



You have 2 free member-only stories left this month. [Sign up for Medium](#) and get an extra one

 **Mohammad Faisal**
Apr 20, 2021 · 4 min read · 

Applying the Liskov Substitution Principle in React

A look at the SOLID principles in action

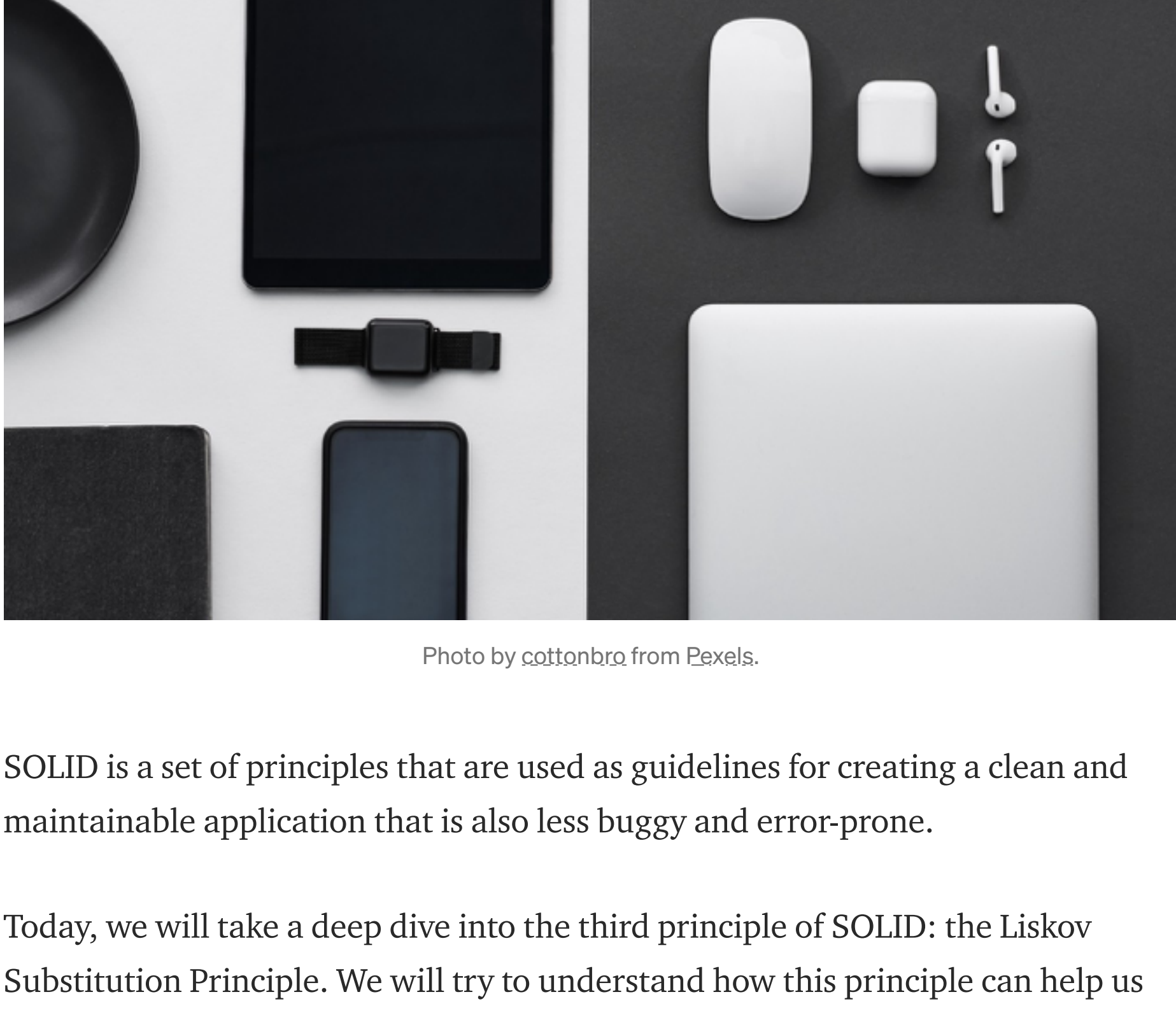


Photo by cattanbra from Pixels.

SOLID is a set of principles that are used as guidelines for creating a clean and maintainable application that is also less buggy and error-prone.

