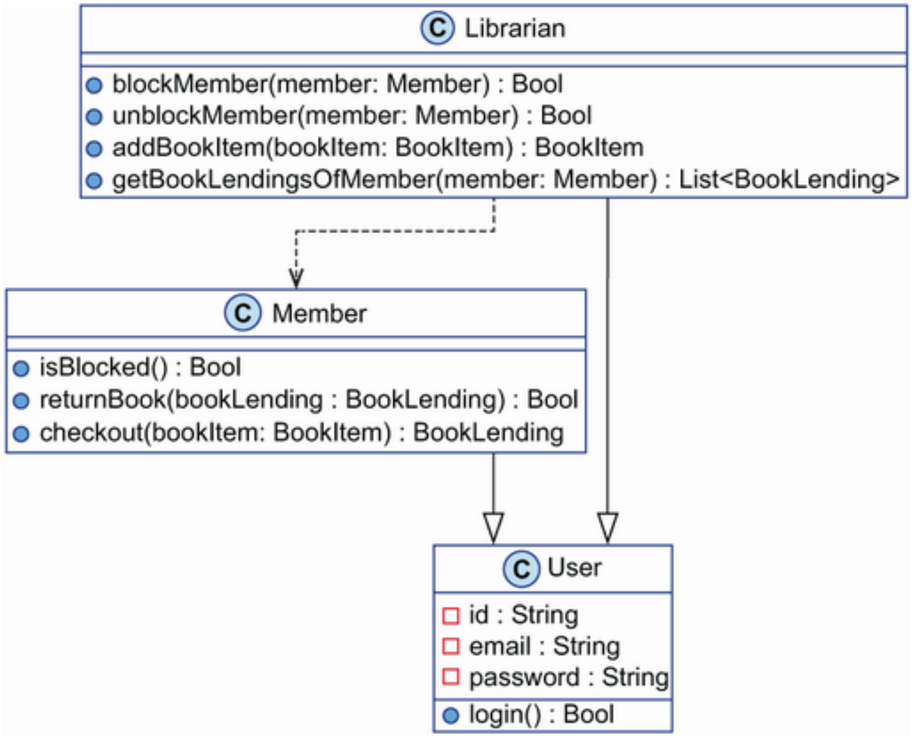
- Se mezcla la información y el comportamiento.
- Los objetos son mutables.
- La información está encapsulada dentro de los objetos (atributos).
- El comportamiento está encapsulado dentro de las clases (métodos).

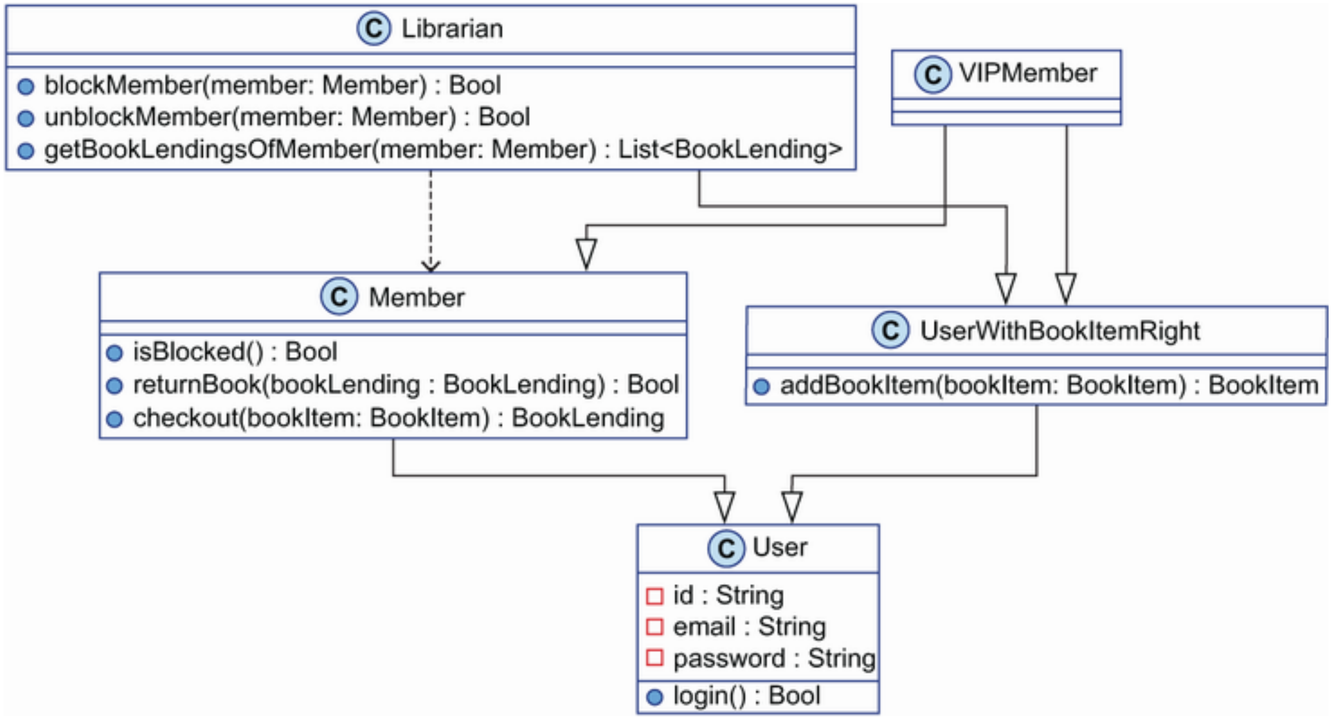
```
class Member {
    isBlocked;

    displayBlockedStatusTwice() {
        var isBlocked = this.isBlocked;
        console.log(isBlocked);
        console.log(isBlocked);
    }
}

member.displayBlockedStatusTwice();
```

