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 logic (method) responsible for creating the book previews fragment was relatively straightforward. It has a clear and singular function, making it easy to integrate into the factory function due to its inherent abstraction.

- The logic (method) in charge of appending the book previews fragment to the DOM also contributed to the ease of encapsulation. This method acts as an internal abstraction, simplifying the process of adding the book preview fragment to the DOM. It allows for flexibility in specifying the DOM HTML element target by accepting it as an argument during the method's invocation.

---

2. What parts of encapsulating your logic were hard?

- The implementation of the method for resetting the page number within the factory function posed some challenges. The utilization of Getters and Setters added complexity to the process, making it a bit cumbersome.

- Additionally, managing the abstractions related to updating and resetting the remaining books proved to be a challenging aspect. I deliberated whether these abstractions should be integrated as methods of the Book Preview object or should be automatically executed based on specific conditions.

---

3. Is abstracting the book preview a good or bad idea? Why?

Abstracting the book preview process is indeed a beneficial idea, primarily due to the complexity involved in its implementation. By encapsulating this logic within the methods of the book preview factory function, the underlying intricate operations can be hidden, leading to the creation of a comprehensible and high-level abstraction. Moreover, this approach ensures compliance with crucial SOLID principles, particularly the Single Responsibility, Open-Closed, and Dependency Inversion Principles, promoting better code organization and maintainability.