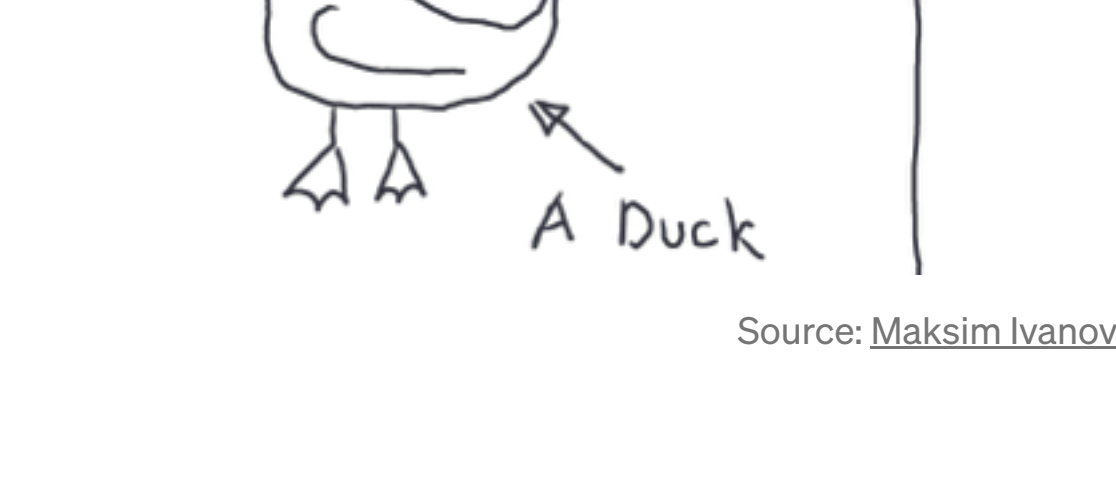
Today, we will take a deep dive into the third principle of SOLID: the Liskov Substitution Principle. We will try to understand how this principle can help us to create a better and cleaner React application.

Other Articles in this Series

1. [Single Responsibility Principle](#)
2. [Open Closed Principle](#)
3. [Interface Segregation Principle](#)
4. [Dependency Inversion Principle](#)

Example

If `PlasticDuck` is a subclass of `Duck`, then we should be able to replace instances of `Duck` with `PlasticDuck` without any surprises.



Source: Maksim Ivanov

That means `PlasticDuck` should fulfill all the expectations set by the `Duck` class.

What Does This Mean in React?

React is not an object-oriented framework because it's basically JavaScript. In the context of React, the main idea behind this principle is:

“Components should abide by some kind of contract.”

At its core, this means there should be some kind of contract between components. So whenever a component uses another component, it shouldn't break its functionality (or create any surprises).

Let's Take a Deeper Dive

Let's take a `ModalHolder` component. This component takes `contentToShow` as a prop and shows it inside a modal:

```
1 import {useState} from 'react';
2 import Modal from 'react-modal';
3
4 export const ModalHolder = ({contentToShow}) => {
5
6   const [visibility, setVisibility] = useState(false);
7
8   return <>
9     <button onClick={() => setVisibility(true)}> Show Modal</button>
10
11     <Modal isOpen={visibility}>
12       <div>{contentToShow}</div>
13     </Modal>
14   </>
15 }
```

ModalHolder.jsx hosted with ♥ by GitHub [view raw](#)

What's the issue here?

