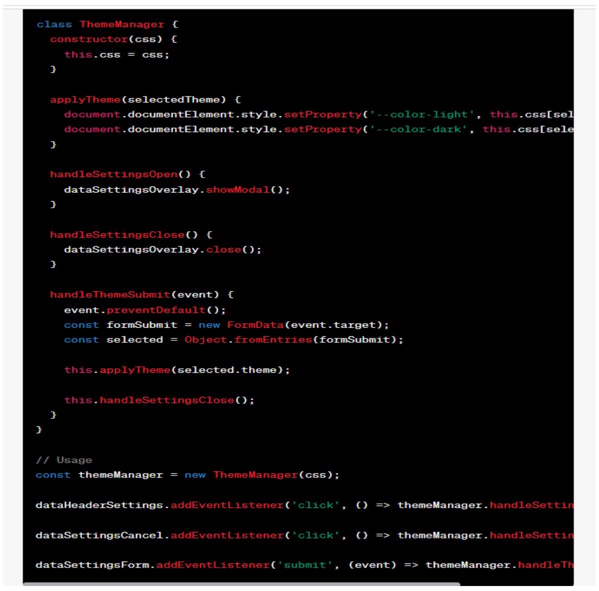
DWA_08 Discussion Questions

In this module you will continue with your “Book Connect” codebase, and further iterate on your abstractions. You will be required to create an encapsulated abstraction of the book preview by means of a single factory function. If you are up for it you can also encapsulate other aspects of the app into their own abstractions.

To prepare for your session with your coach, please answer the following questions. Then download this document as a PDF and include it in the repository with your code.

---

1. What parts of encapsulating your logic were easy?



```
class ThemeManager {
  constructor(css) {
    this.css = css;
  }

  applyTheme(selectedTheme) {
    document.documentElement.style.setProperty('--color-light', this.css[sel
    document.documentElement.style.setProperty('--color-dark', this.css[sele
  }

  handleSettingsOpen() {
    dataSettingsOverlay.showModal();
  }

  handleSettingsClose() {
    dataSettingsOverlay.close();
  }

  handleThemeSubmit(event) {
    event.preventDefault();
    const formSubmit = new FormData(event.target);
    const selected = Object.fromEntries(formSubmit);

    this.applyTheme(selected.theme);

    this.handleSettingsClose();
  }
}

// Usage
const themeManager = new ThemeManager(css);

dataHeaderSettings.addEventListener('click', () => themeManager.handleSettin

dataSettingsCancel.addEventListener('click', () => themeManager.handleSettin

dataSettingsForm.addEventListener('submit', (event) => themeManager.handleTh
```

In this example above, I've encapsulated the theme-related logic in a ThemeManager class. This class has methods for applying themes, handling settings open and close events, and handling theme submissions. By encapsulating these related functions within a class, you create a clean separation of concerns and encapsulate the theme management logic into a single unit. Benefits of this approach include better organization of your code, easier maintenance, and the ability to reuse this logic in different parts of your application if needed

---

2. What parts of encapsulating your logic were hard?

For me encapsulating logic I think it can offer numerous benefits, but there are also challenges and considerations you need to address, like;

