29 Component Composition

Objectives

- Students will learn to about composition vs inheritance
- Students will learn to compose react components using props

Composition and Inheritance

In programming there's two ways to build new classes out of already-created: `inheritance` and `composition`. These concepts apply to every language that support classes and `inheritance`.

Inheritance is a built-in language feature that allows one class to `extend` (or "inherit") from another class. Composition does this manually by saving a reference of one class as a property in the second class.

Inheritance

Inheritance is great. It allows programms to create hierarchies between classes like `Animal` then create classes like `Mammal`, `Bird`, `Fish` and so on.

It turns out that inheritance is great until it isn't. Writing programs with strict inheritance often ends up colliding against reality. In theory inheritance is great. In reality things get messy!

Designing programs solely with inheritance relationships often leads to hard-to-express relationships between classes, and classes being shoe-horned into baffling inheritance schemes just to satisfy inheritance requirements.

Inheritance

```
class Person {}
class Seattleite extends Person {}
```

Composition

`Composition` builds classes out of other classes by creating an instance of one class inside another class instead of directly inheriting from it. Avoiding string inheritance relationships allows the program to escape rigid heirarchy structures and allows us to `compose` new classes out of just the parts of other classes that we need.

Composition

```
class Person {}
class Seattleite {
  constructor() {
    this.person = new Person();
  }
}
```

Favor Composition over Inheritance

Favor composition over inheritance is a programming principle that advocates to composes classes via `composition` rather than through `inheritance`.

Inheritance

```
class Person {}
class Seattleite extends Person {}
```

Composition

```
class Person {}
class Seattleite {
  constructor() {
    this.person = new Person();
  }
}
```

Inheritance in React

Here's what Facebook has to say about inheritance in React:

> At Facebook, we use React in thousands of components, and we haven’t found any use cases where we would recommend creating component inheritance hierarchies.
>
> Props and composition give you all the flexibility you need to customize a component’s look and behavior in an explicit and safe way. Remember that components may accept arbitrary props, including primitive values, React elements, or functions.
>
> If you want to reuse non-UI functionality between components, we suggest extracting it into a separate JavaScript module. The components may import it and use that function, object, or a class, without extending it.

props.children

React has a special property on components `props.children`. This property allows a components to be passed into another component and be rendered inside. (This is how everything inside `<Router>` works!)

Here the `<h1>` and `<p>` inside `<FancyBorder>` are accessible through the `props.childen` prop.

```
function FancyBorder(props) {
  return (
    <div className={'FancyBorder FancyBorder-' + props.color}>
      {props.children}
    </div>
  );
}
```

```
function WelcomeDialog() {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">Welcome</h1>
      <p className="Dialog-message">Thank you for visiting our spacecraft!</p>
    </FancyBorder>
  );
}
```

Custom "holes" in Components

`props.children` is restricted to rendering everything passed into the component all at once. If you want more fine-grain control over things passed into the component you can pass them through props with specific names, just like any other prop.

```
function SplitPane(props) {
  return (
    <div className="SplitPane">
      <div className="SplitPane-left">
        {props.left}
      </div>
      <div className="SplitPane-right">
        {props.right}
      </div>
    </div>
  );
}

function App() {
  return (
    <SplitPane left={<Contacts/>} right={<Chat/>} />
  );
}
```

React Reacting to Double Clicks (and other events)

React has a long list of events you can attach to components and react to.

Refer to the docs to see all the events available and properties attached to event objects.

```
onClick onContextMenu onDoubleClick onDrag onDragEnd onDragEnter onDragExit
onDragLeave onDragOver onDragStart onDrop onMouseDown onMouseEnter onMouseLeave
onMouseMove onMouseOut onMouseOver onMouseUp
```

```
boolean altKey
boolean ctrlKey
boolean metaKey
DOMEventTarget relatedTarget
number screenX
number screenY
boolean shiftKey
... and many many more
```