DWA_07.4 Knowledge Check_DWA7

---

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

1. `createPreview` function

The code demonstrates a reasonable level of abstraction by encapsulating the creation of the preview element and generating HTML dynamically based on the book object.

Some positive aspects of this abstraction include:

Reusability: The createPreviewElement function can be used multiple times to create preview elements for different books.

Separation of concerns: The function focuses on creating the HTML structure and styling for the preview element, keeping the code organized and modular.

Clear documentation: The JSDoc comments provide useful information about the function's purpose, parameters, and return value.

Iteration and display: The code snippet efficiently iterates over a subset of matches and appends the generated preview elements to a container element (starting).

2. `createGenreOption`

This function abstracts the creation of an option element for a genre. It takes the genre ID and name as parameters and returns an option element with the appropriate values. Abstracting this functionality helps separate the concerns of creating the HTML element from the logic of iterating over genres and creating options. It promotes code reuse and enhances maintainability.

3. `handleListItemClick`

Naming: The function name handleListItemClick gives a general idea of what the function does, but it could be more descriptive to convey its purpose more explicitly. Consider a name that reflects the specific action being handled, such as handlePreviewClick or handleListElementClick.

Event Path Processing: The code uses Array.from(event.path || event.composedPath()) to obtain the event path array. While this can work, it relies on the browser support for the composedPath() method. A more robust and cross-browser solution would be preferable, such as using the event target and its ancestors to determine the desired element.

Nested Loops: The code uses nested loops to find a matching book based on the dataset.preview attribute of the clicked element. This can be simplified by using methods like Array.find() or finding a more efficient way to retrieve the book directly using the dataset value.

Element Selection: The code repeatedly uses querySelector and checks for the instance type (instanceof) for multiple elements. It would be better to have a dedicated function or utility to handle these operations, reducing code duplication and making it more readable.

Type Annotations: The code includes //@ts-expect-error comments, indicating that type checking errors are intentionally ignored. While this can be necessary in certain cases, it is generally better to address type issues properly or refactor the code to ensure type safety.

---

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

1. `hanndleSearchFormSubmit`

The function seems to have a clear responsibility of handling the form submission and filtering the books.

Reusability: The function can be used to handle the submit event for any search form and filter a list of books based on the form inputs. This reusability is beneficial.

Readability: The code appears to be straightforward and readable, making it easier to understand and maintain.

2. `dataListButton`

The improvements could be to enhance modularity and maintainability.

3. `handleSettingFormSubmit`

Error Suppression: The line //@ts-expect-error suppresses TypeScript errors. While it may be necessary in some cases, it's generally recommended to address the type errors instead of suppressing them. Ignoring type errors can lead to potential bugs or issues in the code.

Inline Styling: The code directly modifies the CSS variables --color-dark and --color-light on the documentElement (root element). While this approach works, it's worth considering if there are better ways to handle theme changes, such as using CSS classes or toggling predefined stylesheets. This could improve code organization and maintainability.

Query Selector: The code uses document.querySelector to find the settings overlay element. It's important to ensure that the query selector targets the correct element reliably. If there are multiple elements with the attribute [data-settings-overlay], it might not select the desired one. Consider using a more specific selector to avoid potential issues.

---

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

1. `hanndleSearchFormSubmit`

Single Responsibility Principle (SRP):

Separate the responsibilities of handling the form submission and filtering the books into separate functions or classes. This way, each component has a clear and single responsibility.
The handleSearchFormSubmit function can focus solely on handling the submit event and extracting form data.
Open/Closed Principle (OCP):

Create an abstraction for the book filtering logic, such as an interface or base class, that defines a common contract for filtering books.
Implement different filtering strategies that adhere to the abstraction, allowing for easy extension or replacement of filters in the future without modifying the existing code.
Liskov Substitution Principle (LSP):

If there are multiple types of books or filtering criteria, consider defining a common interface or base class for books to ensure substitutability and polymorphism.
Interface Segregation Principle (ISP):

If the codebase involves multiple components or modules, consider defining interfaces that segregate the responsibilities and dependencies of each component. This helps ensure that clients only depend on the specific interfaces they require.
Dependency Inversion Principle (DIP):

Instead of relying on global variables like books, consider passing the necessary dependencies (e.g., a book repository or service) as parameters to the functions or classes that need them. This promotes loose coupling and facilitates easier testing and swapping of dependencies.

2. **dataListButton**

Single Responsibility Principle (SRP):

Extract the rendering logic into a separate function or class responsible for generating the book preview elements.
Move the event listener setup and pagination logic into another function or class responsible for handling user interactions and managing the list display.
Open/Closed Principle (OCP):

Introduce an abstraction or interface for the book rendering logic to allow for different rendering strategies (e.g., different layout options or presentation styles) that can be easily extended without modifying the existing code.
Dependency Inversion Principle (DIP):

Instead of directly referencing the DOM elements (dataListButton and listItemsElement), consider passing these elements or their selectors as parameters to the rendering function or class. This promotes decoupling from the specific DOM structure and allows for easier testing and flexibility in the future.

3. **handleSettingFormSubmit**

Single Responsibility Principle (SRP): The handleSettingsFormSubmit function appears to have a single responsibility, which is to handle the submit event of the settings form and update the theme colors. It does not appear to violate the SRP.

Open-Closed Principle (OCP): The code snippet does not explicitly demonstrate the application of the OCP. However, if the intention is to allow for easy extension or modification of the theme colors, the code could be refactored to use a theme configuration object or class that can be extended or replaced without modifying the existing code.

Liskov Substitution Principle (LSP): The code does not explicitly involve any inheritance or polymorphism, so the LSP is not directly applicable here.

Interface Segregation Principle (ISP): The code does not define any interfaces, so the ISP is not directly applicable here.

Dependency Inversion Principle (DIP): The code snippet does not show explicit dependencies on abstractions or inversion of control. However, it's worth noting that the code relies on the document object and its methods directly. To improve testability and flexibility, you could consider abstracting the interaction with the DOM by introducing a separate module or class that handles DOM manipulation and injecting it into handleSettingsFormSubmit as a dependency.
