DWA_07.4 Knowledge Check_DWA7

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

Using Document Fragments:

In several places, the code uses DocumentFragment objects (e.g., starting, genreHtml, authorsHtml) to create and manipulate DOM elements before appending them to the actual document. This approach is efficient and helps reduce unnecessary reflows and repaints in the browser, leading to better performance when updating the DOM with multiple elements. By utilizing DocumentFragments, the code avoids excessive direct manipulation of the live DOM, making the process smoother and more optimized.

Event Listeners and Handlers:

The code effectively abstracts away the logic for handling various user interactions, such as submitting forms, clicking buttons, and displaying detailed book information. By organizing event listeners and associated handler functions, the code separates concerns and promotes modularity. This abstraction helps keep the event-handling logic organized, making it easier to maintain and extend the application's functionality in the future.

Conditional Theme Handling:

The conditional theme handling abstraction, based on the user's preferred color scheme, allows the application to adapt its appearance to the user's browser settings. By encapsulating the theme-specific logic within the conditional block, the code keeps theme-related operations organized and concise. This abstraction enhances user experience and provides a seamless transition between light and dark themes based on user preferences.

---

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

Error Handling within Loops:

In the loops where you are filtering and matching books based on user input (such as in the search form), there are new console.error("invalid input"); statements inside the loop. Throwing errors within a loop like this can be problematic as it could lead to multiple error messages being thrown, potentially overwhelming the user. A better approach might be to accumulate the errors and then handle them after the loop has finished running. This would provide a more user-friendly way to inform the user about issues with their input.

Inline Styling:

The code uses inline styling to adjust CSS variables for the theme. While this works, it might be better to separate the CSS styles from the JavaScript code. This could be achieved by toggling CSS classes based on the theme selection. Separating styles from the JavaScript logic enhances maintainability and follows the principle of separating concerns.

Repetitive Code:

There is some repetition in the code where similar operations are performed in different event listeners. For example, creating button elements for book previews is repeated in multiple places. This can make the code harder to maintain and update. A more abstracted approach could involve creating functions for common tasks, reducing duplication and improving code readability.

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

Error Handling within Loops:

Single Responsibility Principle (SRP): The loop responsible for filtering and matching books based on user input is handling two responsibilities: filtering and error handling. To adhere to SRP, separate the error handling from the filtering logic. You could create a separate function to validate the user input and throw errors if necessary, outside of the loop. This would make the code more focused and easier to maintain.

Inline Styling:

Open/Closed Principle (OCP): Instead of directly setting CSS variables through inline styling in JavaScript, you could create CSS classes that define different themes. By toggling classes based on the theme selection, you follow the OCP by extending the functionality (adding new themes) without modifying existing code. This also enhances separation of concerns, making the code more modular and maintainable.

Repetitive Code:

Single Responsibility Principle (SRP) / Don't Repeat Yourself (DRY): The repetitive code for creating button elements for book previews can be refactored into a single function. This function would encapsulate the creation of the button element, reducing duplication and adhering to the SRP and DRY principles. Additionally, you could use the Interface Segregation Principle (ISP) to ensure that your functions and classes have specific and focused responsibilities.