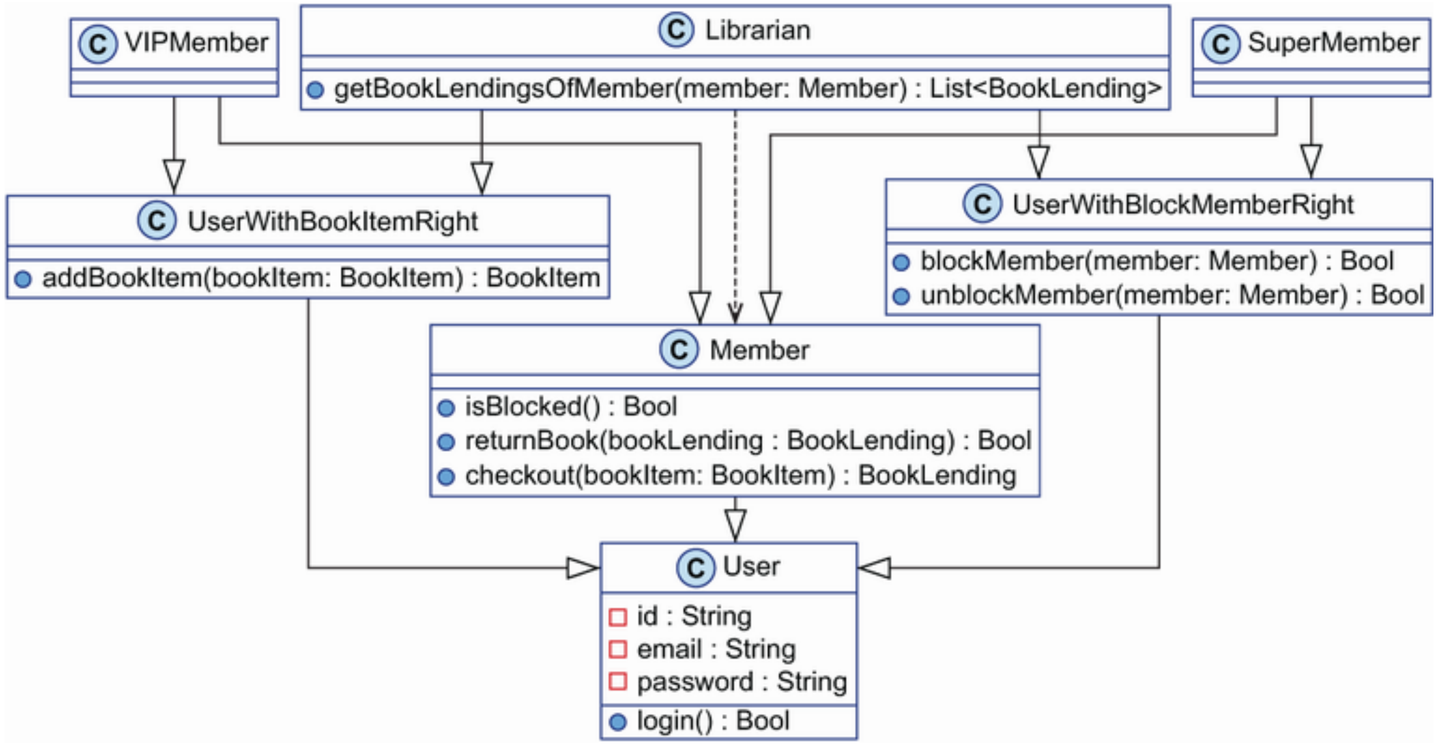
```
class Member {
    isBlocked;

    displayBlockedStatusTwice() {
        console.log(isBlocked);
        console.log(isBlocked);
    }
}

member.displayBlockedStatusTwice();
```

