DWA_07.4 Knowledge Check_DWA7

---

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

1. `‘preview’` function: This function encapsulates the logic for creating a book preview element. It takes in author, id, image, and title as parameters and generates the required HTML markup. This abstraction promotes code reuse, enhances readability, and separates concerns by encapsulating the preview element creation.
2. `‘createOptionsHtml’` function: This function creates HTML options for genres and authors based on the provided data. It takes in the option name and the corresponding data object as parameters and returns a document fragment containing the generated options. This abstraction promotes code reuse and enhances readability by encapsulating the option creation logic.
3. `‘showMore’` function: This function updates the content of the "Show more" button based on the number of remaining books to display. It encapsulates the logic for updating the button text and remaining count. This abstraction improves code readability by separating the button update logic into a separate function.

---

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

1. Event listeners with inline anonymous functions: Several event listeners in the code use inline anonymous functions. For example:

```
document.querySelector("[data-list-button]").addEventListener("click", () => {
```

This approach makes the code harder to read and maintain, especially when the function logic becomes more complex. It violates the Single Responsibility Principle (SRP) by mixing event handling with business logic. To improve this, separate the event handling logic into named functions and attach those functions as event listeners. This promotes code reusability and enhances maintainability.

2. Conditional styling in the settings form event listener: The event listener for the settings form performs conditional styling based on the selected theme. It directly

manipulates the CSS properties of the document's root element. This violates the Open-Closed Principle (OCP) by tightly coupling the event listener to a specific styling implementation.

To improve this, decouple the event listener from the styling implementation. Instead of directly manipulating the CSS properties, emit a custom event or use a callback function to notify the appropriate styling module or class about the theme change. This way, the styling logic can be encapsulated in a separate module or class, adhering to the Single Responsibility Principle (SRP) and promoting modular and maintainable code.

3. Direct DOM manipulation in the search form event listener: The event listener for the search form directly manipulates the DOM by appending preview elements to the document. While it achieves the desired functionality, it tightly couples the search form logic with the DOM structure.

To improve this, separate the DOM manipulation concerns from the search form event listener. Instead of directly appending elements to the document, create a separate function or class responsible for rendering the preview elements. The search form event listener should call this function or class to update the DOM with the desired elements. This adheres to the Single Responsibility Principle (SRP) and improves code organization and maintainability.

---

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

1. Replace inline anonymous functions with named functions: Separate the event handling logic into named functions and attach those functions as event listeners. This promotes code reusability and enhances maintainability. For example:

```
const handleListButtonClick = () => {
  // ...
document.querySelector("[data-list-button]").addEventListener("click",
handleListButtonClic
```

2. Decouple the event listener from styling implementation: Instead of directly manipulating the CSS properties, emit a custom event or use a callback function to notify the appropriate styling module or class about the theme change. This way, the styling logic can be encapsulated in a separate module or class, adhering to the Single Responsibility Principle (SRP) and promoting modular and maintainable code.
3. Separate DOM manipulation concerns from the search form event listener: Create a separate function or class responsible for rendering the preview elements. The search form event listener should call this function or class to update the DOM with the desired elements. This adheres to the Single Responsibility Principle (SRP) and improves code organization and maintainability.
