DWA_07.4 Knowledge Check_DWA7

---

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

```
//--------------------EVENT LISTENERS-----------------------//
/**
 * Event listener for the settings form submission.
 * Updates the theme based on user selection.
 *
 * @param {Event} event - The event object.
 */
DOM.settings.form.addEventListener('submit', (event) => {
    event.preventDefault()
    const formData = new FormData(event.target)
    const { theme } = Object.fromEntries(formData)

    if (theme === 'night') {
        document.documentElement.style.setProperty('--color-dark', '255, 255, 255');
        document.documentElement.style.setProperty('--color-light', '10, 10, 20');
    } else {
        document.documentElement.style.setProperty('--color-dark', '10, 10, 20');
        document.documentElement.style.setProperty('--color-light', '255, 255, 255');
    }

    document.querySelector('[data-settings-overlay]').open = false
})

//-----------------------------------------------------------------------//
```

This code demonstrates good abstraction by hiding the complexities of theme switching within a well-defined, easy-to-understand function, which makes the codebase more modular, readable, maintainable, and reusable.

```
74  //--------------------------------------------------------------------------//
75  /**
76   * Creates and appends author options to the author select element.
77   * @type {DocumentFragment}
78   */
79  const authorsHtml = document.createDocumentFragment()
80  const firstAuthorElement = document.createElement('option')
81  firstAuthorElement.value = 'any'
82  firstAuthorElement.innerText = 'All Authors'
83  authorsHtml.appendChild(firstAuthorElement)
84
85  for (const [id, name] of Object.entries(authors)) {
86      const element = document.createElement('option')
87      element.value = id
88      element.innerText = name
89      authorsHtml.appendChild(element)
90  }
91
92  DOM.search.authors.appendChild(authorsHtml)
93  //--------------------------------------------------------------------------//
```

The code demonstrates good abstraction by encapsulating the details of creating and appending author options within a clear and focused block, improving modularity, readability, reusability, maintainability, and performance. This abstraction allows other parts of the application to use this functionality without needing to understand or duplicate the underlying DOM manipulation logic.

```
/**
 * Updates the "Show more" button with the number of remaining books and enables/disables the button based on the rem
 */
DOM.list.button.innerText = `Show more (${books.length - BOOKS_PER_PAGE})`
DOM.list.button.disabled = (matches.length - (page * BOOKS_PER_PAGE)) > 0

DOM.list.button.innerHTML = `
    <span>Show more</span>
    <span class="list__remaining"> (${(matches.length - (page * BOOKS_PER_PAGE)) > 0 ? (matches.length - (page * BOOK
`
```

```
 with the number of remaining books and enables/disables the button based on the remaining books.

 more (${books.length - BOOKS_PER_PAGE})`
es.length - (page * BOOKS_PER_PAGE)) > 0



> (${(matches.length - (page * BOOKS_PER_PAGE)) > 0 ? (matches.length - (page * BOOKS_PER_PAGE)) : 0})</span>
```

The code demonstrates good abstraction by encapsulating the logic for updating the "Show more" button within a clear and focused block. This abstraction improves modularity, readability, maintainability, and user experience, allowing other parts of the

application to interact with the button without needing to understand or duplicate the underlying logic.

---

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

```
/**
 * Sets the initial theme based on the user's preferred color scheme.
 * If the preferred color scheme is dark, sets the theme to night; otherwise, sets it to day.
 */
if (window.matchMedia && window.matchMedia('(prefers-color-scheme: dark)').matches) {
    DOM.settings.theme.value = 'night'
    document.documentElement.style.setProperty('--color-dark', '255, 255, 255');
    document.documentElement.style.setProperty('--color-light', '10, 10, 20');
} else {
    DOM.settings.theme.value = 'day'
    document.documentElement.style.setProperty('--color-dark', '10, 10, 20');
    document.documentElement.style.setProperty('--color-light', '255, 255, 255');
}
```

The RGB colors and DOM are being manipulated directly, making future changes difficult. Additionally, the code handles both theme detection and DOM updates together. It would be better to separate these into two different functions.

```
/**
 * Event listener for the search form submission.
 * Filters the book list based on the search criteria and updates the display.
 *
 * @param {Event} event - The event object.
 */
DOM.search.form.addEventListener('submit', (event) => {
    event.preventDefault()
    const formData = new FormData(event.target)
    const filters = Object.fromEntries(formData)
    const result = []

    for (const book of books) {
        let genreMatch = filters.genre === 'any'

        for (const singleGenre of book.genres) {
            if (genreMatch) break;
            if (singleGenre === filters.genre) { genreMatch = true }
        }

        if (
            (filters.title.trim() === '' || book.title.toLowerCase().includes(filters.title.toLowerCase())) &&
            (filters.author === 'any' || book.author === filters.author) &&
            genreMatch
        ) {
            result.push(book)
        }
    }
```