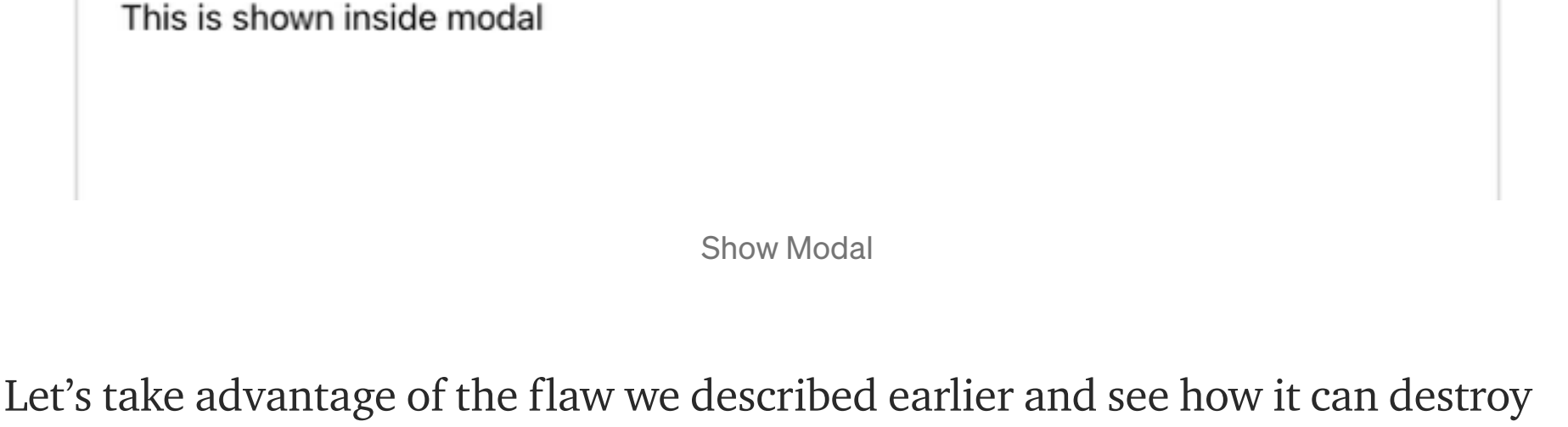
Well, the problem is now there are no restrictions on what can be passed into the `ModalHolder` component. Absolutely anything can be passed into this through the variable `contentToShow`.

First, let's check if our code works and everything goes as expected:

```
1 import React, {useEffect} from 'react';
2 import {ModalHolder} from '../views/liskov-substitution-principle/ModalHolder';
3
4 function App() {
5
6   const modalContent = <div> This is shown inside modal </div>
7
8   return (
9     <div>
10       <ModalHolder contentToShow = {modalContent} />
11     </div>
12   );
13 }
14
15 export default App;
```

App.jsx hosted with ♥ by GitHub [view raw](#)

Now if you open the modal, it will work just fine and show you the modal:



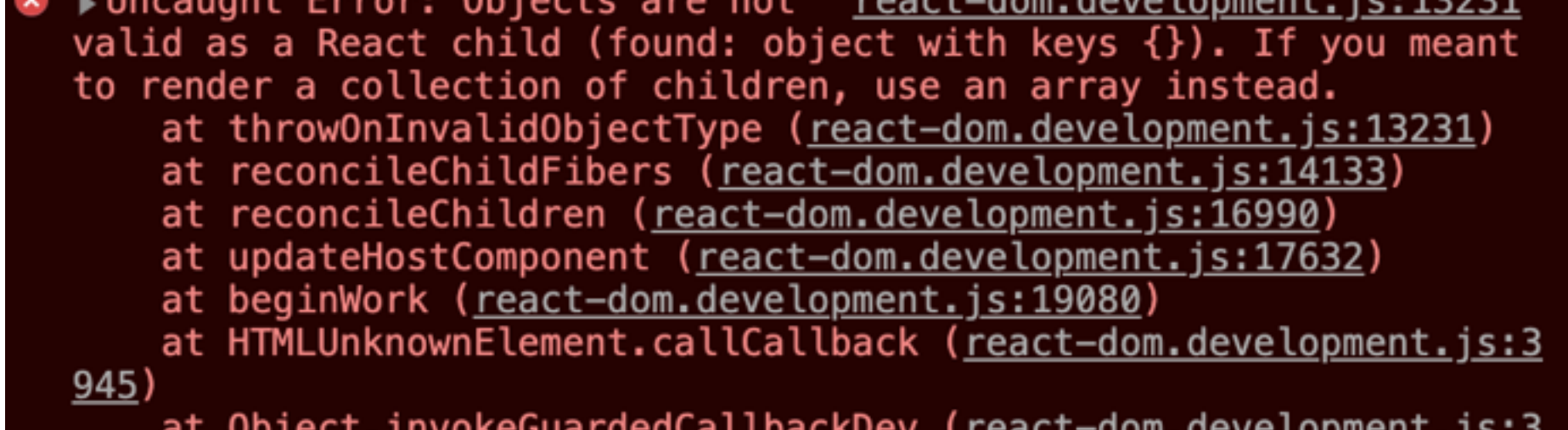
Let's take advantage of the flaw we described earlier and see how it can destroy our application.

