# 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?

```
/**
 * Creates a book preview object.
 *
 * @param {Object} bookData - The data of the book for which to create the preview.
 * @param {Object} authors - An object containing author information with author IDs as keys and author names as values.
 * @returns {Object} The created book preview object.
 */
export function createPreview(bookData, authors) {
    // Destructure bookData
    const { author, id, image, title, genre } = bookData;

    // Create the book preview object
    const bookPreview = {
        id: id,
        image: image,
        title: title,
        genre: genre,
        author: authors[author],
        /**
         * Get a description of the book preview.
         *
         * @returns {string} The description of the book preview.
         */
        getDescription() {
            return `This is a book preview with title "${this.title}" by ${this.author}, in the genre of ${this.genre}, and an image URL of "${this.image}"`;
        },
    };

    return bookPreview;
}
```

Modularity: The factory function, createPreview, encapsulates the logic for creating book preview objects. It takes bookData and authors input and returns a book preview object. This modularity makes it clear where to find and modify the code responsible for creating book previews.

Code Reusability: Once you have the createPreview factory function defined, you can use it to create book previews for different books by passing different bookData and authors as arguments.

---

2. What parts of encapsulating your logic were hard?

Dependencies: If the factory function relies on external data sources or services to retrieve additional information about books or authors, managing these dependencies and ensuring they are correctly initialized can be complex.

Deciding on the structure of the book preview object and what properties it should have can sometimes be challenging. You need to ensure that the object represents the book's information accurately and provides all the necessary data for other parts of the application.

---

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

Abstracting the "book preview" can be a good or bad idea depending on the specific context and goals of the software development project.

## Why Abstracting the bookPreview is a Good Idea:

Scalability: If the application is expected to grow and accommodate new content types, abstracting the preview  can make it easier to introduce and manage these changes. It provides a flexible foundation for extending the preview functionality.

Code Organization: Abstracting the preview can improve code organization, especially if there is a variety of content-related features in the application. It helps to group related functionality together and reduces the complexity of individual components.

## Why Abstracting the bookPreview May Not Be Necessary:

Simplicity: If the application primarily deals with book previews, and there are no immediate plans to introduce other content types, abstracting the bookPreview may add unnecessary complexity to the code.

Development Time: Abstracting the concept of a bookPreview can require additional development time and effort, including designing interfaces.

Maintainability: While abstracting can enhance maintainability in some cases, it can also introduce complexity.

---