Higher Order Component

Using higher order components as a means to enhance React elements.

WE'LL COVER THE FOLLOWING

Higher-order components

For a long period of time, higher-order components were the most popular way to enhance and compose React elements. They look really similar to the decorator design pattern because we have component wrapping and enhancing.

On the technical side, the higher-order component is usually a function that accepts our original component and returns an enhanced/populated version of it. The most trivial example is as follows:

```
var enhanceComponent = (Component) =>
  class Enhance extends React.Component {
    render() {
      return (
        <Component {...this.props} />
      )
    }
  };

var OriginalTitle = () => <h1>Hello world</h1>;
var EnhancedTitle = enhanceComponent(OriginalTitle);

class App extends React.Component {
  render() {
    return <EnhancedTitle />;
  }
};
```

The very first thing that the higher-order component does is to render the original component. It's a good practice to proxy pass the `props` to it. This way

we will keep the input of our original component. And here comes the first big benefit of this pattern - because we control the input of the component we

may send something that the component usually has no access to. Let's say that we have a configuration setting that `OriginalTitle` needs:

```
var config = require('path/to/configuration');

var enhanceComponent = (Component) =>
  class Enhance extends React.Component {
    render() {
      return (
        <Component
          {...this.props}
          title={ config.appTitle }
        />
      )
    }
  };

var OriginalTitle  = ({ title }) => <h1>{ title }</h1>;
var EnhancedTitle = enhanceComponent(OriginalTitle);
```

The knowledge for the `appTitle` is hidden into the higher-order component. `OriginalTitle` knows only that it receives a `prop` called `title`. It has no idea that this is coming from a configuration file. That's a huge advantage because it allows us to isolate blocks. It also helps with the testing of the component because we can create mocks easily.

Another characteristic of this pattern is that we have a nice buffer for additional logic. For example, if our `OriginalTitle` needs data from a remote server as well. We may query this data in the higher-order component and again send it as a prop.

```
var enhanceComponent = (Component) =>
  class Enhance extends React.Component {
    constructor(props) {
      super(props);

      this.state = { remoteTitle: null };
    }
    componentDidMount() {
      fetchRemoteData('path/to/endpoint').then(data => {
        this.setState({ remoteTitle: data.title });
      });
    }
    render() {
      return (
        <Component
          {...this.props}
          title={ config.appTitle }
```

```
          remoteTitle={ this.state.remoteTitle }
        />
      )

    }
  };

var OriginalTitle  = ({ title, remoteTitle }) =>
  <h1>{ title }{ remoteTitle }</h1>;
var EnhancedTitle = enhanceComponent(OriginalTitle);
```

Again, the `OriginalTitle` knows that it receives two props and has to render them next to each other. Its only concern is how the data looks, not where it comes from and how.

Dan Abramov made a really good point that the actual creation of the higher-order component (i.e. calling a function like `enhanceComponent`) should happen at a component definition level. Or in other words, it's a bad practice to do it inside another React component because it may be slow and lead to performance issues.

Given below is an implementation of a higher order component:

```
import React from 'react';
import PropTypes from 'prop-types';
import enhanceComponent from './enhanceComponent.jsx';


var Content = (props) => <p>I am { props.name }</p>;
var EnhancedContent = enhanceComponent(Content);

Content.propTypes = {
  name: PropTypes.string
};

export default class App extends React.Component {
  render() {
    return <EnhancedContent name='Content component' />;
  }
};
```

Now that we know how higher order components work, in the next section, we will be discussing an alternate method of composing components in React.