DWA_07.4 Knowledge Check_DWA7

---

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

```
/**
 * Creates a book preview element with given details.
 *
 * @param {PreviewData} data - The book data that is needed to create the preview item.
 * @returns {HTMLElement} The created preview element.
 */
const createPreview = (data) => {
  const { author, id, image, title } = data;
  const element = document.createElement("div");
  element.className = "preview";
  element.dataset.id = id;

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

          <div class="°°°">
            <h3 class="°°°">${title}</h3>

            <p class="°°°">${author}</p>
          </div>
      `;

  return element;
};
```

- The `createPreview` function abstracts the creation of a book preview element..
- By encapsulating the logic for creating the preview element, the function abstracts away the implementation details from the caller.
- The `createPreview` function has a clear responsibility: creating a book preview element. It doesn't handle other unrelated tasks. This adherence to the Single Responsibility Principle ensures that the function is focused and maintainable.

```
/**
 * Updates the "data-list-active" overlay element's HTML, using the data of the given book, so that the overlay
 * correctly displays the details of the selected book.
 *
 * @param {OverLayBookData} book - The book object.
 */
export const loadBookOverlayData = (book) => {
  const { list } = documentHtml;
  const publishedDate = new Date(book.published);

  list["overlay-image"].setAttribute("src", book.image);
  list["overlay-blur"].setAttribute("src", book.image);
  list["overlay-title"].innerText = `${book.title}`;
  list["overlay-subtitle"].innerText = `${
    authors[book.author]
  } (${publishedDate.getFullYear()})`;
  list["overlay-description"].innerText = `${book.description}`;
};
```

- Many of the same reasons from the above example apply to this method as well.

```
 * @param {Event} event - The event object.
 */
const handleShowMore = (event) => {
  const newLength = state["books-per-page"] * 2;
  const prevLength = state["books-per-page"];

  state["books-per-page"] =
    newLength > state.matches.length ? state.matches.length : newLength;
  state["extracted-books"] =
    state["books-per-page"] === state.matches.length
      ? state.matches
      : state.matches.slice(0, state["books-per-page"]);

  const itemsToLoad = state["extracted-books"].slice(
    prevLength,
    state["books-per-page"]
  );

  loadListItems(itemsToLoad);
  updateShowMoreBtn(state.matches.length - state["books-per-page"]);
};
```

- The function's responsibility is clear: managing the "Show More" action. It doesn't handle unrelated tasks, adhering to the Single Responsibility Principle.
- The function's signature (`handleShowMore(event: Event)`) adheres to the Interface Segregation Principle.

- It only exposes the necessary input (`event`) without imposing unnecessary requirements on the caller.

