DWA_07.4 Knowledge Check_DWA7

---

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

- createPreviewElement(): Abstracts the process of creating a preview element for a book. It takes a book object as input and returns a preview element with the necessary HTML structure and content. This abstraction encapsulates the details of creating a preview element, making the code more modular and reusable.
- createOptionElements(): Abstracts the process of creating option elements for a dropdown menu. It takes a container name, a default value, and options as input, and generates the corresponding option elements. This abstraction allows for the dynamic creation of dropdown options based on the provided data, reducing code duplication and enhancing maintainability.
- app.init(): This method abstracts the initialization process of the application. It sets up event listeners and handles various interactions with the DOM. By encapsulating the initialization logic within this abstraction, the code becomes more organized and easier to manage.

---

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

- Matching logic in the search form submission event listener: The matching logic for filtering books based on user input is embedded within the event listener function. This makes the code harder to read and understand. It would be better to extract the matching logic into a separate function, improving code readability and maintainability.
- Inline styling of theme colors: The code sets the theme colors directly by manipulating the CSS variables using inline style properties. This tightly couples the theme-related code with the DOM, making it harder to modify or extend the theming functionality. It would be better to abstract the theme handling into a separate module or class, adhering to the Single Responsibility Principle.
- Inline HTML generation in the list button's innerHTML assignment: The code generates HTML markup directly within the innerHTML assignment for the list button. This approach mixes HTML generation with JavaScript code, which can lead to readability issues and potential security vulnerabilities. It would be better

to separate the HTML generation into a template or utilize a templating engine for improved separation of concerns.

---

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

- Extract the matching logic into a separate function or class: By separating the matching logic into a dedicated module or class, the code adheres to the Single Responsibility Principle. This allows for better organization and maintainability, as well as enabling easier testing and reusability.
- Implement a ThemeHandler class: Create a separate class responsible for managing the application's theme. This class should encapsulate the theme-related functionality and adhere to the Single Responsibility Principle. It can provide methods for setting the theme, updating the CSS variables, and handling theme-related events.
- Utilize a templating engine or separate HTML templates: Instead of generating HTML markup directly within the JavaScript code, consider using a templating engine like Handlebars or adopting a modular approach with separate HTML templates. This separation of concerns improves code readability, facilitates collaboration between front-end designers and developers, and reduces the risk of introducing HTML injection vulnerabilities.
