Create a Book Inventory Service

File: `src/app/book-inventory.service.ts`

Setup:

- **a. Generate the Service:**
  Generate a service using Angular CLI:

  ```
  ng generate service book-inventory
  ```

You will:

1. **Create a Service:**
   - Create `BookInventoryService` as above with methods to get and delete books.
   - Mark the service as injectable at the root level.
2. **Update Component (TS):**
   - Inject the service into `BookInventoryComponent`.
   - On initialization, retrieve the inventory from the service.
   - Delegate deletion logic to the service and refresh the local inventory.

Tips:

- The service is marked as `@Injectable({ providedIn: 'root' })` so it's a singleton provided at the root level.
- The service holds a private inventory array.
- The `getInventory()` method returns a copy of the inventory.
- The `deleteBook(book: Book)` method updates the inventory by filtering out the specified book.

Code:

```
// src/app/book-inventory.service.ts
import { Injectable } from '@angular/core';
import { Book } from './book';

@Injectable({
  providedIn: 'root'
})
```

```
export class BookInventoryService {
  // Hard-coded inventory (could fetch this data from an
API in future)
  private inventory: Book[] = [
    {
      ISBN: "978-1492056205",
      title: "Angular Up & Running",
      author: "Shyam Seshadri",
      year: 2018,
      price: 39,
      featured: true,
      coverImages: ["/assets/angular-up-and-running.png"]
    },
    {
      ISBN: "978-1593279509",
      title: "Eloquent JavaScript, 3rd Edition",
      author: "Marijn Haverbeke",
      year: 2018,
      price: 29.99,
      featured: false,
      coverImages: ["/assets/eloquent-javascript.jpg"]
    },
    {
      ISBN: "978-1491904244",
      title: "You Don't Know JS Yet: Get Started",
      author: "Kyle Simpson",
      year: 2020,
      price: 34.99,
      featured: false,
      coverImages: ["/assets/ydkjs-cover.jpg"]
    },
    {
      ISBN: "978-1449331818",
      title: "Learning JavaScript Design Patterns",
      author: "Addy Osmani",
      year: 2012,
      price: 25.99,
      featured: true,
      coverImages: ["/assets/js-design-patterns.png"]
    }
  ];

  // Return the current inventory
  getInventory(): Book[] {
    return [...this.inventory]; // return a copy
  }

  // Delete a book by ISBN
  deleteBook(book: Book): void {
    this.inventory = this.inventory.filter(b => b.ISBN !==
book.ISBN);
  }
}
```

2. Update the Book Inventory Component to Use the Service

File: `src/app/book-inventory/book-inventory.component.ts`

You will:

- **a. Import and Inject the Service:**
  Import your service and add it to the component's constructor.

- **b. Retrieve the Inventory:**
  In `ngOnInit()`, call the service's `getInventory()` method to set your local `inventory` property.

- **c. Update the Delete Method:**
  Instead of modifying the inventory directly, call the service's `deleteBook()` method and then refresh your local inventory (if necessary).

Tips:

- Inject `BookInventoryService` in the constructor.
- On initialization, retrieve the inventory from the service.
- When deleting a book, delegate that responsibility to the service and then refresh the local inventory.

Code:

```
// src/app/book-inventory/book-inventory.component.ts
import { Component, OnInit } from '@angular/core';
import { Book } from '../book';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { HoverHighlightDirective } from './hover-
highlight.directive';
import { BookFilterPipe } from '../book-filter.pipe';
import { BookInventoryService } from ‘../book-
inventory.service';

@Component({
  selector: 'app-book-inventory',
  standalone: true,
  imports: [CommonModule, HoverHighlightDirective,
FormsModule, BookFilterPipe],
  templateUrl: './book-inventory.component.html',
  styleUrls: ['./book-inventory.component.css']
})
export class BookInventoryComponent implements OnInit {
  currentDate: Date = new Date();
  searchTerm: string = '';
  inventory: Book[] = [];

  constructor(private bookService: BookInventoryService) {}

  ngOnInit(): void {
    // Get the inventory from the service
    this.inventory = this.bookService.getInventory();
  }

  trackByISBN(index: number, book: Book): string {
    return book.ISBN;
  }

  deleteBook(book: Book): void {
    // Use the service to delete the book
    this.bookService.deleteBook(book);
    // Update local inventory after deletion
    this.inventory = this.bookService.getInventory();
  }
}
```

The service uses Dependency Injection (DI):

Dependency Injection (DI) is a design pattern used in Angular (and other frameworks) to supply components or services with their required dependencies rather than having them create those dependencies themselves.

1. What is Dependency Injection?

- **Concept:**
  DI is a pattern where a class receives its dependencies (services, objects, or values) from an external source rather than creating them itself. This helps to decouple components from their dependencies.

- **Angular Implementation:**
  Angular has a built-in dependency injection system that automatically provides instances of services to components, directives, pipes, or even other services. When you declare a dependency in a constructor, Angular's DI system "injects" the required instance.

- In this example, Angular will automatically provide an instance of `BookInventoryService` when it creates the component:

```
constructor(private bookService: BookInventoryService) { }
```

2. Benefits of Dependency Injection

- **Decoupling:**
  Components don't need to know how to create or configure their dependencies; they just declare what they need.

- **Testability:**
  It's easier to mock dependencies in unit tests since you can replace real services with mocks or stubs.

- **Reusability:**
  Services and components can be reused in different parts of an application without worrying about how dependencies are constructed.

- **Maintainability:**
  By managing dependencies centrally (e.g., via providers), you have a single place to configure or update them.

3. How Angular's DI Works

- **Providers:**
  You register classes (services) with the Angular injector by declaring them as providers. For example, using `@Injectable({ providedIn: 'root' })` makes a service available application-wide as a singleton.

- **Injection Tokens:**
  Angular uses tokens to identify dependencies. Tokens can be class types, strings, or custom objects. When you declare a dependency, Angular matches the type or token to a provider.

- **Constructor Injection:**
  Dependencies are typically injected via constructor parameters. Angular inspects the types of constructor parameters and supplies the appropriate instances.

4. Alternatives to Dependency Injection

As a comparison, here are some alternatives to DI, but DI is preferred:

- **Manual Instantiation:**
  - **Description:**
    A component could create its own dependencies using the `new` keyword.

  Example:

  ```
  export class SomeComponent {
    private bookService = new BookInventoryService();
  }
  ```

  - **Why we don't use it instead:**
    - Tight coupling: The component is directly responsible for constructing the service.
    - Difficult to test: It's harder to replace the dependency with a mock.
    - No centralized configuration: Changing how the service is created requires updating each component.