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?

- Creating the BookPreview class and encapsulating the function to generate a book preview element were relatively straightforward.

- This class effectively abstracted the process of extracting essential data from the book object and constructing a button element to represent the book preview. Additionally, it handled the inner HTML content of the button by incorporating book data. This encapsulation resulted in a cleaner and more modular code structure, enhancing readability and maintainability.

---

2. What parts of encapsulating your logic were hard?

  - The challenging aspects of encapsulating the logic included deciding which specific functionalities to encapsulate.
  - Identifying the ideal balance between abstraction and implementation details can be complex. Furthermore, incorporating the this keyword and understanding how to effectively use getter methods added a layer of complexity. However, these challenges were essential in achieving a high degree of abstraction and encapsulation.

---

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

Encapsulating and abstracting the book preview is a sound concept for various compelling reasons:

- **Modularity and Reusability**: The encapsulation within the BookPreview class encourages modularity and reusability. This enables the creation of multiple instances of the BookPreview class for various books, each instance encapsulating the logic and manipulation specific to that book preview. This approach streamlines maintenance, testing, and expansion of the codebase.

- **Abstraction and Encapsulation:** By encapsulating the book preview functionality within a class, it becomes possible to abstract away the intricate implementation details. This abstraction offers a clean and well-defined interface for interacting with book previews. Other parts of the code can interact with BookPreview instances without the need to delve into specific implementation details. This fosters encapsulation and reduces interdependencies between components.
- **Code Organization:** The utilisation of a class helps in effectively organising related code, enhancing the code's comprehensibility and navigability. All logic and DOM manipulation linked to book previews are contained within the class, providing clarity regarding where to locate and modify this functionality.
- **Code Maintainability**: Encapsulating functionality within a class substantially improves code maintainability. When changes or new features need to be introduced to the book preview functionality, they can be implemented within the class, limiting the scope of impact on other areas of the codebase.
- **Code Consistency**: Encapsulating functionality within a class establishes a consistent and standardised approach to creating and manipulating book previews. This consistency can reduce the occurrence of bugs and simplify the codebase's understanding and maintenance, especially when multiple developers collaborate on the project.
-

By abstracting and encapsulating the book preview, the codebase becomes more organised, modular, and maintainable, which is beneficial for enhancing the overall quality and efficiency of the application.