This function is too long, making it difficult to understand and maintain. It handles multiple tasks, which complicates making any future changes.

```
 * Event listener for the "Show more" button.
 * Loads and displays more book previews when the button is clicked.
 */
DOM.list.button.addEventListener('click', () => {
    const fragment = document.createDocumentFragment()

    for (const { author, id, image, title } of matches.slice(page * BOOKS_PER_PAGE, (page + 1) * BOOKS_PER_PAGE)) {
        const element = document.createElement('button')
        element.classList = 'preview'
        element.setAttribute('data-preview', id)

        element.innerHTML = `
            <img
                class="preview__image"
                src="${image}"
            />

            <div class="preview__info">
                <h3 class="preview__title">${title}</h3>
                <div class="preview__author">${authors[author]}</div>
            </div>
        `

        fragment.appendChild(element)
    }

    DOM.list.items.appendChild(fragment)
    page += 1
})
```

Making amendments to the dom inside the eventlistener is going to be very difficult.

---

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

```
/**
 * Sets the initial theme based on the user's preferred color scheme.
 * If the preferred color scheme is dark, sets the theme to night; otherwise, sets it to day.
 */
if (window.matchMedia && window.matchMedia('(prefers-color-scheme: dark)').matches) {
    DOM.settings.theme.value = 'night'
    document.documentElement.style.setProperty('--color-dark', '255, 255, 255');
    document.documentElement.style.setProperty('--color-light', '10, 10, 20');
} else {
    DOM.settings.theme.value = 'day'
    document.documentElement.style.setProperty('--color-dark', '10, 10, 20');
    document.documentElement.style.setProperty('--color-light', '255, 255, 255');
}
```

Sets the theme:
You can create a themeManager class where it encapsulates the theme settings.

```
class ThemeManager {
  constructor(themeElement, styleElement) {
    this.themeElement = themeElement;
    this.styleElement = styleElement;
  }
```

```
    setTheme(theme) {
        if (theme === 'dark') {
            this.themeElement.value = 'night';
            this.styleElement.setProperty('--color-dark', '255, 255, 255');
            this.styleElement.setProperty('--color-light', '10, 10, 20');
        } else {
            this.themeElement.value = 'day';
            this.styleElement.setProperty('--color-dark', '10, 10, 20');
            this.styleElement.setProperty('--color-light', '255, 255, 255');
        }
    }

    applyPreferredTheme() {
        const isDarkMode = window.matchMedia &&
window.matchMedia('(prefers-color-scheme: dark)').matches;
        this.setTheme(isDarkMode ? 'dark' : 'light');
    }
}

const themeManager = new ThemeManager(DOM.settings.theme,
document.documentElement.style);
themeManager.applyPreferredTheme();
```

In the subclasses you will most likely find setTheme & applyTheme.
Breaking the function down for easy readability and when you want to change the code.

Search Function:

```
/**
 * Event listener for the search form submission.
 * Filters the book list based on the search criteria and updates the display.
 *
 * @param {Event} event - The event object.
 */
DOM.search.form.addEventListener('submit', (event) => {
    event.preventDefault()
    const formData = new FormData(event.target)
    const filters = Object.fromEntries(formData)
    const result = []

    for (const book of books) {
        Let genreMatch = filters.genre === 'any'

        for (const singleGenre of book.genres) {
            if (genreMatch) break;
            if (singleGenre === filters.genre) { genreMatch = true }
        }

        if (
            (filters.title.trim() === '' || book.title.toLowerCase().includes(filters.title.toLowerCase())) &&
            (filters.author === 'any' || book.author === filters.author) &&
            genreMatch
        ) {
            result.push(book)
        }
    }
```

You will have a class name of searchBooks with the subclasses of filter & updatePage.

```
class searchBooks {
        constructor ( book, filters ) {
                this.book = book
                this.filters = filters
}
}
```

Show more button:

```
 * Event listener for the "Show more" button.
 * Loads and displays more book previews when the button is clicked.
 */
DOM.list.button.addEventListener('click', () => {
    const fragment = document.createDocumentFragment()

    for (const { author, id, image, title } of matches.slice(page * BOOKS_PER_PAGE, (page + 1) * BOOKS_PER_PAGE)) {
        const element = document.createElement('button')
        element.classList = 'preview'
        element.setAttribute('data-preview', id)

        element.innerHTML = `
            <img
                class="preview__image"
                src="${image}"
            />

            <div class="preview__info">
                <h3 class="preview__title">${title}</h3>
                <div class="preview__author">${authors[author]}</div>
            </div>
        `

        fragment.appendChild(element)
    }

    DOM.list.items.appendChild(fragment)
    page += 1
})
```

Will have a class named bookPreview with subclasses createElement. Can have another class named bookLoader with a subclass named loadMoreBooks.

```
class bookPreview {
        constructor ( author, id, image, title ) {
                this.author = author
                this.id = id
                this.image = image
                this.title = title
}
}
```
