Application Overview: Library Management System

We'll model the system with the following classes:

1. **`Book`**: Represents a book in the library.
2. **`User`**: Represents a user of the library.
3. **`Librarian`**: Inherits from `User`, represents a librarian who can manage books.
4. **`Member`**: Inherits from `User`, represents a member who can borrow books.
5. **`Library`**: Manages the overall functionality of adding books, borrowing, and returning.

Features to Implement:

- **Inheritance**: `Librarian` and `Member` inherit common properties from `User`.
- **Polymorphism**: Different users (`Librarian` and `Member`) can interact with the `Library` in different ways.
- **Encapsulation**: Methods to protect and control access to properties.

Steps to Implement:

1. Define the `Book` class

typescript
Copy code

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

    constructor(title: string, author: string) {
        this.title = title;
        this.author = author;
        this.isAvailable = true;  // Book is available by default
    }

    // Getter method for the book's title
    getTitle(): string {
        return this.title;
    }

    // Method to check if the book is available
    checkAvailability(): boolean {
        return this.isAvailable;
```

```
    }

    // Method to borrow the book
    borrowBook(): void {
        if (this.isAvailable) {
            this.isAvailable = false;
            console.log(`${this.title} has been borrowed.`);
        } else {
            console.log(`${this.title} is currently unavailable.`);
        }
    }

    // Method to return the book
    returnBook(): void {
        this.isAvailable = true;
        console.log(`${this.title} has been returned.`);
    }
}
```

2. Define the `User` class and subclasses `Librarian` and `Member`

typescript
Copy code

```
class User {
    protected name: string;

    constructor(name: string) {
        this.name = name;
    }

    // Basic user method
    getName(): string {
        return this.name;
    }
}

// Librarian can manage books
class Librarian extends User {
    constructor(name: string) {
        super(name);
    }
```

```
    // Method to add a book to the library
    addBook(library: Library, book: Book): void {
        library.addBook(book);
        console.log(`Librarian ${this.name} added ${book.getTitle()}
to the library.`);
    }
}

// Member can borrow books
class Member extends User {
    constructor(name: string) {
        super(name);
    }

    // Method for borrowing a book
    borrowBook(library: Library, bookTitle: string): void {
        library.borrowBook(bookTitle);
        console.log(`${this.name} borrowed ${bookTitle}.`);
    }

    // Method for returning a book
    returnBook(library: Library, bookTitle: string): void {
        library.returnBook(bookTitle);
        console.log(`${this.name} returned ${bookTitle}.`);
    }
}
```

3. Define the `Library` class

typescript
Copy code

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

    // Method to add a book to the library
    addBook(book: Book): void {
        this.books.push(book);
    }

    // Method to borrow a book
    borrowBook(title: string): void {
        const book = this.books.find(b => b.getTitle() === title);
```

```
        if (book && book.checkAvailability()) {
            book.borrowBook();
        } else {
            console.log(`${title} is not available in the
library.`);
        }
    }

    // Method to return a book
    returnBook(title: string): void {
        const book = this.books.find(b => b.getTitle() === title);
        if (book) {
            book.returnBook();
        } else {
            console.log(`${title} does not belong to this
library.`);
        }
    }
}
```

4. Putting It All Together

typescript

Copy code

```
// Create a new library
const library = new Library();

// Create some books
const book1 = new Book("The Great Gatsby", "F. Scott Fitzgerald");
const book2 = new Book("1984", "George Orwell");

// Create a librarian and add books to the library
const librarian = new Librarian("John");
librarian.addBook(library, book1);
librarian.addBook(library, book2);

// Create a member and borrow/return books
const member = new Member("Alice");
member.borrowBook(library, "The Great Gatsby");
member.returnBook(library, "The Great Gatsby");
```

Running the Program:

1. Install TypeScript and set up your environment.
    - Create a new folder in VS Code.
    - Initialize a Node.js project with `npm init`.
    - Install TypeScript: `npm install typescript --save-dev`.
    - Install `ts-node` for easy execution: `npm install ts-node --save-dev`.
    - Create a `tsconfig.json` file by running `npx tsc --init`.
2. Save the above code in a `library.ts` file.
3. Compile and run the TypeScript code:
    - Run `npx ts-node library.ts` in the terminal.

Summary of OOP Concepts Used:

1. **Classes and Objects**: Classes (`Book`, `User`, etc.) represent real-world entities. Objects of these classes interact with each other.
2. **Inheritance**: `Librarian` and `Member` inherit from `User`, gaining common properties while adding unique behavior.
3. **Polymorphism**: The `Library` interacts differently with a `Librarian` (adding books) and a `Member` (borrowing/returning books).

This project gives you a good overview of how OOP principles can be applied in TypeScript. Let me know if you need further clarification or help!