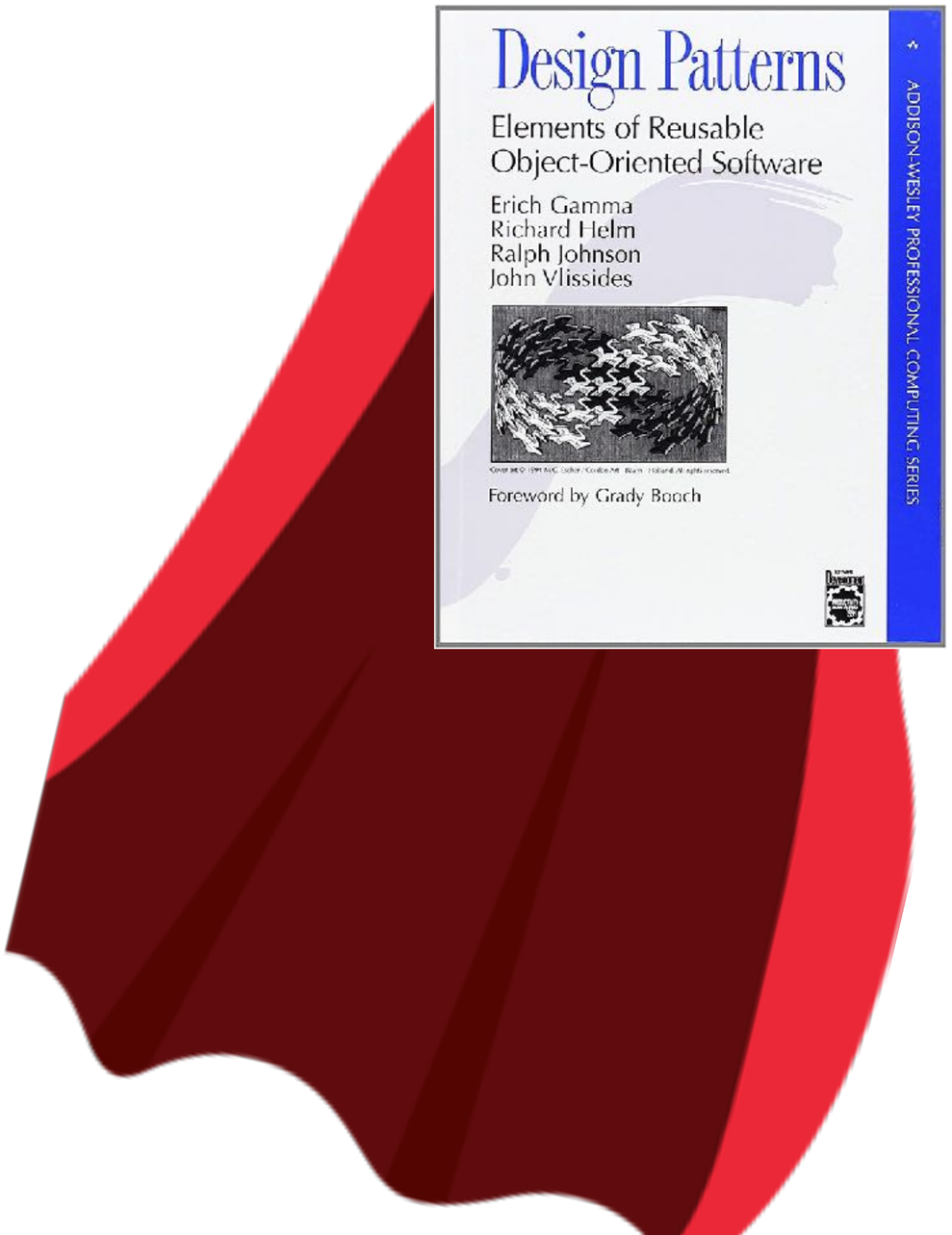
¿Por qué los sistemas en orientación a objetos son tan complejos?

