# DWA_07.4 Knowledge Check_DWA7

---

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

1. **‘createBookPreviewElement(book)’**:
This function encapsulates the creation of a book preview element. It takes a book object as input and generates an HTML element with the necessary structure and content. It abstracts away the details of creating the element, making the code more readable and maintainable.

2. ‘**renderBookPreviews()’**:
This function handles the rendering of book previews on the page. It takes a slice of `matches` (an array of books) and iterates over them, calling `createBookPreviewElement()` to create each preview element. It abstracts the rendering logic, separating it from other concerns and promoting the Single Responsibility Principle.

3. **‘updateListButton()’**:
This function updates the list button's text and disabled state based on the number of remaining books. It encapsulates the logic for updating the button, making it more reusable and easier to understand.

---

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

1. ‘**handleSearchFormSubmit(event)’**:
This function handles the form submission for the search functionality. It performs multiple tasks, including filtering the books, updating the UI, and manipulating DOM elements. To improve this abstraction, we can apply the Single Responsibility Principle and separate the concerns. We can create separate functions for filtering the books and updating the UI, making the code more modular and maintainable.

2. ‘**handleListButtonClick()’**:
This function handles the list button click event. It retrieves the next set of books to display and appends them to the DOM. However, it combines the retrieval logic with the DOM manipulation, violating the Single Responsibility Principle. A better approach would be to separate the retrieval of books from the DOM manipulation, creating a clear separation of concerns.

3. ‘**createBookPreviewElement(book) (duplicated)’**:
Although this function is one of the best abstractions mentioned earlier, there is a duplication of the same function in the code. Duplicated code violates the DRY (Don't Repeat Yourself) principle and can lead to maintenance issues. To improve this, we can remove the duplicated function and keep only one instance of `createBookPreviewElement(book)`.

---

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

1. For '**handleSearchFormSubmit(event)'**, we can create separate functions for filtering the books based on search criteria and updating the UI. Following the Single Responsibility Principle, the filtering logic can be extracted into a separate function, such as `filterBooks(filters)`. Similarly, the UI update can be abstracted into a function like `updateUIWithFilteredBooks(filteredBooks)`. This way, each function will have a clear responsibility and can be independently tested and maintained.

2. In **'handleListButtonClick()'**, we can separate the logic for retrieving the next set of books from the DOM manipulation code. We can create a function, such as `getNextBooksSlice()`, that returns the next slice of books based on the current page and number of books per page. Then, the retrieved slice can be passed to a separate function responsible for appending the new book previews to the DOM, such as `appendBookPreviewsToDOM(bookSlice)`. This separation of concerns will improve code readability and maintainability.

3. As mentioned earlier, there is a duplicated '**createBookPreviewElement(book)'** function in the code. To follow the DRY principle, we can remove one of the duplicated functions, keeping only a single instance of `createBookPreviewElement(book)` in the code. The duplicated function can be safely removed without affecting the functionality.

---