Let's try to pass an object into the `ModalHolder` and see what happens:

```
1 import React, {useEffect} from 'react';
2 import {ModalHolder} from '../views/liskov-substitution-principle/ModalHolder';
3
4 function App() {
5
6   const modalContent = { key: " value" }
7
8   return (
9     <div>
10       <ModalHolder contentToShow = {modalContent} />
11     </div>
12   );
13 }
14
15 export default App;
```

App.jsx hosted with ♥ by GitHub [view raw](#)

This code is perfectly fine and will give no compilation error. Now let's open our application and see what happens if we click on the button:



Error

So our application is crashing even though our code has no error. What went wrong here?

Our `Modal` component is allowed to contain another React component. But other components are not bonded to follow that because there is no contract.

What's the Solution?

Now we will see the importance of using TypeScript in our application and why it's important. Let's refactor our `ModalHolder` component to TypeScript and see what happens:

```
1 import { ReactElement, useState } from 'react'
2 import Modal from 'react-modal'
3
4 interface ModalHolderProps {
5   contentToShow: JSX.Element
6 }
7
8 export const ModalHolder = ({ contentToShow : ModalHolderProps }) => {
9   const [visibility, setVisibility] = useState(false)
10
11   return (
12     <>
13       <button onClick={() => setVisibility(true)}> Show Modal</button>
14
15       <Modal isOpen={visibility}>
16         <div>{contentToShow}</div>
17       </Modal>
18     </>
19   )
20 }
```

ModalHolder.tsx hosted with ♥ by GitHub [view raw](#)

So now we have refactored our component to accept the prop `contentToShow` only when it gets a `JSX.Element`.

If someone wants to pass anything that's not a valid component to render, we will get an error:



Valid usage of ModalHolder

Voila! Now all other components that want to plug into the `ModalHolder` component need to follow a contract so that they don't create any unexpected behavior.

Did We Do It?

We have designed our `ModalHolder` component in such a way that no child component that uses this component is able to create any unexpected behavior because they must abide by the rules set by the parent.

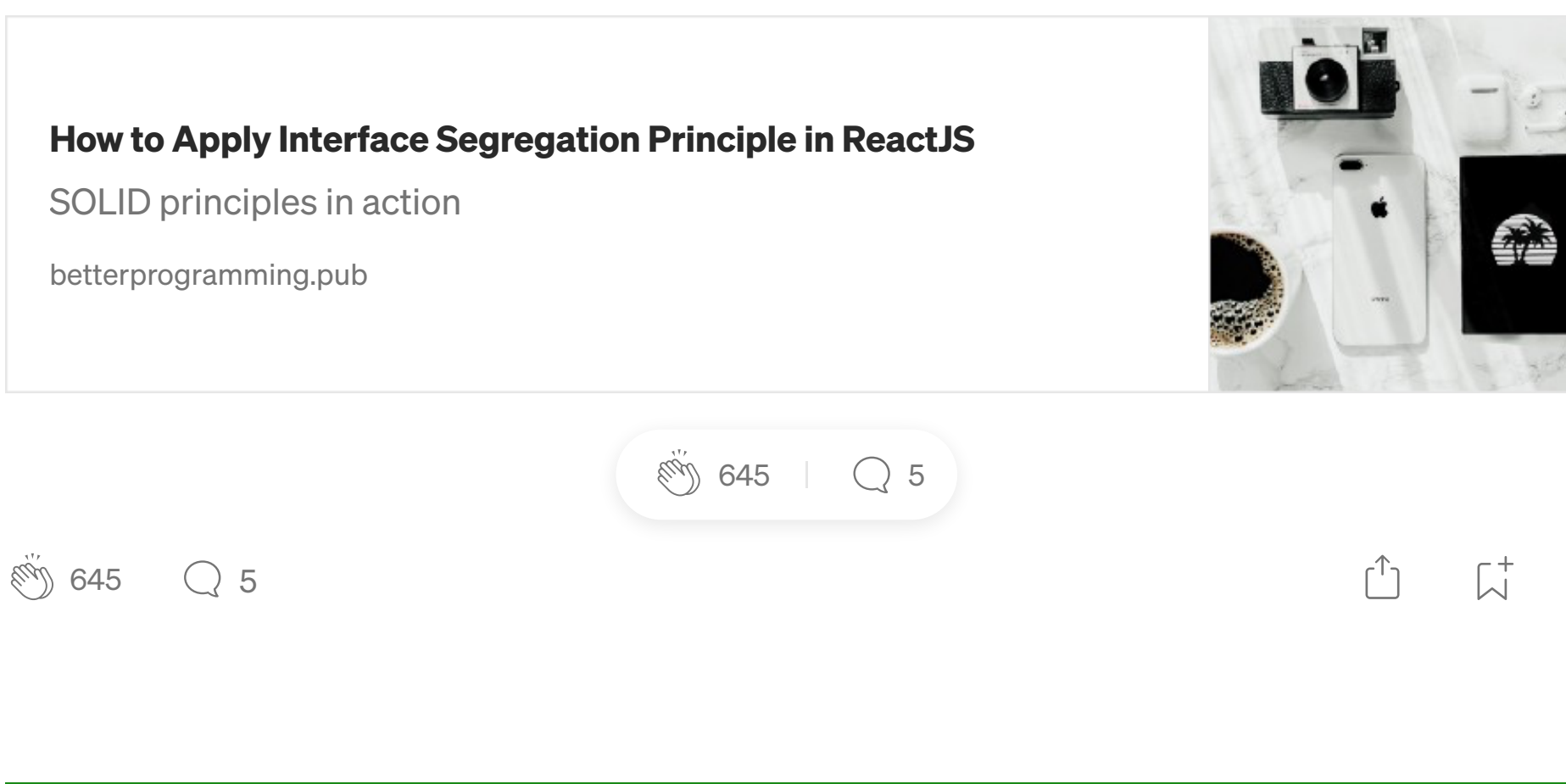
That's exactly what **Liskov Substitution Principle** is all about.

