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?

Initially the createPreview function was quite easy to encapsulate as it for the most part needed to be contained in a factory function with a caller in order to abstract the functionality away. This also meant that I could use the functionality by passing it different objects each time by calling the pageLoader function.

```
/**
 * This function creates the html element that displays the basic information of a book
 * as a button and returns that element.
 *
 * @param {string} author - author of the book as an id string
 * @param {string} id - the id of the book as an id string
 * @param {string} image - the cover image as a link string
 * @param {string} title - the title of the book as a string
 * @returns {HTMLElement} - the button element
 */
const createPreview = ({ author, id, image, title }) => {
  const element = document.createElement("button");
  element.classList = "preview";
  element.setAttribute("data-preview", id);

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

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

---

2. What parts of encapsulating your logic were hard?
I found that further encapsulating my logic for the pageLoader function was clunky but serviceable. It had one of the 2 required bits to make it a factory function in that it was passed an object but was notably lacking the return feature as it simply manipulated the DOM.

We could thus make a higher order function which simply called the pageLoader function and passed it the path it needed to append to and the slice of books or media to be displayed. This would mean that we need to modify our data structure to accommodate for all the extra bits of information to be passed to the function and wrap it neatly in for instance the extracted object.

It kind of thus showed me that my modules and functions were not as independent as I hoped and thought and that maybe the abstraction and way I worked with SOLID beforehand could have applied more to these functions.

```
/**
 * This is a caller function that calls createPreview
 * with the amount of books calculated by the EXTRACTED variable
 * and appends the returned elements to our DOM
 *
 * @param {Array} EXTRACTED - An array of objects representing items to load.
 * @param {string} EXTRACTED.author - The author of the item.
 * @param {string} EXTRACTED.image - The image URL for the item.
 * @param {string} EXTRACTED.title - The title of the item.
 * @param {string} EXTRACTED.id - The unique identifier of the item.
 * @returns {void} - This function only modifies the DOM
 */
export const pageLoader = (EXTRACTED) => {
  const fragment = document.createDocumentFragment();

  for (const { author, image, title, id } of EXTRACTED) {
    const preview = createPreview({
      author,
      id,
      image,
      title,
    });

    fragment.appendChild(preview);
  }

  htmlSelector.list.items.appendChild(fragment);
};
```

Above the function which I had difficulty with. Below a loose interpretation of how we could modify the function.

```
export const createPageGenerator = (createPreview, targetAttributes) ⇒ {

const {htmlPath, extracted} = targetAttributes

return const pageLoader ⇒ () {
    const fragment = document.createDocumentFragment();

      for (const { iterableItems } of extracted) {

          const preview = createPreview({ ...iterableItems});

          FRAGMENT.appendChild(preview);
      };

    htmlPath.appendChild(FRAGMENT);

  };
};

createPageGenerator(createPreview, setBooks)
```

---

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

With my difficulties expressed above I believe it is also good to look at why we would need to abstract to such a micro level in this program. I think that funny enough it was difficult to change these functions due to the fact that the program itself is fairly limited by the data it is displaying and wouldn't need a function that creates functions which load the page as in theory we only need to load it a few times and don't need these almost save states of previous previews to be stored or created.

So perhaps for the level of abstracting down to the preview function is a good idea but beyond that for the scope of this project I find it hard to imagine where more we would need factory functions. If however the data structure changed or we needed to represent say a page of movies or music albums or even artworks that more factory functions would come in handy allowing us to not necessarily hardcode the values into our page generation module but instead pass little objects which dictate what is on each page as it's generated.
