DWA_08 Discussion Questions

In this module you will continue with your “Book Connect” codebase, and further iterate on your abstractions. You will be required to create an encapsulated abstraction of the book preview by means of a single factory function. If you are up for it you can also encapsulate other aspects of the app into their own abstractions.

To prepare for your session with your coach, please answer the following questions. Then download this document as a PDF and include it in the repository with your code.

---

1. What parts of encapsulating your logic were easy?
The preview function: This function takes in parameters (author, id, image, title) and encapsulates the logic to create a preview button element. It sets the class, attributes, and inner HTML of the button element based on the provided data. The function hides the implementation details of creating the button element and returns the final element.

The fragment variable: This variable is used to create a DocumentFragment, which is an in-memory container for holding a group of DOM elements. By appending elements to the fragment, the code encapsulates the logic of building a set of elements before appending them to the actual DOM. This can provide performance benefits as the elements are added to the DOM only once when the fragment is appended.

These parts of the code encapsulate specific functionality and hide the underlying implementation details. They provide an interface for creating a preview button element (preview function) and a container for efficient element creation (fragment variable).

---

2. What parts of encapsulating your logic were hard?

Lack of Modularity: The code snippet combines multiple responsibilities within a single block, making it harder to isolate and encapsulate specific functionalities. For better

encapsulation, it would be beneficial to break down the code into smaller, reusable functions, each responsible for a specific task.

Implicit Dependency: The code references an authors object without any indication of its origin or how it's populated. It's unclear where this object comes from, and if it's a global object, it introduces a hidden dependency. Encapsulation is easier when dependencies are explicitly passed as parameters or managed within the encapsulated scope.

Mixing DOM Manipulation: The code directly manipulates the DOM by creating elements and appending them to a DocumentFragment. While this can work for simple scenarios, it can become harder to encapsulate and maintain as the complexity of the application grows. Consider separating the DOM manipulation logic into a dedicated module or using a framework that provides better abstractions for managing the UI.

---

3. Is abstracting the book preview a good or bad idea? Why?
Reusability: By abstracting the book preview functionality into a separate function, you can easily reuse it in multiple places within your codebase. This promotes code reuse and reduces duplication, which leads to cleaner and more maintainable code.

Encapsulation: Abstracting the book preview allows you to encapsulate the logic related to creating the preview button element and setting its properties (such as class, attributes, and inner HTML) within a single function. This encapsulation hides the implementation details and provides a clear interface for creating book previews, improving code organization and readability.

Maintainability: Having a separate function for creating book previews makes it easier to make changes or updates to the preview functionality. If there are any modifications or enhancements required in the future, you only need to update the logic within the abstracted function, rather than searching and modifying multiple occurrences throughout the codebase.

Testability: Abstracting the book preview allows you to write dedicated unit tests for that specific functionality. Testing a focused function is generally easier and more effective than testing a larger block of code that includes unrelated functionality.

However, there are also cases where abstracting the book preview into a separate function may not be necessary or beneficial, such as:

Single Use: If the book preview functionality is only needed in one specific location and is unlikely to be reused elsewhere, abstracting it into a separate function may introduce unnecessary complexity.

Simplified Code: If the book preview logic is simple and concise, and there is no need for extensive customization or variations, keeping it inline within the calling code may be more straightforward and readable.

In general, abstracting the book preview into a separate function is a good idea when it promotes reusability, encapsulation, maintainability, and testability. However, the decision ultimately depends on the specific context, complexity, and requirements of your codebase.