- Se mezcla la información y el comportamiento.
- Los objetos son mutables.
- La información está encapsulada dentro de los objetos (atributos).
- El comportamiento está encapsulado dentro de las clases (métodos).

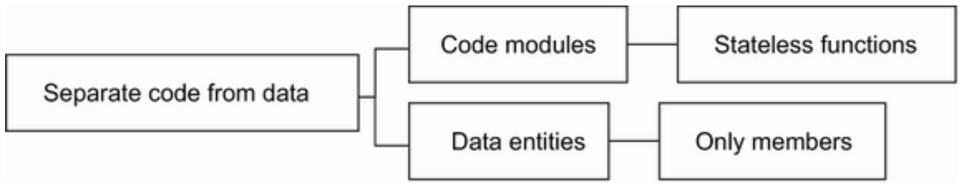

Librarian
blockMember(member: Member) : Bool
unblockMember(member: Member) : Bool
addBookItem(bookItem: BookItem) : BookItem
getBookLendingsOfMember(member: Member) : List<BookLending>
Member
isBlocked() : Bool
returnBook(bookLending : BookLending) : Bool
checkout(bookItem: BookItem) : BookLending
User
id : String
email : String
password : String
login() : Bool


Librarian
blockMember(member: Member) : Bool
unblockMember(member: Member) : Bool
getBookLendingsOfMember(member: Member) : List<BookLending>
VIPMember
Member
isBlocked() : Bool
returnBook(bookLending : BookLending) : Bool
checkout(bookItem: BookItem) : BookLending
UserWithBookItemRight
addBookItem(bookItem: BookItem) : BookItem
User
id : String
email : String
password : String
login() : Bool


VIPMember
Librarian
getBookLendingsOfMember(member: Member) : List<BookLending>
SuperMember
UserWithBookItemRight
addBookItem(bookItem: BookItem) : BookItem
UserWithBlockMemberRight
blockMember(member: Member) : Bool
unblockMember(member: Member) : Bool
Member
isBlocked() : Bool
returnBook(bookLending : BookLending) : Bool
checkout(bookItem: BookItem) : BookLending
User
id : String
email : String
password : String
login() : Bool


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Programación Orientada a Datos


Separate code from data
Code modules
Stateless functions
Data entities
Only members

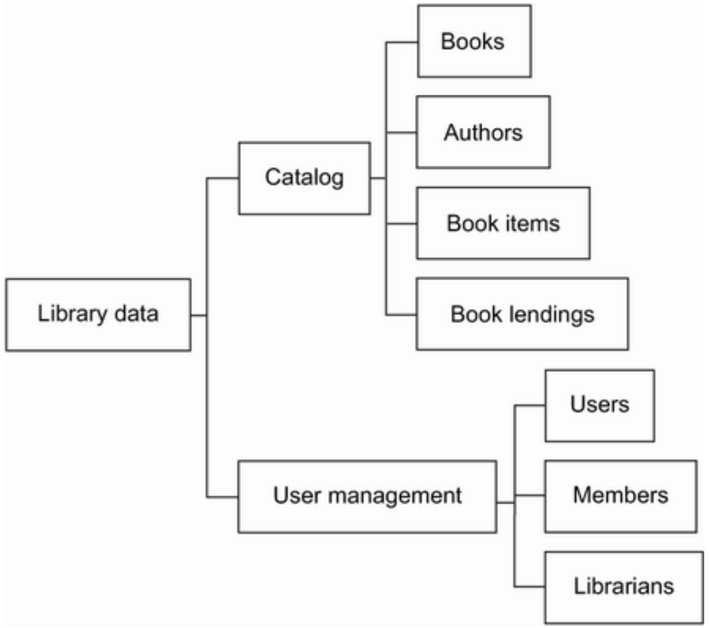
Sistema Único de Gestión Bibliotecaria (SUGBI)

