DWA_07.4 Knowledge Check_DWA7

---

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

-CreateButtonElement
This function abstracts the creation of a button element representing a book. It takes a book as input and uses its properties to construct the HTML structure of the button element, including the image and text. It encapsulates the logic for creating the button element and makes the code more readable

-CreateOptionElement
This function abstracts the creation of an option element for a select dropdown. It takes a value and text as input and creates the option element with the corresponding properties. Abstracting the functionality into a separate function promotes code reusability and separates the concerns of making the option element from the rest of the code.

-CreateOptions
It inputs the data object and the container's name (genre or author). It creates the options using the “createOptionElemnt” function and appends them to the appropriate container. This abstraction helps in reducing code duplicates and makes the code more maintainable

---

2. Which were the three worst abstractions, and why?
-Booklist
It would have been beneficial to break down the code into smaller, reusable modules or functions to improve code organization, readability, and maintainability it lacked modularization.
-CreateButtonElement

Mixing Ui manipulation with data handling. Directly manipulate the DOM by creating HTML elements and appending them. This can lead to code that is slightly cooped to the UI, making it harder to change or update the UI in the future. It would be better to separate the UI manipulation logic from the data handling.

-Lack of error handling
My code did not have any error-handling mechanism. It is important to handle potential errors, such as network failures or invalid user inputs, to provide a better user experience and prevent unexpected issues.

---

3. How can The three worst abstractions be improved via SOLID principles.
1. Single Responsibility Principle(SRP)
-Extract the logic for creating button elements and options into separate classes or functions. This way the “createButonElemnt” and “createOptions” functions will have a single responsibility and can be easily reused and tested
-Create a “buttonElementFactory” class or function that handles the creation of button elements based on the book object
-Create an “OptionElementFactory” class or function that handles the creation of option elements for select dropdowns
-Move the code related to handling button clicks (e.g ‘handleSeacrhCancel’, ‘handleSettingsCancel’ into separate event handler classes or functions

2. Open-Closed Principle (OCP)
-The code can be made more extensible by using abstraction and interfaces. For example, you can define an interface like ‘buttonElemnFactoryInterface’ and have multiple implementations based on different book representations or Ui requirements. This way, you can add or modify the behavior of button elements without modifying existing code.

3. Dependency Inversion Principle (DIP)

-Use dependency injection to provide dependencies to the functions or classes instead of hard-coding them. For example, pass the ‘books’ , ‘authors’ , and ‘BOOK_PER_PAGE’ as parameters to the functions that require them. This reduces the coupling between modules and makes them more flexible and testable.
