OOP

PRESENTATION

Libary management systems

ฟีเจอร์ที่มีอยู่ในโค้ด

1. การจัดการหนังสือ

- เก็บข้อมูลของหนังสือ (isbn, title, author)
- ตรวจสอบสถานะหนังสือ (checkAvailability())
- ยืมหนังสือ (เปลี่ยน isAvailable = false)
- คืนหนังสือ (เปลี่ยน isAvailable = true)
- มี getter (getTitle, getISBN, getAuthor)

เงื่อนไขที่เข้า

1..Implements Interface
Book ต้องมี method ครบตามที่ IBorrowable กำหนดไว้
2. Encapsulation
ใช้ private กับ property (isbn, title, author, isAvailable)
ป้องกันไม่ให้แก้ค่าตรง ๆ จากภายนอก
3.State Control (การควบคุมสถานะ)
isAvailable = true เมื่อสร้าง
markAsBorrowed() เปลี่ยนสถานะให้ไม่ว่าง
markAsReturned() เปลี่ยนสถานะให้กลับมาเป็นว่างอีกครั้ง
4.Abstraction
ภายนอกไม่จำเป็นต้องรู้ isAvailable ทำงานยังไงใช้ method checkAvailability() แทนการเช็ค property โดยตรง

```
class Book implements IBorrowable {
  private isbn: string;
  private title: string;
  private author: string;
  private isAvailable: boolean;

  constructor(isbn: string, title: string, author: string) {
    this.isbn = isbn;
    this.title = title;
    this.author = author;
    this.isAvailable = true;
  }

  public checkAvailability(): boolean {
    return this.isAvailable;
  }

  public markAsBorrowed(): void {
    this.isAvailable = false;
  }

  public markAsReturned(): void {
    this.isAvailable = true;
  }

  public getTitle(): string {
    return this.title;
  }

  public getISBN(): string {
    return this.isbn;
  }

  public getAuthor(): string {
    return this.author;
  }
}
```

2. การจัดการผู้ใช้

- เก็บข้อมูลผู้ใช้ (id, name)
- เก็บหนังสือที่ยืม (borrowedBooks[])
- ฟังก์ชันยืม (borrowBook(book))
- ฟังก์ชันคืน (returnBook(book))
- ดูรายการหนังสือที่ยืม (getBorrowedBooks())
- ตรวจสอบว่าผู้ใช้นี้ยืมหนังสือเล่มนี้ไปแล้วหรือยัง (hasBorrowed(book))

เงื่อนไขที่เข้า

1.Implements Interface
class User implements IUsern
ต้องมี method ตาม IUser ครบ
2.Encapsulation
ใช้ protected กับ id, name, borrowedBooks ไม่ให้เข้าถึงตรง ๆ จากภายนอก
แต่ subclass เข้าถึงได้ method getName() ให้เข้าถึงแทน
3.Aggregation
User มี borrowedBooks: Book[] คือความสัมพันธ์แบบ “User ถือครองหลาย Book” แต่ไม่ใช่การเป็นเจ้าของ
borrowBook(book: Book) เพิ่มหนังสือเข้าไปในรายการ
returnBook(book: Book) ลบหนังสือออก
hasBorrowed(book: Book) ตรวจสอบว่ามีเล่มนั้นหรือยัง
5.Abstraction
ภายนอกไม่ต้องรู้ว่าเก็บ borrowedBooks ใน array แค่เรียก method

```
class User implements IUser {
  protected id: number;
  protected name: string;
  protected borrowedBooks: Book[];

  constructor(id: number, name: string) {
    this.id = id;
    this.name = name;
    this.borrowedBooks = [];
  }

  public borrowBook(book: Book): void {
    this.borrowedBooks.push(book);
  }

  public returnBook(book: Book): void {
    const index = this.borrowedBooks.indexOf(book);
    if (index !== -1) {
      this.borrowedBooks.splice(index, 1);
    }
  }

  public getBorrowedBooks(): Book[] {
    return this.borrowedBooks;
  }

  public getName(): string {
    return this.name;
  }

  public hasBorrowed(book: Book): boolean {
    return this.borrowedBooks.includes(book);
  }
}
```

3. การจัดการห้องสมุด

- เพิ่มหนังสือเข้า Library (addBook)
- ลงทะเบียนผู้ใช้ (registerUser)
- ผู้ใช้ยืมหนังสือ (borrowBook(user, book))
- เช็คก่อนว่าผู้ใช้ลงทะเบียนหรือยัง
- เช็คว่าหนังสือมีคนยืมไปแล้วหรือเปล่า
- เช็คว่าผู้ใช้นี้ยืมเล่มนี้ไปแล้วหรือยัง
- ถ้าผ่านทั้งหมด (ยืมได้)
- ผู้ใช้คืนหนังสือ (returnBook(user, book))
- เช็คก่อนว่าผู้ใช้ลงทะเบียนหรือยัง
- เช็คว่าผู้ใช้นี้ยืมเล่มนี้จริง ๆ หรือไม่
- ถ้าผ่านทั้งหมด → คืนได้
- ดูรายการหนังสือที่ผู้ใช้ยืมอยู่ (getBorrowedBooks(user))

เข้าเงื่อนไข

1.Encapsulation

มีการใช้ private books: Book[] และ private users: User[] ไม่ให้เข้าถึงหรือแก้ไขข้อมูลโดยตรงจากภายนอกการจัดการทำผ่าน method เท่านั้น

2.Association / Aggregation

Library เก็บ books และ users ไว้ใน array ความสัมพันธ์ระหว่าง

Library Book และ Library Userความสัมพันธ์นี้เป็น Aggregation

3.Abstraction

ภายนอกไม่จำเป็นต้องรู้ว่า Library เก็บ books และ users ใน array ยังไง

ใช้ method จัดการแทนเช่น การยืม (borrowBook) และคืน (returnBook)