Requerimientos

- Existen dos tipos de usuarios de la biblioteca, los *miembros* y los *bibliotecarios*.
- Los *usuarios* pueden iniciar sesión en el sistema utilizando su correo electrónico y una contraseña.
- Los *miembros* pueden llevarse *libros* en préstamo.
- Los *miembros* y los *bibliotecarios* pueden buscar *libros* por *título* o por *autor*.
- Los *bibliotecarios* pueden bloquear y desbloquear *miembros* (por ejemplo, cuando se retrasan en la devolución de un libro).
- Los *bibliotecarios* pueden visualizar los *libros que actualmente se han prestado* a un miembro.
- Existen varios *ejemplares de un libro*.

Entidades de datos necesarias

- `Catálogo`
  - Información de `libros`
  - Información sobre `autores`
  - Información de `ejemplares`
  - Información de `préstamos`
- `Administración de usuarios`
  - Información sobre `usuarios`
  - Información sobre `miembros`
  - Información sobre `bibliotecarios`


Books
Authors
Catalog
Book items
Library data
Book lendings
Users
User management
Members
Librarians

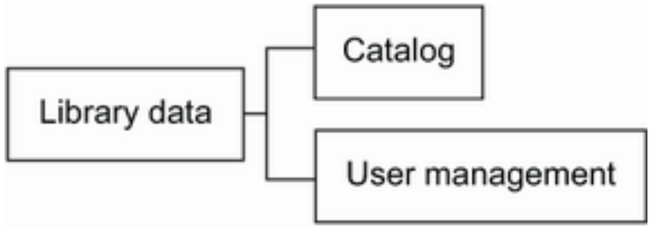
Sistema Único de Gestión Bibliotecaria (SUGBI)

