Refactoring and JSX Spread Operators

We're going to add some classes to our component to help make it look just a little bit prettier.

If you'll recall from our `ReactReadingTime` component, when we rendered the `ReadingTime` widget, we added some classes to it:

```
<ReadingTime text={this.state.text} className='col-lg-2 well' />
```

But if you inspect the DOM that it actually rendered, these class names are nowhere to be found. This is because those class names are just passed into our custom component as `this.props.className` and because we didn't do anything with them, they just died there.

We can do the following to add these classes to our markup:

```
render() {
  return (
    <div className={this.props.className}>
      <p>
        Estimated read time:<br /><br />
        <span>{this.state.readTime}</span>
      </p>
    </div>
  );
}
```

And this will work just fine. The names of the classes in the `className` prop will be passed into the `div` and rendered into the DOM. But what if later we want to expand this component and add more props? Then we have to add another attribute to that `div` to pass the new props in. What if we decide to release this module into the wild on npm and the end user wants to add custom attributes to this component? That's where JSX spread operators come into play. With spread operators we can rewrite that like this:

```
render() {
  const {text, ...tags} = this.props;

  return (
    <div {...tags}>
      ...
    </div>
  );
}
```

What's going on there? What we're saying here is to take all the props and apply them to our `div`. So if our props looked like this:

```
{
  className: 'foo',
  name: 'bar'
}
```

Then the component would end up looking like this:

```
<div className='foo' name='bar'></div>
```

This allows us to pass in any number of arbitrary props to the component that we as the module developers may never think of. It also saves a whole lot of typing. Which makes for developer happiness.

So if you update your code to use those spread operators, you should have a nice well drawn around your reading time widget! We're getting there!

Let's update the `ReadingTime` widget to make the output look just a bit nicer:

```
return (
  <div {...props}>
    <p>
      Estimated read time:<br /><br />
      <span>{this.state.readTime} minutes</span>
    </p>
  </div>
)
```

All we did there was just add the word `minutes` at the end of the reading time count.