**What Is the Liskov Substitution Principle?**

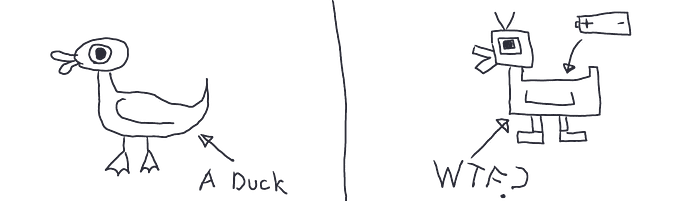
In simple terms, this principle says:

“Subclasses should be substitutable for their superclasses.”

That means subclasses of a particular class should be able to replace the superclass without breaking any functionality.

**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](https://maksimivanov.com/posts/liskov-substitution-principle/)

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:

ModalHolder.jsx

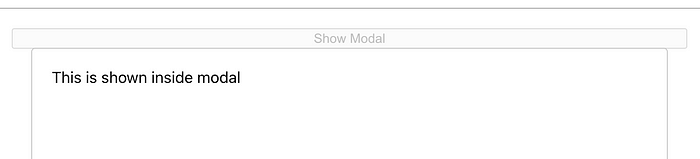
What’s the issue here?

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:

App.jsx

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



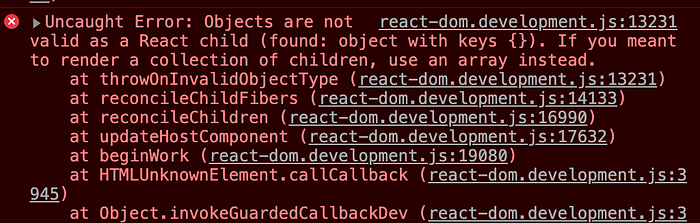
Show Modal

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:

App.jsx

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:

ModalHolder.tsx

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.