Practical examples of dirty code vs. clean code.

Clean Part 2

Day 82/100.

DRY up this code.

Take a look at the code sample below. Go ahead and step back from your monitor. Do you see any patterns? Notice that the component Thingie is identical to ThingieWithTitle with the exception of the title component. This is a perfect candidate for DRYing.

```
// Dirty
import Title from './Title';
export const Thingie = ({ description }) => (
  <div class="thingie">
    <div class="description-wrapper">
      <Description value={description} />
    </div>
  </div>
);
export const ThingieWithTitle = ({ title, description }) => (
  <div>
    <Title value={title} />
    <div class="description-wrapper">
      <Description value={description} />
    </div>
  </div>
);
```

Here we've allowed the passing of children to Thingie. We've then created ThingieWithTitle that wraps Thingie, passing in the Title as its children.

```
 //Clean
import Title from './Title';
export const Thingie = ({ description, children }) => (
  <div class="thingie">
    {children}
    <div class="description-wrapper">
      <Description value={description} />
    </div>
  </div>
);
export const ThingieWithTitle = ({ title, ...others }) => (
  <Thingie {...others}>
    <Title value={title} />
  </Thingie>
);
```

Default values

Take a look at the following code snippet. It defaults the className to "icon–large" using a logical OR statement, similar to the way your grandfather might have done it.

```
// Dirty
const Icon = ({ className, onClick }) => {
  const additionalClasses = className || 'icon-large';
  return (
    <span
      className={`icon-hover ${additionalClasses}`}
      onClick={onClick}>
    </span>
  );
};
```

Here we use ES6’s default syntax to replace undefined values with empty strings. This allows us to use ES6’s single statement form of the fat-arrow function, thus eliminating the need for the return statement.

```
// Clean
const Icon = ({ className = 'icon-large', onClick }) => (
  <span className={`icon-hover ${className}`} onClick={onClick} /
>
);
```

In this even cleaner version, the default values are set in React.

```
// Cleaner
const Icon = ({ className, onClick }) => (
  <span className={`icon-hover ${className}`} onClick={onClick} /
>
);
Icon.defaultProps = {
  className: 'icon-large',
};
```

Why is this cleaner? And is it really better? Don't all three versions do the same thing? For the most part, yes. The advantage of letting React set your prop defaults, however, is that it produces more efficient code, defaults props in aclass based lifecycle component, as well as allows your default values to be checked againstprotoTypes. But there is one more advantage: it declutters the default logic from that of the component itself.

More tips:

- Separate stateful aspects from rendering.

Mixing your stateful data-loading logic with your rendering (or presentation) logic can lead to component complexity. Instead, write a stateful container component whose single responsibility is to load the data.

Then write another component whose sole responsibility is to display the data. This is called the Container Pattern.

- Use stateless functional components

Stateless functional components (SFCs) were introduced in React v0.14.0, and they are used to greatly simplify a render-only component. But some developers haven't let go of the past.

Destructure when applicable.

ES6 introduced the concept of destructuring, which really is your best friend. Destructuring allows you to "pull apart" properties of an object or elements of an array.

Regards Lux Tech Academy.