So yes, we did it!

I hope you enjoyed this article as much I enjoyed writing it.

Leave your thoughts below. And have a Great Day :D

Have something to say? Get in touch with me via [LinkedIn](#)



645 5

645 5

Sign up for Coffee Bytes

By Better Programming

A newsletter covering the best programming articles published across Medium [Take a look.](#)

[Get this newsletter.](#)

More from Better Programming

Advice for programmers.

 Lucas Sonnabend · Apr 20, 2021

The Dangers of Async in Python and How To Avoid Them

Understand the async pitfalls in Python with a non-preemptive event loop example — Any async web servers in Python are often heralded as easy to use and very performant. But there are some caveats that can trip up even...

Python · 5 min read

Share your ideas with millions of readers. [Write on Medium](#)

 Charuka Herath · Apr 20, 2021

Web Development in 2021: Dynamic vs. Static vs. Single-Page Architecture

Know the architecture that best suits your website — It is easy to overlook that we have more ways of building websites than with just single-page...

JavaScript · 7 min read

 jsmanifest · Apr 20, 2021

Chain of Responsibility in JavaScript

Run a chain of functions that do stuff — Knowing design patterns is vital in the software industry, as they have been proven to solve real-life problems in business applications. For example, Publish/Subscribe is a common...

JavaScript · 5 min read

 Cristian Salcescu · Apr 19, 2021

9 JavaScript Features That Can Make Any Java Developer Scream

this parameter, objects are hash maps, objects inherit from objects, var has no block scope, and more — Java and JavaScript are sometimes confused...

JavaScript · 5 min read

 Jose Granja · Apr 19, 2021


5 Browser Dev Tools Tips To Ace Your Web Development Skills

Level up your browser coding skills — Browser dev tools have become essential to doing our job as web programmers. Over the years, vendors...

Programming · 6 min read

[Read more from Better Programming](#)

Recommended from Medium

 Murali Krishna Gari... · in Geek Cult...  **Preloading Strategy in Angular** **Save Loading Time.**

 Csharp Space **How to read Connection string from Appsetting.json in ASP.NET Core**

 Konya Haber **chat**

 Chidume Nnamdi... · in Bits and ... **Build a Password Field for the Terminal using Nodejs**

 Max Koretsky · in Angular in Depth **Angular's \$digest is reborn in the newer version of Angular**

 Doug Kintop **Image processing in node with Jimp**

 KARTHIK KONDPAK **Hoisting**

 LiveRunGrow **Creating my first chrome extension**