DWA_07.4 Knowledge Check_DWA7

---

1. Which were the three best abstractions, and why?

createAttributeHTML function:

- The function serves a singular purpose by creating and returning an attribute fragment.

- It operates at a high level of abstraction and offers flexibility as it can produce a document fragment for both genres and authors. Furthermore, it has the potential for expansion to generate additional **attributeHTMLs** like **ISBN** and **Publisher** attributes, provided that the attribute database source (the argument) adheres to the same key/value pair structure.

- The function can generate an attribute document fragment for various types of attribute sources; it's not constrained to a specific attribute like genres or authors, as it can handle both.

- The source databases for genres or authors are external to the function's core logic. They are passed as arguments with each invocation, depending on the specific attribute type being fragmented. This practice minimizes the need for modifications to the abstraction since new or altered attribute databases (such as authors or genres) can be supplied with ease in each function call.

handleToggleDialog function:

- The function maintains a single, well-defined task.

- It demonstrates flexibility by automatically detecting the status of a feature's dialog; if the dialog is open, it closes it, and if it's closed, it opens it.

- This flexibility extends beyond a single feature dialog and can be seamlessly applied to various dialogs, including settings, book preview, and search dialogs.

- It accommodates future expansion by allowing the addition of new feature dialogs without requiring alterations to the underlying code.

createBookPreviewsHTML function:

- The function turns the complex and repetitive code logic of creating a book previews fragment into an abstraction that is high-level and reusable.

- The books database doesn't form part of the function's inner logic; instead, it is passed as an argument with each invocation. This minimizes modification of the abstraction as new or altered book databases can be passed with each function call. For example, the supplied/main books database or the filtered books database (which is non-existent and only created when a user submits the book search form).

- However, by supplying a page number greater than 0, it indicates that the book previews fragment will be appended to a new page of the app's book catalog. As such, the function can extract the correct range of books from the book source argument based on the corresponding current page number of the app.

---

2. Which were the three worst abstractions, and why?

updateRemainingBooks function:

- This function performs an additional task separate from updating the number of remaining books. It invokes another function, **disableListButton**, which disables the "show more" next button when there are no books remaining.

- The inclusion of this extra functionality within the abstraction makes it less focused and may lead to the need for modifications if the decision to disable the button changes.

handleBookFilterSearch function:

- This function is overly long and contains sections of code logic that could have been abstracted further.

- By breaking down the code into smaller, more specialized sub-abstractions, the function could have become more straightforward and modular.
- This lack of decomposition and organization within the function makes it challenging to maintain and understand.

toggleThemeHandler function:

- This function incorporates CSS styles for both dark and light themes within its internal logic.
- The inclusion of CSS styles in the JavaScript function is generally not a good practice because it tightly couples the application's logic with its styling.
- If, in the future, the RGB color values for the dark or light themes need to be modified, changes must be made directly within the code.
- This lack of separation between logic and styling may hinder the maintainability and flexibility of the abstraction, making it challenging to adapt the application's visual appearance without code alterations.

---

3. How can The three worst abstractions be improved via SOLID principles.

updateRemainingBooks function:

- To adhere to the **Single Responsibility Principle** (**SRP**), the function should be restructured to specifically return the number of remaining books. Simultaneously, a separate **disableListButton** function should be adapted to receive this count as an argument. Consequently, this adjustment ensures that the **disableListButton** function's sole responsibility is managing the button's status based on the provided count. This separation of concerns aligns with the **SRP**, fostering a more robust and maintainable codebase.

handleBookFilterSearch function:

- Applying the **Single Responsibility Principle** (**SRP**), the function's code can be further abstracted by creating sub-abstractions for specific tasks such as iterating over the main book library database to find searched book titles, genres, and/or authors. This refined design allows for easier comprehension and maintainability of the codebase. Additionally, the **Open-Closed Principle** can be easily applied by making these sub-abstractions open for extension through different search criteria without modifying the existing code in the future.

toggleThemeHandler function:

- To enhance the function's flexibility, maintainability, and alignment with **SOLID** principles, the CSS theme styles object can be encapsulated within a dedicated module, such as "**data.js.**" This approach ensures a clear separation of concerns and upholds the **Open-Closed Principle**.

  The function can then accept this CSS style object as an argument, promoting the function's independence from low-level details and reinforcing its reliance on abstractions. This restructuring aligns with the **Dependency Inversion Principle**, contributing to a more robust and adaptable design.