DWA_07.4 Knowledge Check_DWA7

---

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

1. **handleSettingsOverlayToggle:** This abstraction effectively encapsulates the behavior of toggling the settings overlay's visibility. It follows the SRP by handling a single responsibility – toggling the overlay

```
// Existing example: handleSettingsOverlayToggle function has a
single responsibility of toggling the settings overlay's visibility.
export const handleSettingsOverlayToggle = () => {
    const settingsOverlay =
document.querySelector('[data-settings-overlay]');
    settingsOverlay.toggleAttribute('open');
};
```

2. **handleSettingsSave:** This function manages the logic for changing the theme based on user input. It clearly encapsulates the theme-changing behavior and maintains a focused responsibility.

```
// Existing example: handleSettingsSave function can be extended for
new themes without modifying the existing code.
export const handleSettingsSave = (event) => {
    // ... existing code ...
    if (isDay) {
        // Update light and dark colors for the day theme
    } else {
        // Update light and dark colors for the night theme
    }
    // ... existing code ...
};
```

3. **createSearchOverlay:** This function creates the search overlay and populates the author and genre selectors with options.

---

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

1. **handleShowMoreClick:** While this function handles the "Show More" functionality, it directly interacts with the `searchButton` and the `PAGES` variable from outside its scope. This violates the Single Responsibility Principle and encapsulation. It should ideally only manage the "Show More" action and delegate interactions with other elements to their respective modules.

2. **handlePreviewClick:** This function is responsible for handling the click event on a preview element and updating the overlay with corresponding data. However, it also performs DOM manipulations and styling updates. Splitting these responsibilities into separate functions or classes would improve modularity and adherence to SRP.

3. createSearchOverlay: it could be split further for improved modularity, it effectively encapsulates the process of setting up the search overlay and options.

```
/**
 * Event handler for the header search button. When clicked it will open the
search
 * overlay and allow the user to input data that they would like the books to
 * be filtered by. This event handler is also used to create the options in
both
 * the author and genre selectors
 */
export const createSearchOverlay =
    () => {
        // Variable used to store the overlay and used to toggle it's open
attribute
        const searchOverlay =
            document.querySelector(
                '[data-search-overlay]'
            );
        searchOverlay.toggleAttribute(
            'open'
```

```
);
// Variable used to store the input for title of the search overaly
const title =
    document.querySelector(
        '[data-search-title]'
    );
title.focus();
// Variable used to store the select element for genre
const genreSelector =
    document.querySelector(
        '[data-search-genres]'
    );
// Variable used to store the select element for author
const authorSelector =
    document.querySelector(
        '[data-search-authors]'
    );

// Variable used to create a new HTML element for 'authorSelector'
const option =
    document.createElement(
        'option'
    );
// creation of the first option for 'All Authors'
option.setAttribute(
    'value',
    'any'
);
option.innerHTML =
    'All Authors';
authorSelector.appendChild(
    option
);

// Variable used to create a new option for 'genreSelector'
const optionGenres =
    document.createElement(
        'option'
```

```
            );
        // Creation of first option for 'All Genres'
        optionGenres.setAttribute(
            'value',
            'any'
        );
        optionGenres.innerHTML =
            'All Genres';
        genreSelector.appendChild(
            optionGenres
        );

        // Loop through the authors and create the options with the correct
ID's and names
        for (const i in authors) {
            const authorName =
                authors[i];
            const authorId =
                i;
            const option =
                document.createElement(
                    'option'
                );
            option.setAttribute(
                'value',
                authorId
            );
            option.innerHTML =
                authorName;
            authorSelector.appendChild(
                option
            );
        }
        // Loop through genres and create the options with the correct ID's and
names
        for (const i in genres) {
            const genreName =
                genres[i];
```

```
            const genreId =
                i;
            const option =
                document.createElement(
                    'option'
                );
            option.setAttribute(
                'value',
                genreId
            );
            option.innerHTML =
                genreName;
            genreSelector.appendChild(
                option
            );
        }
    };
```

---

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

```
/**
 * This module handles the creation of the search overlay and
populating the selectors with options.
 */

// Separate the logic for creating options into a function
function createOption(value, text) {
    const option = document.createElement('option');
    option.setAttribute('value', value);
    option.innerHTML = text;
    return option;
```

```
}

// Separate the logic for populating selectors into a function
function populateSelector(selector, options) {
    for (const optionData of options) {
        const { value, text } = optionData;
        const option = createOption(value, text);
        selector.appendChild(option);
    }
}

// Function to create and show the search overlay
export function createSearchOverlay() {
    const searchOverlay =
document.querySelector('[data-search-overlay]');
    searchOverlay.toggleAttribute('open');

    const title = document.querySelector('[data-search-title]');
    title.focus();

    const genreSelector =
document.querySelector('[data-search-genres]');
    const authorSelector =
document.querySelector('[data-search-authors]');

    const authorOptions = [
        { value: 'any', text: 'All Authors' },
        // Add more author options as needed
    ];

    const genreOptions = [
        { value: 'any', text: 'All Genres' },
        // Add more genre options as needed
    ];

    populateSelector(authorSelector, authorOptions);
    populateSelector(genreSelector, genreOptions);

    // Call separate functions to populate author and genre options
```

```
    populateSelector(authorSelector, authors.map((authorName,
authorId) => ({ value: authorId, text: authorName })));
    populateSelector(genreSelector, genres.map((genreName, genreId)
=> ({ value: genreId, text: genreName })));
}
```

Key improvements made:

Single Responsibility Principle: The code is broken down into separate functions, each responsible for a single task. createOption handles creating option elements, populateSelector handles populating selectors, and createSearchOverlay handles the overall search overlay creation.

Modularity: The code is organized in a more modular way. The logic for creating options and populating selectors is separated from the main function.

Maintainability: If you need to change how options are created or how selectors are populated, you can modify these separate functions without affecting the main functionality of creating the search overlay.

Readability: By giving meaningful names to functions and using descriptive variable names, the code becomes more readable and easier to understand.

Remember that the code structure and principles can be adapted to your specific use case and requirements.
