



Composition vs. Inheritance

Developing Applications using ReactJS





Objectives

- To understand Composition and the use of containment and specialisation
- To understand why inheritance is rarely used with React



Composition

- **One of key features of ReactJS**
- **Components by different developers should work well together**
- **Important that functionality can be added to component without affecting the codebase**
 - E.g. Should be possible to add local state to component without affecting its children
 - State and lifecycle hooks in components should be used in moderation



Composition - Containment

- React has special *children* prop to pass children elements directly into the output of a parent component
- Useful in cases where parent component doesn't know what their children will be ahead of time
- Can be used in functions or in classes
 - Works as equally well in either



Composition - Containment

- **EXAMPLE 1 – a containment component for other components**
 - The containment component can be the same for all children elements inside it

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

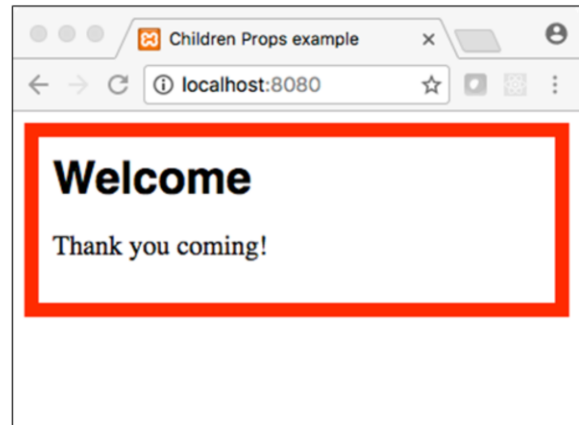
function ParentComponent() {
  return (
    <ContainmentComponent color="red">
      <h1 className="title">Welcome</h1>
      <p className="message">Thanks for coming!</p>
    </ContainmentComponent>
  );
}
```



Composition - Containment

- **EXAMPLE 1 – a containment component for other components**
 - Given a stylesheet applied as on the left, the output would look as on the right...

```
<style>
  .conStyle {
    padding: 10px 10px;
    border: 10px solid;
  }
  .conStyle-red {
    border-color: red;
  }
  .title {
    margin: 0;
    font-family: sans-serif;
  }
  .message {
    font-size: larger;
  }
</style>
```



118

Anything inside the `<ContainerComponent>` JSX tag in the `<ParentComponent>` get passed into the `<ContainerComponent>` as a `children` prop. Since `<ContainerComponent>` renders `{props.children}` inside a `<div>`, the passed elements appear in the final output.



Composition - Containment

- You may also wish to be able to address different subcomponents within a containment
- **EXAMPLE 2 – a containment component addressing sub-components**

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

function ParentComponent() {
  return (
    <SplitterComponent
      left={<LeftSplit /> right={<RightSplit />} />
    );
}>
```

119

The component `<LeftSplit>` will return `<div className="LeftSplit">I am the left split</div>`

The component `<RightSplit>` will return `<div className="RightSplit">I am the right split</div>`

These are just objects and so can be passed as props like any other data.

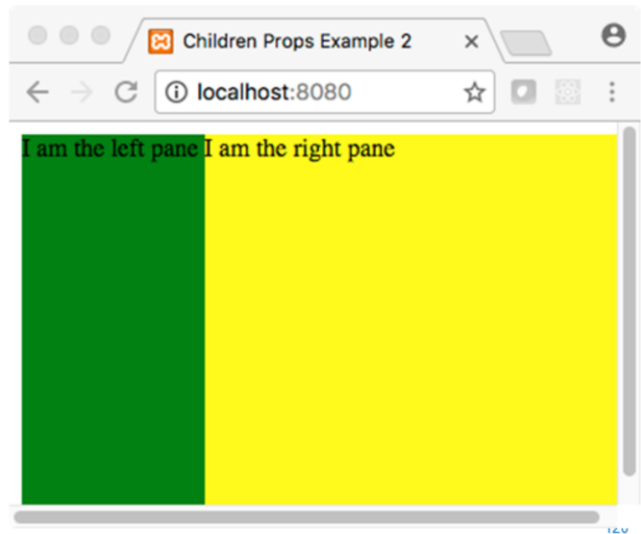


Composition - Containment

■ EXAMPLE 2 – a containment component addressing sub-components

- Given a stylesheet applied as on the left, the output would look as on the right...

```
<style>
  html, body, #app {
    width: 100%;
    height: 100%;
  }
  .SplitterComponent {
    width: 100%;
    height: 100%;
  }
  .SplitterComponent-left {
    float: left;
    width: 30%;
    height: 100%;
  }
  .SplitterComponent-right {
    float: left;
    width: 70%;
    height: 100%;
  }
  .LeftSplit {
    width: 100%;
    height: 100%;
    background: green;
  }
  .RightSplit {
    width: 100%;
    height: 100%;
    background: yellow;
  }
</style>
```



Anything inside the `<ContainerComponent>` JSX tag in the `<ParentComponent>` get passed into the `<ContainerComponent>` as a `children` prop. Since `<ContainerComponent>` renders `{props.children}` inside a `<div>`, the passed elements appear in the final output.



Composition – Specialisation

- Some components can be considered as ‘special cases’ of others
- A more specific component can render a more generic one and configure it with props
- **EXAMPLE 1 - FirstChild is special case of Parent**

```
function FirstChildComponent(props) {  
  return (  
    <ContainmentComponent color={props.color}>  
      <h1 className="title">{props.title}</h1>  
      <p className="message">{props.message}</p>  
      {props.children}  
    </ContainmentComponent>  
  );  
};  
function ParentComponent() {  
  return (  
    <FirstChildComponent color="red"  
      title="Welcome"  
      message="Thanks for coming!"  
    />  
  );  
};
```

121

Given that the `<ContainmentComponent>` and styling are as in the first example for Containment, the output should be the same.

A further example can be seen at:

<http://codepen.io/gaearon/pen/gwZbYa?editors=0010>



Inheritance

- **Facebook (the developers of React) have not found any cases where component inheritance hierarchies would be recommended**
- **Props and composition give all flexibility needed to customise component look and behaviour**



Objectives

- To understand Composition and the use of containment and specialisation
- To understand why inheritance is rarely used with React

Exercise Time



- **No exercise for this chapter.**