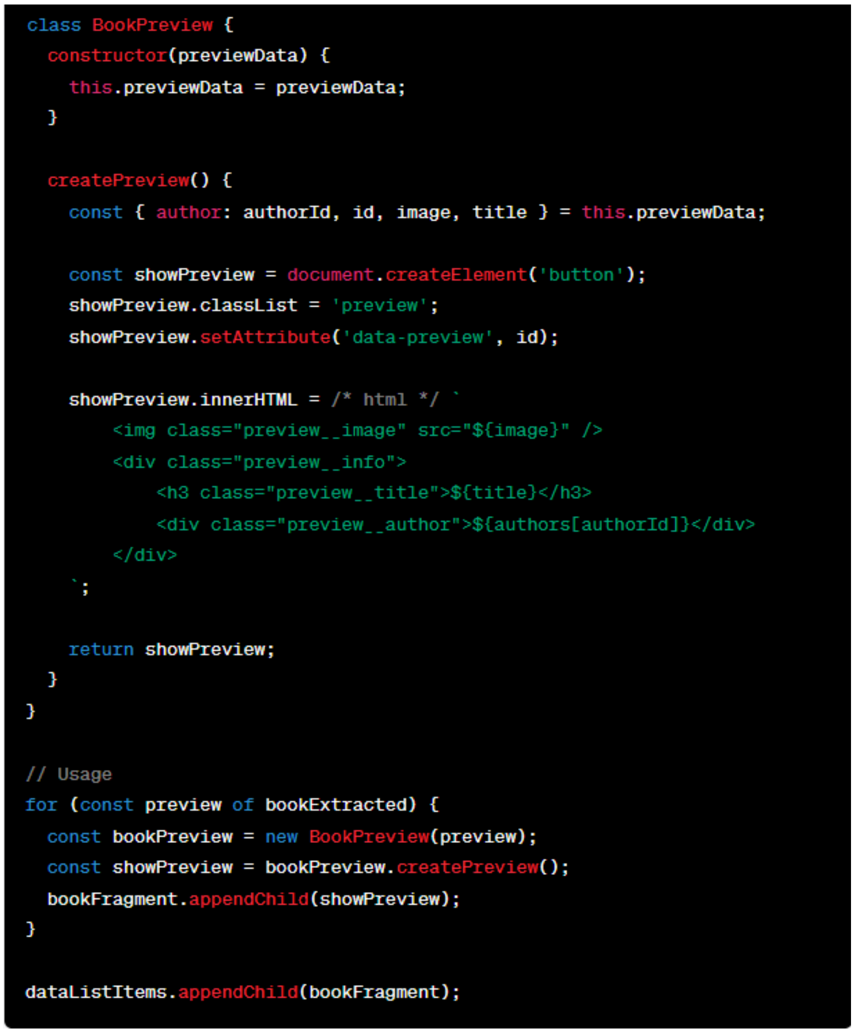
Identifying Abstractions: Deciding how to group related functionality into meaningful abstractions can be challenging. You need to carefully analyze your code and identify logical units that can be encapsulated.

Despite these challenges, encapsulation remains a valuable approach for creating maintainable, modular, and organized code. Addressing these challenges through careful design, clear documentation, and collaboration with your team can help you fully leverage the benefits of encapsulation while minimizing its downsides.

---

3. Is abstracting the book preview a good or bad idea? Why?

Abstracting the book preview functionality is a good idea in certain scenarios, but it also depends on the complexity and requirements of the project because abstraction is beneficial when it leads to cleaner, more modular, and maintainable code. However, it's essential to consider the specific needs of your project. If the book preview functionality is a crucial and complex part of your application, abstracting it could be a good idea. On the other hand, for smaller projects or simpler functionality, it might be more efficient to keep the code closer to where it's used. Always weigh the benefits against the potential downsides and choose an approach that best fits your project's requirements.



```javascript
class BookPreview {
  constructor(previewData) {
    this.previewData = previewData;
  }

  createPreview() {
    const { author: authorId, id, image, title } = this.previewData;

    const showPreview = document.createElement('button');
    showPreview.classList = 'preview';
    showPreview.setAttribute('data-preview', id);

    showPreview.innerHTML = /* html */ `
        <img class="preview__image" src="${image}" />
        <div class="preview__info">
            <h3 class="preview__title">${title}</h3>
            <div class="preview__author">${authors[authorId]}</div>
        </div>
    `;

    return showPreview;
  }
}

// Usage
for (const preview of bookExtracted) {
  const bookPreview = new BookPreview(preview);
  const showPreview = bookPreview.createPreview();
  bookFragment.appendChild(showPreview);
}

dataListItems.appendChild(bookFragment);
```

The BookPreview class encapsulates the creation of book previews. The constructor takes the previewData object as a parameter, which contains the necessary information to create the preview. The createPreview method generates the preview HTML structure and returns a DOM element.