---

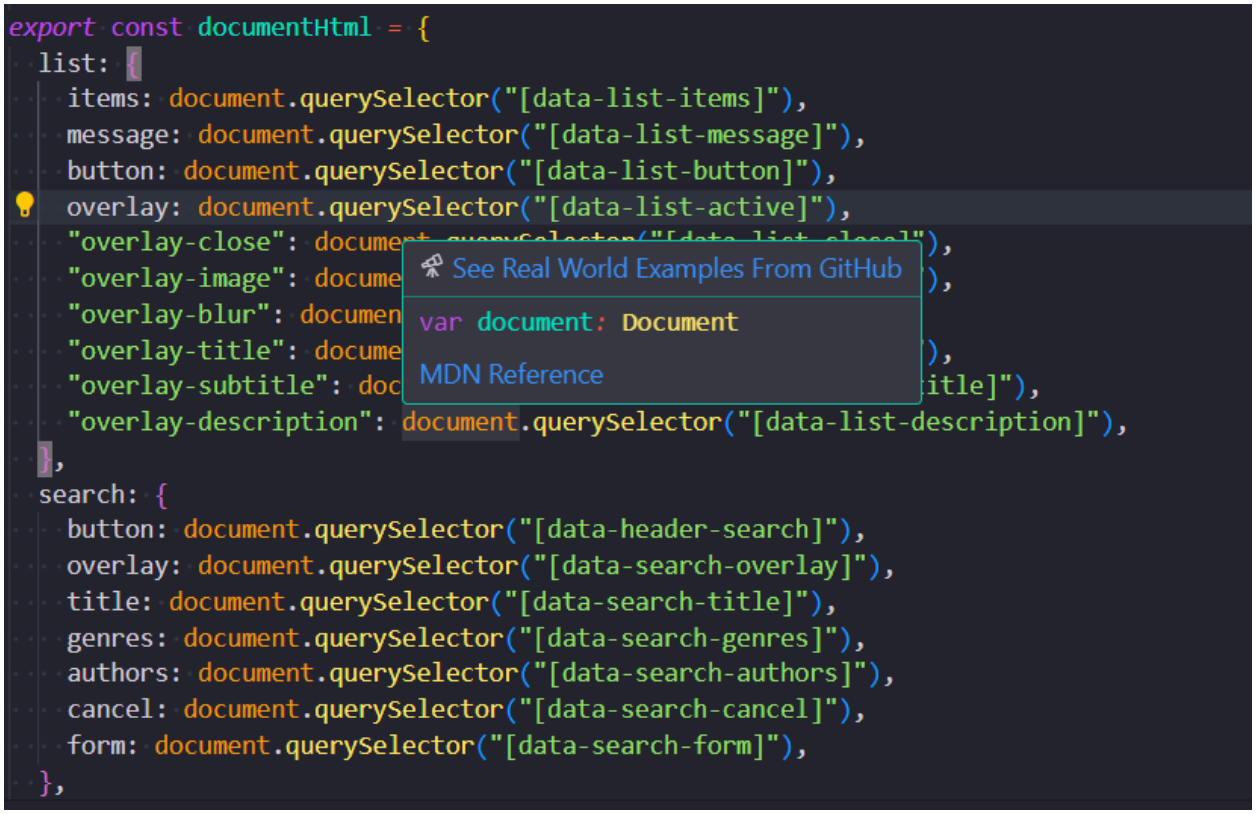
2. Which were the three worst abstractions, and why?

```
 * inner text is the name of the genre. Then add the created genre option to the genres dropdown.
 */
Object.entries(genres).forEach(([id, name]) => {
  const genreOption = document.createElement("option");
  genreOption.value = id;
  genreOption.innerText = name;
  search.genres.appendChild(genreOption);
});

// Create a new option element for the authors dropdown, set its value and inner text.
const allAuthorsOption = document.createElement("option");
allAuthorsOption.value = "any";
allAuthorsOption.innerText = "All Authors";
// Add the "All Authors" option to the authors dropdown
search.authors.appendChild(allAuthorsOption);

/**
 * Loop through the predefined list of authors in the "data.js" file.
 * For each author, create a new option element whose value is the ID of the author and whose
 * inner text is the name of the auhtor. Then add the created author option to the authors dropdown
 */
Object.entries(authors).forEach(([id, name]) => {
  const authorOption = document.createElement("option");
  authorOption.value = id;
  authorOption.innerText = name;
  search.authors.appendChild(authorOption);
});
```

- This code can be rewritten into a method that dynamically performs the action of adding options to any dropdown.
- This would avoid unnecessary duplication of code and make it more maintainable and reusable.



- It is not a terrible display of abstraction, however it is lacking in some aspects.

---

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

- **(S)** The function now has a clear responsibility: creating and appending option elements to a filter dropdown. It works generically for any filter type.
- **(O)** The function is open for extension because you can easily add new filter types without modifying the existing code.
- **(I)** The function provides a clear interface: three parameters (`defaultOptionText`, `filterKey`, and `optionsList`). It doesn't force clients to depend on unnecessary methods or properties.

```
const getElementByDataAttribute = (attribute) => {
  const element = document.querySelector(`[data-${attribute}]`);

  if (!(element instanceof HTMLElement)) {
    document.body.innerText =
      "An unexpected error has occurred. Please refresh the page.";

    throw new Error(
      `Element with attribute 'data-${attribute}' could not be found.`
    );
  }

  return element;
};

/**
 * An object literal that contains references to all the HTML elements
 * referenced through the operation of the app either upon initialisation or
 * while its running (via event listeners). This arrangement allows for a structured
 * view and access to all user interface elements.
 */
export const documentHtml = {
  list: {
    items: getElementByDataAttribute("list-items"),
    message: getElementByDataAttribute("list-message"),
```

- The introduction of a new method called `getElementByDataAttribute` encapsulates the logic for retrieving an `HTMLElement` based on a given data attribute.
- This method abstracts away the details of querying the DOM and error handling, making the code more modular and easier to understand.