4.Behavior / Business Logic

borrowBook() ตรวจสอบความถูกต้อง (user ต้องลงทะเบียน)

returnBook() ตรวจสอบการคืน (ต้องเคยยืมมาก่อน)

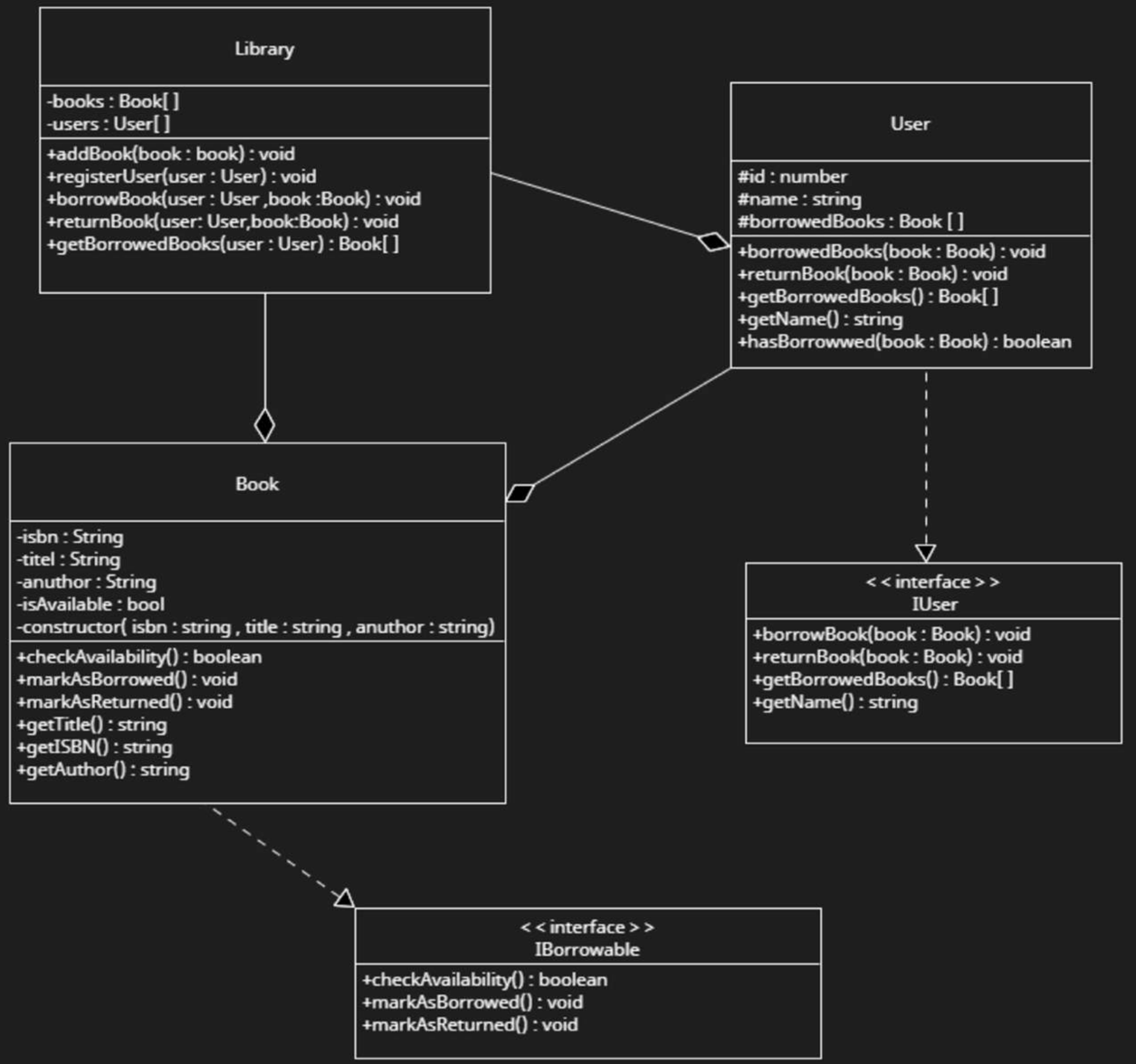
นี่คือการใส่ กฎเกณฑ์ ลงไปใน class

```
class Library {
  private books: Book[] = [];
  private users: User[] = [];

  public addBook(book: Book): void {
    this.books.push(book);
  }

  public registerUser(user: User): void {
    this.users.push(user);
  }

  public borrowBook(user: User, book: Book): void {
    if (!this.users.includes(user)) {
      console.log(`User ${user.getName()} is not registered.`);
      return;
    }

    if (user.hasBorrowed(book)) {
      console.log(`${user.getName()} already borrowed "${book.getTitle()}"`);
      return;
    }

    if (!book.checkAvailability()) {
      console.log(`"${book.getTitle()}" is not available.`);
      return;
    }

    book.markAsBorrowed();
    user.borrowBook(book);
    console.log(`${user.getName()} borrowed "${book.getTitle()}"`);
  }

  public returnBook(user: User, book: Book): void {
    if (!this.users.includes(user)) {
      console.log(`User ${user.getName()} is not registered.`);
      return;
    }

    if (!user.hasBorrowed(book)) {
      console.log(`${user.getName()} has not borrowed "${book.getTitle()}"`);
      return;
    }

    book.markAsReturned();
    user.returnBook(book);
    console.log(`${user.getName()} returned "${book.getTitle()}"`);
  }

  public getBorrowedBooks(user: User): Book[] {
    return user.getBorrowedBooks();
  }
}
```

(parameter) book: Book

Diagrams

uml.uxf




THANK YOU
GET IN TOUCH WITH US: