DWA_07.4 Knowledge Check_DWA7

---

1. Which were the three best abstractions, and why?
In the given code, the top three abstractions are:

1. Preview Function: The `preview` function abstracts the process of creating a preview element for a book. It takes in the necessary data such as author, id, image, and title, and dynamically generates the HTML markup for the preview. By encapsulating this functionality within a separate function, the code becomes more modular and easier to understand. It promotes reusability and improves code readability.

2. CreateOptionsHtml Function: The `createOptionsHtml` function abstracts the process of generating HTML options for genres and authors in the search form. It takes in the option name (e.g., "All Genres" or "All Authors") and the corresponding genre or author data. It dynamically creates the HTML options and returns them as a fragment. This abstraction allows for the easy addition of new options or changes to the genre and author data without modifying the logic that generates the HTML options.

3. ShowMoreHTML Function: The `showMoreHTML` function abstracts the logic related to displaying the "Show more" button and the remaining number of books. It sets the text and visibility of the button based on the number of remaining books to be shown. By encapsulating this logic in a separate function, it improves code readability and makes it easier to manage the behavior of the "Show more" button throughout the application.

According to the SOLID principles these three functions adhere to the Single responsibility principle, Open-Closed principle and Interface Segregation Principle. How they do this is that they have a single responsibility that focuses on specific tasks, which means that they have well defined purposes that makes the code maintainable and easier to understand. The codes of the functions also allow modification because of their specific functions which takes away the need to modify the core code adhering to the Open-Closed principle. They also promote a clear separation of concerns which avoids the creation of bloated or monolithic functions or objects adhering to the Interface Segregation Principle.

---

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

1. Event Listeners: The event listeners for search toggling, theme settings, closing the data list, and form submissions are all defined directly within the code. This approach can make the code harder to read, maintain, and test.

2. Direct DOM Manipulation: The code directly manipulates the DOM by selecting elements and modifying their properties, such as `innerHTML`, `innerText`, and `toggleAttribute`. While direct DOM manipulation can be necessary in certain situations, excessive use of it can lead to code that is tightly coupled to the specific HTML structure.

3. Global HTML Object Literal: The `html` object literal serves as a centralized container for references to various HTML elements used throughout the application. While it provides a structured way to access these elements, it introduces tight coupling between the HTML structure and the JavaScript code.

---

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

1. To separate concerns and adhere to the Single Responsibility Principle (SRP), we can extract the event handling logic into separate functions or classes. By decoupling the event handling from the rest of the code, we improve modularity, maintainability, and testability.

2.To address the violation of the Open-Closed Principle (OCP) and the Dependency Inversion Principle (DIP), we can introduce a layer of abstraction between the

application logic and the DOM manipulation. This can be achieved by using a templating engine or a component-based framework. These tools provide a higher-level interface for generating and manipulating the HTML, allowing for better separation of concerns and making the code more modular and flexible.

3. Instead of relying on a global object literal for HTML element references, we can adopt a more modular approach by encapsulating related elements into reusable components or modules. This follows the principles of the Single Responsibility Principle (SRP) and the Dependency Inversion Principle (DIP). Each component can manage its own HTML elements and expose a clean interface for interacting with them. By reducing the reliance on a global object and promoting encapsulation, we improve code organization, maintainability, and testability.
