Which were the three best abstractions, and why?

1. **createPreviewElement**: This function abstracts the creation of a preview element for a book. It encapsulates the logic for creating the necessary HTML elements and setting their attributes based on the book object. By abstracting this functionality into a separate function, it improves code organisation and readability. It allows the code to focus on the high-level concept of creating a preview element without getting into the details of DOM manipulation.
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. **handleSettingsFormSubmit**: This function abstracts the handling of the submit event for the settings form. It extracts the form data, processes the theme value, and updates the CSS variables accordingly. By abstracting this logic into a separate function, it improves code organisation and readability. It also allows for easier modification or extension of the handling logic without impacting other parts of the codebase.

Which were the three worst abstractions, and why?

1. **Inline HTML Generation**: The code includes multiple instances of inline HTML generation using template literals. While it works for simple cases, it can become hard to read and maintain as the HTML structure and complexity grow. It would be better to separate the HTML templates into separate functions or templates using a templating engine for better code organisation and readability.
2. **Direct DOM Manipulation**: The code directly manipulates the DOM by creating elements and appending them to specific containers. This tightly couples the JavaScript code with the HTML structure, making it harder to modify or refactor the HTML layout without impacting the JavaScript logic. Using a modern JavaScript framework or library that supports declarative rendering and component-based architecture, such as React or Vue.js, would provide better separation of concerns and maintainability.
3. **Event Handling**: The code manually attaches event listeners to DOM elements using imperative event handling. While it works, it can lead to potential memory leaks or event listener conflicts if not properly managed. Utilising a library or framework that provides a more declarative approach to event handling, such as event delegation or reactive event binding, would simplify event management and improve code maintainability.

How can the three worst abstractions be improved via SOLID principles?

Single Responsibility Principle (SRP):
- Identify classes or modules that have more than one responsibility.
- Split these classes into smaller, more focused classes, each with a single responsibility.
- Ensure that each class or module has only one reason to change.

Open/Closed Principle (OCP):
- Identify code that needs to be modified when adding new functionality.
- Encapsulate the varying behaviour behind an interface or an abstract class.
- Implement new functionality by extending the interface or abstract class rather than modifying existing code.

Interface Segregation Principle (ISP):
- Split large interfaces into smaller and more specific interfaces.
- Clients should not be forced to depend on interfaces they don't use.
- Implement interfaces that are tailored to the specific needs of the clients.