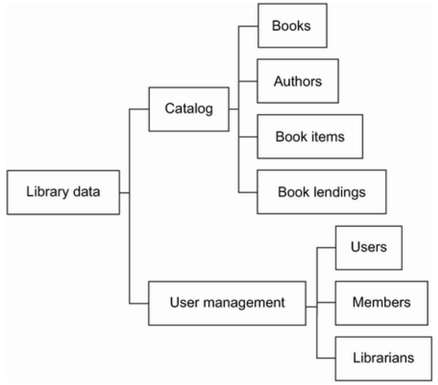
Requerimientos

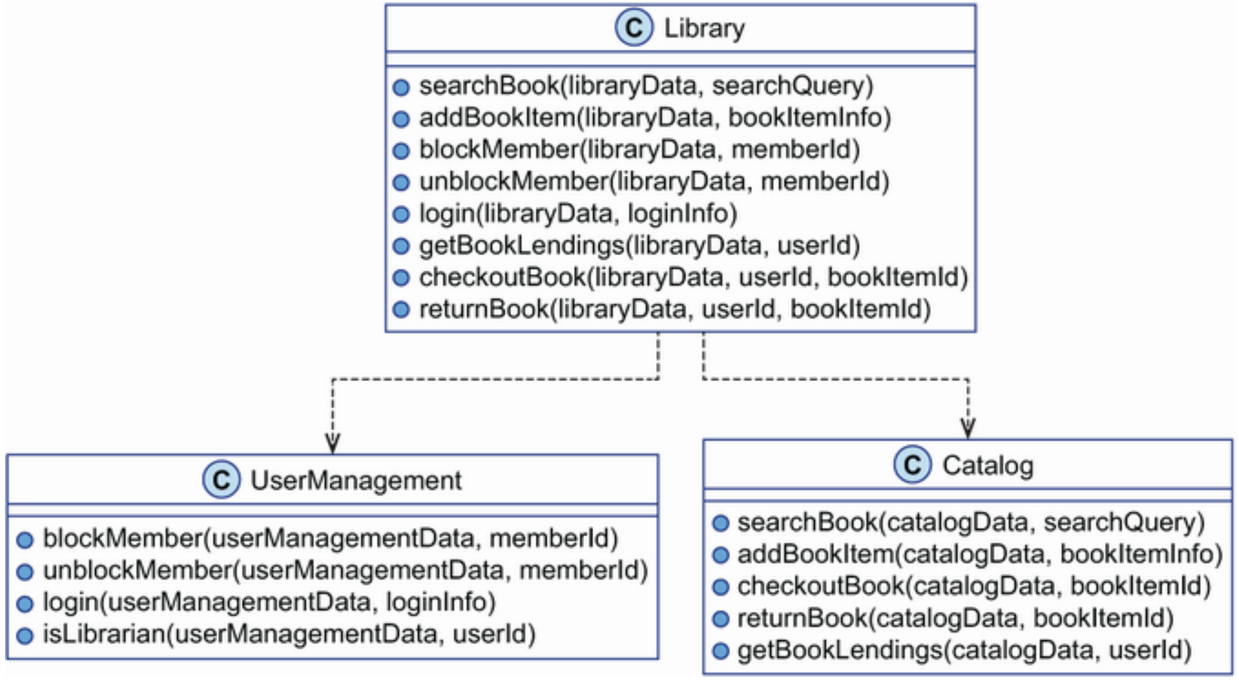
- Existen dos tipos de usuarios de la biblioteca, los miembros y los bibliotecarios.
- Los usuarios pueden *iniciar sesión en el sistema* utilizando su correo electrónico y una contraseña.
- Los miembros pueden *llevarse libros* en préstamo.
- Los miembros y los bibliotecarios pueden *buscar libros* por título o por autor.
- Los bibliotecarios pueden *bloquear y desbloquear miembros* (por ejemplo, cuando se retrasan en la devolución de un libro).
- Los bibliotecarios pueden *visualizar los libros que actualmente se han prestado a un miembro*.
- Existen varios ejemplares de un libro.

Funcionalidad necesaria

- Buscar libros.
- Bloquear a un miembro.
- Desbloquear a un miembro.
- Iniciar sesión en el sistema.
- Visualizar los libros que actualmente se han prestado a un miembro.
- Prestar un libro.
- Regresar un libro.
- Añadir un ejemplar.
- Determinar si un usuario es un bibliotecario.


Library data
Catalog
User management


Library data
Catalog
Books
Authors
Book items
Book lendings
User management
Users
Members
Librarians


Library
searchBook(libraryData, searchQuery)
addBookItem(libraryData, bookItemInfo)
blockMember(libraryData, memberId)
unblockMember(libraryData, memberId)
login(libraryData, loginInfo)
getBookLendings(libraryData, userId)
checkoutBook(libraryData, userId, bookItemId)
returnBook(libraryData, userId, bookItemId)
UserManagement
blockMember(userManagementData, memberId)
unblockMember(userManagementData, memberId)
login(userManagementData, loginInfo)
isLibrarian(userManagementData, userId)
Catalog
searchBook(catalogData, searchQuery)
addBookItem(catalogData, bookItemInfo)
checkoutBook(catalogData, bookItemId)
returnBook(catalogData, bookItemId)
getBookLendings(catalogData, userId)

Desarrollo web
en Clojure.

