DWA_07.4 Knowledge Check_DWA7

---

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

```
/**
 * An object literal that contains references to all the HTML elements
 * referenced through the operation of the app either upon initialization or
 * while its running (via event listeners). This ensure that all UI elements can
 * be accessed and seen in a structured manner in a single data structure.
 */
export const html = {
  header: {
    search: document.querySelector("[data-header-search]"),
    settings: document.querySelector("[data-header-settings]"),
  },
  list: {
    items: document.querySelector("[data-list-items]"),
    message: document.querySelector("[data-list-message]"),
    button: document.querySelector("[data-list-button]"),
    active: document.querySelector("[data-list-active]"),
    blur: document.querySelector("[data-list-blur]"),
    image: document.querySelector("[data-list-image]"),
    title: document.querySelector("[data-list-title]"),
    subtitle: document.querySelector("[data-list-subtitle]"),
    description: document.querySelector("[data-list-description]"),
    close: document.querySelector("[data-list-close]"),
  },
  search: {
    overlay: document.querySelector("[data-search-overlay]"),
    form: document.querySelector("[data-search-form]"),
    title: document.querySelector("[data-search-title]"),
    genres: document.querySelector("[data-search-genres]"),
    authors: document.querySelector("[data-search-authors]"),
    cancel: document.querySelector("[data-search-cancel]"),
  },
  settings: {
    overlay: document.querySelector("[data-settings-overlay]"),
    form: document.querySelector("[data-settings-form]"),
    theme: document.querySelector("[data-settings-theme]"),
    cancel: document.querySelector("[data-settings-cancel]"),
  },
};
```

By creating a separate file containing html elements which are needed, when they are used in the main scripts.js file, it is much easier for developers to find and maintain code they are looking for

```
for (const book of matches.slice(0, BOOKS_PER_PAGE)) {
  const bookPreview = createBookPreview(book);
  starting.appendChild(bookPreview);
}

function createBookPreview({ author, id, image, title }) {
  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>
  `;

  return element;
}
```

By encapsulating the logic for creating a book preview element, you make the code more modular and easier to maintain

```
function handleListItemClick(event) {
  const pathArray = Array.from(event.path || event.composedPath());
  let active = null;

  for (const node of pathArray) {
    if (active) break;

    if (node?.dataset?.preview) {
      active = findActiveBook(node.dataset.preview);
    }
  }

  if (active) {
    updateHTML(active);
  }
}

function findActiveBook(previewId) {
  for (const singleBook of books) {
    if (singleBook.id === previewId) {
      return singleBook;
    }
  }
  return null;
}

function updateHTML(activeBook) {
  html.list.active.open = true;
  html.list.blur.src = activeBook.image;
  html.list.image.src = activeBook.image;
  html.list.title.innerText = activeBook.title;
  html.list.subtitle.innerText = `${authors[activeBook.author]} (${new Date(activeBook.published).getFullYear()})`;
  html.list.description.innerText = activeBook.description;
}

html.list.items.addEventListener("click", handleListItemClick);
```

The logic for handling the click event is encapsulated within the handleListItemClick function which makes the code more maintainable by splitting it into focused functions

---

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

```
function findActiveBook(previewId) {
  for (const singleBook of books) {
    if (singleBook.id === previewId) {
      return singleBook;
    }
  }
  return null;
}
```

This function doesn't adhere to the Single Responsibility Principle, as it has two responsibilities in a single function.

```
function filterBooks(formData) {
  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);
    }
  }

  return result;
}
```

This function also violates the Single Responsibility Principle(SRP).

```
function updateList(result) {
  page = 1;
  matches = result;

  if (result.length < 1) {
    html.list.message.classList.add("list__message_show");
  } else {
    html.list.message.classList.remove("list__message_show");
  }

  html.list.items.innerHTML = "";
  const newItems = document.createDocumentFragment();

  for (const { author, id, image, title } of result.slice(0, 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>
    `;

    newItems.appendChild(element);
  }

  html.list.items.appendChild(newItems);
  html.list.button.disabled = matches.length - page * BOOKS_PER_PAGE < 1;

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

  window.scrollTo({ top: 0, behavior: "smooth" });
  html.search.overlay.open = false;
}
```

Relating to the Single Responsibility Principle(SRP) there is still some coupling in the updateList function

---

3. How can The three worst abstractions be improved via SOLID principles.
    1. These responsibilities could be separated into 2 different functions to adhere to the SRP
    2. By adding a separate function for the building of the result array
    3. By adding further abstraction between the filterBooks and updateList functions, by adding a updateListDisplay function, so that if the filtering logic is changed in the future updateList will not be affected

---