DWA_07.4 Knowledge Check_DWA7

---

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

```
/**
 * This function populates the options for the search function with either the genres or authors
 * and modifies the DOM elements that correspond to those categories
 *
 * @param {object} dataObject - object containing either author info or genre info
 * @param {string} dataObject.innerText - search option for everything in the category as a string
 * @param {string} dataObject.objectRepresentation - type of object being passed as a string
 * @param {object} dataObject.data - ID and name of authors or genres
 * @returns {void} - This function only modifies the DOM
 */
const createAllOptions = (dataObject) => {
  const { innerText, objectRepresentation, data } = dataObject;
  const optionsElement = document.createDocumentFragment();
  let element = document.createElement("option");
  element.value = "any";
  element.innerText = innerText; // this adds an option called all genres
  optionsElement.appendChild(element);

  for (const [id, name] of Object.entries(data)) {
    // another for loop choice that populates the select element options
    element = document.createElement("option");
    element.value = id;
    element.innerText = name;
    optionsElement.appendChild(element);
  }

  htmlSelector.search[objectRepresentation].appendChild(optionsElement); // adds all options to the selec
};
```

This function has a single responsibility in that it populates a field with elements.
It can also be passed different objects in dataObject to extend the use of the function.
It also has no hard coded values that it checks for instead it relies on abstracted information to be passed to it.

:. This function implements SRP, OCP and DIP.

```
/**
 * 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;
};
```

This function has a single responsibility in that it creates an element.
It can also be passed different objects as arguments to extend the use of the function.
:. This function implements SRP & OCP.

The last example is not a single piece of code adhering to the SOLID principle but instead the creation of a module as it takes away the unnecessary view of an entire function (the handleDescriptionFunction) and abstracts it to its event listeners meaning that all the behind the scenes page generation is handle by a ‘private’ function.

---

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

The handleSearchSubmit function, The View.js module and I don’t necessarily have more applicable abstractions that adhere to SOLID.

```
export const handleSearchSubmit = (event) => {
  event.preventDefault();
  const formData = new FormData(event.target);
  const filters = Object.fromEntries(formData);
  const result = [];

  for (const singleBook of books) {
    const titleMatch =
      filters.title.trim() === "" ||
      singleBook.title.toLowerCase().includes(filters.title.toLowerCase());
    const authorMatch =
      filters.author === "any" || singleBook.author === filters.author;

    let genreMatch = filters.genre === "any";

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

    if (titleMatch && authorMatch && genreMatch) result.push(singleBook);
  }

  if (result.length < 1) {
    htmlSelector.list.message.classList.add("list__message_show");
    htmlSelector.search.overlay.open = false;
  } else {
    htmlSelector.list.message.classList.remove("list__message_show");
  }

  MATCHES = result;
  PAGE = 0;
  htmlSelector.list.items.innerHTML = "";
  const extracted = MATCHES.slice(
    PAGE * BOOKS_PER_PAGE,
    (PAGE + 1) * BOOKS_PER_PAGE
  );
  pageLoader(extracted);
  showMoreButton();

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

---

3. How can The three worst abstractions be improved via SOLID principles.
In the case of handleSearchSubmit the function can be improved with the SOLID principle by:
- Splitting the function into multiple smaller functions such as book filtering, element creation and element appending (the SRP method)
- Pass better arguments to the function so that it can search multiple databases given different parameters. This could be done by passing an object that has the user search criteria stored in it and then validators to check that these criteria match another set of criteria given the database to check. (OCP)
-Due to the manner the function is coded its ISP has no modularity and thus the function cannot have multiple segregated interfaces and muddles many interfaces together. (ISP)
-Lastly the function relies heavily on external variables such as books and matches as well as the htmlSelector. We could improve this by instead passing it these variables and thus replace what elements it is able to generate allowing for dependency injection thus applying more to the DIP.

In the case of the View.js module we have some loosely related functions scattered in a helpers module. We could instead modularise each function with its own set of independent variables to pass to the function which would increase modularity and allow for a better OCP as each function can be targeted without having issues with global scope pollution. It would also mean that each module adheres closer to the SRP.

Lastly there are minor improvements that can be made to some of the better examples of abstraction:

For instance the createAllOptions function could outsource the element creation functionality to adhere better to the SRP as it now can handle more generic element creation.

With the createPreview function we could do the same thing by abstracting the element creation to enhance SRP adherence and possibly even pass a single object as a parameter to improve the modularity of the function and adhering to OCP in a stricter manner.