DWA_07.4 Knowledge Check_DWA7

---

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

- Functions:

I have used Functions like createPreviewFunction, createOption, and event listeners such as addEventListener to hide and perform specific tasks. This organizes the code and makes it easier to understand and maintain.

- Objects:

I have used Objects to represent data structures, such as books, authors, and genres. These objects provide a convenient way to store and access related information. For example, the authors object is used to retrieve the name of an author based on their ID, and the books object contains details about each book.

- Event-driven programming:

I have used event listeners and handling events to allow interactivity. Event listeners such as addEventListener are used to detect user actions, such as clicking a button or submitting a form.
The corresponding event handlers define the behavior that should occur when an event is triggered. This event-driven approach enables dynamic updates and interactions with the user interface.

- callback function.

document.querySelector('[data-list-button]').addEventListener('click', () => {...}): Listens for a click event on the list button element and executes the provided callback function.
document.querySelector('[data-list-items]').addEventListener('click', (event) => {...}): Listens for a click event on the list items and executes the provided callback function.

---

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

Inconsistent Naming:
The naming conventions used in the code are inconsistent. For example, the function createPreviewFunction should ideally be named createPreview for clarity and consistency.

Lack of Modularization:
The entire code is written in a single script.js file, which can make it harder to read, understand, and maintain. It's generally recommended to modularize code into separate files based on their functionality and responsibilities.

Unoptimized DOM Manipulation:
The code snippet includes multiple DOM manipulations, such as appending elements and modifying styles, within loops. Performing frequent DOM manipulations can have a negative impact on performance. It's generally more efficient to manipulate the DOM outside of loops or use techniques like document fragments to minimize reflows.

---

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

Lack of Single Responsibility Principle (SRP):

The createPreviewFunction function has a mixture of responsibilities:
creating a preview element and appending it to a fragment.
To improve this, we can separate these responsibilities into two separate functions: one for creating the preview element and another for appending it to the fragment.

Violation of Open/Closed Principle (OCP):

The code contains hard-coded logic for setting the theme based on the preferred color scheme. To make this code more extensible, we can introduce a theme manager that encapsulates the theme-related functionality.

Interface Segregation Principle (ISP) violation:
The code doesn't explicitly demonstrate an ISP violation, but it's important to note that adhering to SOLID principles also involves ensuring that interfaces are not bloated with

unnecessary methods. If you have any specific interface or abstraction in mind that needs improvement, please let me know.
