|  |  |  |
| --- | --- | --- |
| Monday | Day 1 | Section 1: introduction |
| Section 2: Variables and Datatypes |
| Section 3: Operators |
| Tuesday | Day2 | Section 4: Flow Control Statements |
| Section 5: Objects and Arrays |
| Section 6: Functions |
| Wednesday | Day 3 | Section 7: Arrow Functions |
| Section 8: Variable Prefixes |
| Section 9: Interfaces |
| Section 10: Classes |
| Thrusday | Day 4 | Section 11: Inheritance |
| Section 12: Access modifiers, Enacpsulation and Static |
| Section 13: Polymorphism |
| Section 14: Type Casting |
| Section 15: Modules |
| Friday | Day 5 | Section 16: More Types |
| Section 17: Decorators |
| Section 18: Error Handling |
| Section 19: Wrap Up |

<https://www.geeksforgeeks.org/when-to-use-interfaces-and-when-to-use-classes-in-typescript/?ref=asr5>

**Problem Statement**

You are tasked with **designing a simple library management system with TypeScript**. The system should allow **tracking books, authors, and patrons.**

Use Object-Oriented Programming concepts to model these entities.

**Requirements**

**Create Classes**: Create classes for Book, Author, and Patron.

**Implement Encapsulation:** Use private or protected properties where appropriate, with getters and setters.

**Use Inheritance:** Create an **Item**base class and have Book inherit from it.

**Implement Abstraction**: Use interfaces for actions a library item can perform (e.g., borrow and return).

**Polymorphism:** Implement polymorphism to handle different types of items in the library.

Demonstrate Usage: Instantiate objects and demonstrate various OOP concepts.

**Class Details**

* Item:
  + Properties: title (string), year (number).
  + Method: displayInfo().

**Book (inherits from Item):**

* Properties: author (Author), isbn (string).
* Method: Override displayInfo() to show book-specific details.

**Author:**

* Properties: name (string), country (string).

**Patron:**

* **Properties: name (string), memberId (number).**

**Methods:**

* borrow(item: Item): Marks an item as borrowed.
* returnItem(item: Item): Marks an item as returned.

**Tasks**

Define the **Item** class and **Book**subclass.

Create an interface **Borrowable** with borrow and **returnItem** methods.

Implement the **Patron**class with methods to interact with borrowable items.

Use encapsulation for sensitive properties in classes.

Demonstrate inheritance and polymorphism by creating instances of **Book** and using **Patron** methods.

**Expected Output**

Display the details of books borrowed by a patron.

Show polymorphism by borrowing different types of items (use Book).

//solution

// 1. Define an abstract class Item with basic details  
abstract class Item {  
    protected title: string;  
    protected year: number;

    constructor(title: string, year: number) {  
        this.title = title;  
        this.year = year;  
    }

    // Abstract method to be implemented by subclasses  
    abstract displayInfo(): void;  
}

// 2. Define the Author class  
class Author {  
    constructor(  
        private name: string,  
        private country: string  
    ) {}

    getAuthorInfo(): string {  
        return `${this.name} from ${this.country}`;  
    }  
}

// 3. Book class extending Item  
class Book extends Item {  
    private author: Author;  
    private isbn: string;

    constructor(title: string, year: number, author: Author, isbn: string) {  
        super(title, year);  
        this.author = author;  
        this.isbn = isbn;  
    }

    displayInfo(): void {  
        console.log(`Book: ${this.title}, Year: ${this.year}, Author: ${this.author.getAuthorInfo()}, ISBN: ${this.isbn}`);  
    }  
}

// 4. Define a Borrowable interface  
interface Borrowable {  
    borrow(item: Item): void;  
    returnItem(item: Item): void;  
}

// 5. Patron class implementing Borrowable  
class Patron implements Borrowable {  
    private name: string;  
    private memberId: number;  
    private borrowedItems: Item[] = [];

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

    borrow(item: Item): void {  
        console.log(`${this.name} borrowed "${item instanceof Book ? item['title'] : ''}"`);  
        this.borrowedItems.push(item);  
    }

    returnItem(item: Item): void {  
        console.log(`${this.name} returned "${item instanceof Book ? item['title'] : ''}"`);  
        this.borrowedItems = this.borrowedItems.filter(i => i !== item);  
    }

    displayBorrowedItems(): void {  
        console.log(`Borrowed Items by ${this.name}:`);  
        this.borrowedItems.forEach(item => item.displayInfo());  
    }  
}

// Demonstration of the assignment

// Creating an author and a book  
const author = new Author("J.K. Rowling", "United Kingdom");  
const book1 = new Book("Harry Potter", 1997, author, "123-4567890123");

// Creating a patron  
const patron = new Patron("John Doe", 1);

// Borrowing and returning a book  
patron.borrow(book1);  
patron.displayBorrowedItems(); // Should show borrowed book details

// Returning the book  
patron.returnItem(book1);  
patron.displayBorrowedItems(); // Should show no borrowed items

Commands:

cdk init app --language=typescript

`npm run build`   compile typescript to js  
\* `npm run watch`   watch for changes and compile  
\* `npm run test`    perform the jest unit tests  
\* `npx cdk deploy`  deploy this stack to your default AWS account/region  
\* `npx cdk diff`    compare deployed stack with current state  
\* `npx cdk synth`   emits the synthesized CloudFormation template