DWA_07.4 Knowledge Check_DWA7

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

#1 The code below only updates the book count every time the showMore button is clicked, so it meets the Single-Responsibility principle(SRP)

```
/**
 * This function updates the number of books remaining when the showMore button
 * is clicked.remaining books decrements by 36
 */
const updateRemainingBooksCount = () => {
    let remainingBooks = books.length - (page * BOOKS_PER_PAGE);
    const listButtonElement = document.querySelector('[data-list-button]');

    listButtonElement.innerText = `Show more (${remainingBooks})`;
    listButtonElement.enabled = remainingBooks > 0;

    listButtonElement.innerHTML = `
      <span>Show more</span>
      <span class="list__remaining"> (${remainingBooks > 0 ? remainingBooks : 0})</span>
    `;

    document.querySelector('[data-list-button]').addEventListener('click', () => {

        remainingBooks -= 36;
        updateRemainingBooksCount();
      });

  }

  updateRemainingBooksCount();
```

#2 The two functions below meet the SRP and OCP because the first one creates a fragment +36 books everytime the button is clicked and the activeBook function only shows the description of the clicked books ,they are also open for extension but closed for modification.

```
dataListButton.addEventListener('click', () => {
    const fragment = document.createDocumentFragment();

    const booksPerPage = matches.slice(page * BOOKS_PER_PAGE, (page + 1) * BOOKS_PER_PAGE);
    for (const book of booksPerPage) {
        const preview = createBookPreview(book);
        fragment.appendChild(preview);
    }

    document.querySelector('[data-list-items]').appendChild(fragment);
    page += 1;
});


/**
 * This function handles the logic of retrieving the active book based on the clicked element.
 * @param {event} event
 * @returns {element}
 */

const getActiveBook = (event) => {
    const pathArray = Array.from(event.path || event.composedPath());

    for (const node of pathArray) {
        if (node?.dataset?.preview) {
            return books.find((singleBook) => singleBook.id === node.dataset.preview) || null;
        }
    }

    return null;
}
```

#3 The code below meets both SRP & OCP standards because it only has a single responsibility and it is also open for extension but closed for modification

```
/**
 * This is the message that will be displayed if there are no matching books
 */
const errorMessage = () => {
  const listMessage = document.querySelector('[data-list-message]');
  if (result.length < 1) {
    listMessage.classList.add('list__message_show');
  } else {
    listMessage.classList.remove('list__message_show');
  }
};

errorMessage();
```

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

#1 The initializeOverlayFunctionality function handles multiple responsibilities, such as setting up event listeners, opening and closing overlays, and focusing elements. This violates the SRP, as the function should have a single responsibility.

```
const initializeOverlayFunctionality = () => {
    const searchOverlay = document.querySelector('[data-search-overlay]');
    const settingsOverlay = document.querySelector('[data-settings-overlay]');
    const searchCancel = document.querySelector('[data-search-cancel]');
    const settingsCancel = document.querySelector('[data-settings-cancel]');
    const headerSearch = document.querySelector('[data-header-search]');
    const headerSettings = document.querySelector('[data-header-settings]');
    const listClose = document.querySelector('[data-list-close]');
    const searchTitle = document.querySelector('[data-search-title]');
    const listActive = document.querySelector('[data-list-active]');

    searchCancel.addEventListener('click', () => {
      closeOverlay(searchOverlay);
    });

    settingsCancel.addEventListener('click', () => {
      closeOverlay(settingsOverlay);
    });

    headerSearch.addEventListener('click', () => {
      openOverlay(searchOverlay);
      focusElement(searchTitle);
    });

    headerSettings.addEventListener('click', () => {
      openOverlay(settingsOverlay);
    });

    listClose.addEventListener('click', () => {
      closeOverlay(listActive);
    });
  /**
   * This function opens overlays
   * @param {overlay} overlay
   */
    function openOverlay(overlay) {
      overlay.open = true;
```

#2 The code below directly accesses and manipulates DOM elements (listActiveOverlay, listActiveBlurImage, etc.) within the event handler. This tightly couples the code to the specific DOM structure, violating the DIP.

```
290  const listActiveOverlay =  document.querySelector('[data-list-active]')
291  const listActiveBlurImage = document.querySelector('[data-list-blur]')
292  const listActiveImage = document.querySelector('[data-list-image]')
293  const titleOfBook = document.querySelector('[data-list-title]')   // 2 (event listerners)
294  const subtitleOfBook = document.querySelector('[data-list-subtitle]')
295  const descriptionOfBook = document.querySelector('[data-list-description]')
296  const dataListItems = document.querySelector('[data-list-items]')
297
298  dataListItems.addEventListener('click', (event) => {
299      /**
300       * Checks if element (book) is active / clicked then it displays the clicked book information
301       */
302      const active = getActiveBook(event);
303
304      if (active) {
305          listActiveOverlay.open = true;
306           listActiveBlurImage.src = active.image;
307           listActiveImage.src = active.image;
308           titleOfBook.innerText = active.title;
309           subtitleOfBook.innerText = `${authors[active.author]} (${new Date(active.published).getFullYear()})`;
310           descriptionOfBook.innerText = active.description;
311      }
312  });
```

#3 The code below does two things ,it renders the book preview and also generates the the preview elements which violates the SRP.

```
const renderBookPreviews = () => {
    // Get the book previews for the current page
    const bookPreviews = matches
      .slice((page - 1) * BOOKS_PER_PAGE, page * BOOKS_PER_PAGE)
      .map(createPreviewElement);

    // Append each preview element to the 'starting' container
    bookPreviews.forEach((preview) => {
      starting.appendChild(preview);
    });

    // Append the 'starting' container to the 'data-list-items' container
    document.querySelector('[data-list-items]').appendChild(starting);
  }

  // Call the renderBookPreviews function to display the book previews
  renderBookPreviews();

  const genreHtml = createOptionsFragment(genres, 'All Genres');
  document.querySelector('[data-search-genres]').appendChild(genreHtml);

  const authorsHtml = createOptionsFragment(authors, 'All Authors');
  document.querySelector('[data-search-authors]').appendChild(authorsHtml);
```

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

- The first one can be refactored into smaller functions, each with a clear and single responsibility.
- The second, To adhere to the SRP principle, I can consider introducing abstractions or interfaces to decouple the code from specific DOM elements and rely on dependency injection.
- The third code can be split into to functions