DWA_07.4 Knowledge Check_DWA7

---

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

1. **Applying functions within a function:** Allows for organizing, modularizing, and making code easier to understand. It abstracts away the details of checking the color scheme and calling the necessary functions to set the theme and theme colors.

   For example; I've set separate functions for each preferred color scheme within a function called "**setThemeBasedOnColorScheme**"

```
const themeManager = {
    setThemeBasedOnColorScheme() {
      const settingsTheme = document.querySelector('[data-settings-theme]');
      const prefersDarkMode = window.matchMedia && window.matchMedia('(prefers-color-scheme: dark)').matches;

      if (prefersDarkMode) {
        this.setTheme('night');
        this.setThemeColors('255, 255, 255', '10, 10, 20');
      } else {
        this.setTheme('day');
        this.setThemeColors('10, 10, 20', '255, 255, 255');
      }
    },

    setTheme(theme) {
      document.querySelector('[data-settings-theme]').value = theme;
    },

    setThemeColors(colorDark, colorLight) {
      document.documentElement.style.setProperty('--color-dark', colorDark);
      document.documentElement.style.setProperty('--color-light', colorLight);
    },
  };

  // Call the method of the themeManager object to set the theme based on the color scheme.
  themeManager.setThemeBasedOnColorScheme();
```

2. **Adding an object as an abstraction:** where you can organize related code into the same local scope. This improves code readability, and separate concerns. It also allows you to reuse the object's methods throughout your codebase, making it easier to maintain and modify the theme-related functionality.

---

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

1. **The Obscuring Abstraction:** This abstraction hides critical details that are difficult to understand or debug. It acts like a black box, making it incredibly frustrating to pinpoint and fix issues within the code.

2. **The Undocumented Abstraction:** This abstraction lacks essential documentation or explanations. It's like being handed a tool without instructions – you might guess at its purpose, but you'll struggle to use it effectively or maintain it over time.

3. **The Poorly Named Abstraction:** This abstraction suffers from unclear or misleading names and inadequate documentation. This makes it hard to decipher the abstraction's intended function and behavior, leading to confusion, misuse, and challenges when collaborating with other developers.

---

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

1. **Interface Segregation Principle (ISP)**: Design interfaces that expose only the necessary and relevant information to clients, hiding implementation details that are not needed. This promotes a clearer separation between the client and the implementation.

2. **Single Responsibility Principle (SRP**): By applying the SRP, each abstraction will have a clear and well-defined purpose. This clarity allows for more accurate and descriptive naming. When an abstraction has a single responsibility, it becomes easier to choose meaningful names that accurately represent its purpose and behavior.
