# DWA_07.4 Knowledge Check_DWA7

---

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

Book Class:
The Book class is a good choice because it neatly groups together all the information and actions related to a book.This helps keep things organized and follows the idea that each thing (in this case, a book) should have one main purpose.

Validation Functions:
The validateMatchesAndPage function is a good tool for checking if certain data is correct.It does this one specific job well, which is making sure the data is in the right format, and that's why it's a good choice.

Create Preview Element Function:
The createPreviewElement function is a handy way to make a small piece of a book's information for display.
It's a good choice because it's responsible for just one thing: creating these small pieces for books to show in a list.

---

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

Event Listeners:
The way we set up event listeners for different buttons and elements is a bit tangled and doesn't follow the idea that each part of our code should do just one thing.
To make it better, we can split it up so that each button has its own code for what happens when it's clicked. This makes our code easier to understand and change later.

Theme Configuration Logic:
Inside the code that handles the save button click, we mix together two things: changing how the app looks (theme) and working with what's on the screen (UI).It's a good idea to

separate these two jobs. One part of our code should handle changing themes, and another part should deal with the user interface. This makes our code more organized and less confusing.

Show More Button Logic:
The code for making the "Show More" button work and changing how the books appear on the screen is all mixed together.
We can make it cleaner by splitting these tasks into separate pieces of code. One part can handle the button's job, and another can deal with how the books look. This follows the idea that each piece of code should do just one job.

---

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

Event Listeners:
Single Responsibility Principle (SRP): Separate the event handling logic into distinct functions or classes, each responsible for a specific action. For example, create separate functions/classes for handling search, settings, and book details.

Open-Closed Principle (OCP): Make the event handling code extensible by allowing easy addition of new event handlers without modifying existing code. You can achieve this by implementing a dynamic event registration system.

Theme Configuration Logic:
Single Responsibility Principle (SRP): Extract the theme configuration logic into a separate class or function that handles theme-related operations only. This class or function should not directly manipulate the UI elements but should provide an API to update the theme.

Dependency Inversion Principle (DIP): Abstract away the theme configuration implementation so that it can be easily replaced or extended without changing the higher-level logic.

Show More Button Logic:
Single Responsibility Principle (SRP): Extract the logic for handling the "Show More" button click event into a dedicated function or class. This function or class should be responsible for updating the book previews and managing the state of displayed books.
Open-Closed Principle (OCP): Design the "Show More" functionality to be extensible, allowing for easy modification or extension of the behavior without altering the existing

code. This could involve defining an interface for book preview management and implementing it in different classes for different behaviors.

---