DWA_07.4 Knowledge Check_DWA7

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

- The Create Preview Element function:
  This function encapsulates the creation of a preview element for a book. It abstracts away the details of creating the HTML structure for a book preview. This abstraction improved readability and maintainability.

- The createSearchOptions and initialiseSearchOption functions:
  This function abstracts the process of creating the search options. It makes the code modular and reusable,

- The setTheme function:
  This function abstracts the theme-setting logic and makes it easy to manage and toggle between themes. The abstraction improves code readability.

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

- The DOM manipulations and event listeners:
  The query selectors and event listeners are scattered, and make the code tightly coupled and difficult to maintain.

- The inline HTML in the createPreviewElement function is mixed with JS code which makes it hard to read and maintain.

- Related functionalities are not grouped into separate classes and modules and lack clear boundaries and organization.

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

I could refactor this code according to the SOLID principles
Single Responsibility Principle (SRP):I could create modules or classes that have a single responsibility. Encapsulate related functionalities within separate modules or classes, focusing on a single responsibility for each. This will help to achieve better separation of concerns and improve maintainability.

Open/Closed Principle (OCP): I could make the code to be open for extension but closed for modification. Encapsulate event listeners and DOM manipulations within appropriate modules or classes, allow them to be extended without modifying the existing code. This will make the code more flexible and easier to maintain.

Dependency Inversion Principle (DIP): Depends upon abstractions, not concretions. create modules or classes in a way that they depend on abstractions rather than concrete implementations. This allows for better decoupling and easier testing, maintenance, and future changes.