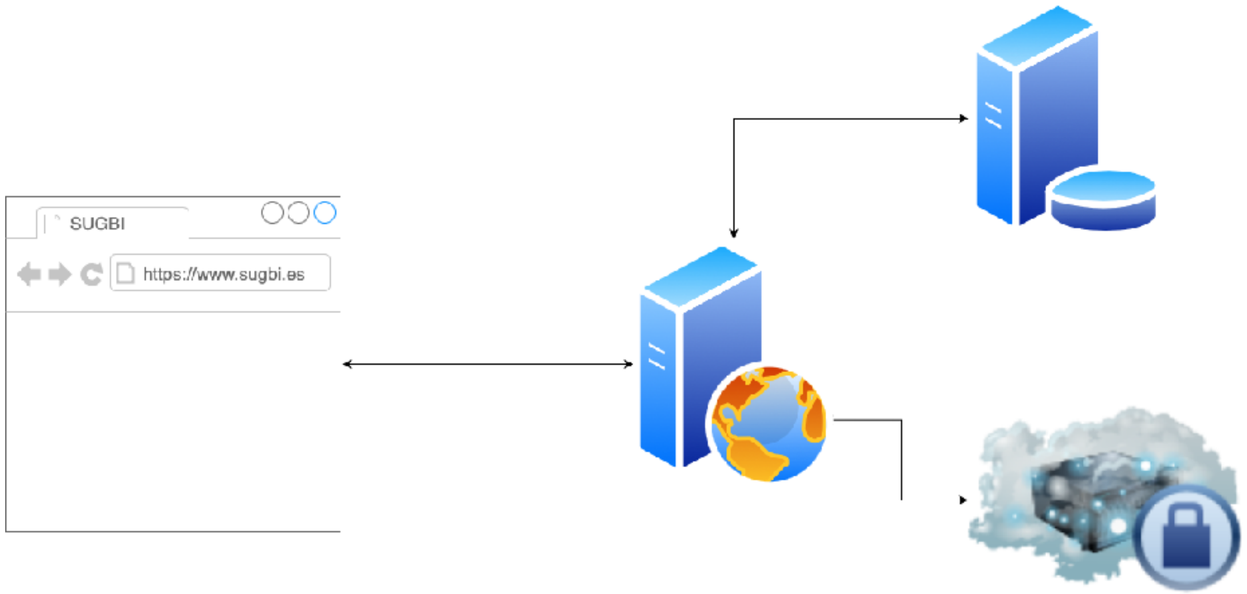
Luminus

¿Cómo crear un proyecto web?

- Ejecutar la siguiente instrucción:

  ```
  lein new luminus <nombre-proyecto> +jetty +postgres +swagger +cljs +re-frame +shadow-cljs +auth
  ```

- `+jetty` — Servidor Web.
- `+postgres` — Agrega la configuración para conectarse a una base de datos de PostgreSQL.
- `+swagger` — Añade soporte para Swagger UI.
- `+cljs` — Añade y configura ClojureScript al proyecto.
- `+re-frame` — Añade un pequeño *framework* para el desarrollo del *front end*.
- `+shadow-cljs` — Añade un compilador dinámico de ClojureScript.
- `+auth` — Añade una biblioteca para facilitar la autenticación.


SUGBI
https://www.sugbi.es

Base de datos

PostgreSQL

Utilizando `docker-compose`

```
version: '3.1'

services:
  db:
    image: postgres:14.0
    command: postgres
             -c log_statement=all
             -c timezone='America/Mexico_City'
    ports:
      - 5432:5432
    volumes:
      - ./postgres-init-scripts:/docker-entrypoint-initdb.d
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: ppassword
```

Scripts de inicialización de la base de datos

```
create user sugbi with encrypted password 'spassword';

create database sugbi_dev;

grant all privileges on database sugbi_dev to sugbi;
```

Configuración de la base de datos

- `:database-url`

  ```
  "jdbc:postgresql://localhost:6543/sugbi_dev?
  user=sugbi&password=spassword"
  ```

Ahora... ¿cómo se ejecuta?

1. Instalar las dependencias de npm (`npm i`).
2. Levantar el entorno de desarrollo (`npx shadow-cljs watch app`).
3. Desde el REPL de Clojure, iniciar el servidor web (`(user/start)`).

Migraciones

`Migratus` *Cheatsheet*

- `(user/create-migration name)`
  Crea un nuevo *timestamp* con una migración.
- `(user/reset-db)`
  Restaura la base de datos (Ejecuta las migraciones *down*, seguido de las migraciones *up*).
- `(user/migrate)`
  Aplica las migraciones nuevas.
- `(user/rollback)`
  Restaura la base de datos a la última migración.

📚 Les devoirs

- Leer *¿Quién diría que un '1 = 1' podría ser tan dañino?* (https://computo.fciencias.unam.mx/publicaciones/SQLInjection-PabloGL2019.pdf).
- Ver *What Haskell Taught Us When We Weren't Looking* (https://www.youtube.com/watch?v=Pmhap3acJvs).