WELCOME TO DAY 2 OF REACTJS BOOTCAMP

AGENDA

- React Lifecycle Methods
- Composibility
- Passing Props
- Refs

REACT LIFECYCLE METHODS

- render()
- getInitialState()
- getDefaultProps()
- componentWillMount()
- componentDidMount()
- componentWillRecieveProps(nextProps)
- shouldComponentUpdate(nextProps, nextState)
- componentWillUpdate(nextProps, nextState)
- componentDidUpdate(prevProps, prevState)
- componentWillUnmount()

these can not update state

RENDER

- This is the meat of your component and generates the React Element to draw into the DOM
- Pure function - does not modify state

GETINITIALSTATE

- return the basic representation of your state. Invocked once before component is mounted

GETDEFAULTPROPS

- return the basic representation of your props. Invocked once before component is mounted
- values set on this.props in case something does not come in
- Can also set propTypes to enforce type of objects passed in as props
- propType mismatch issues a warning in development

COMPONENTWILLMOUNT

- Invoked once before initial render
- chance to update state before render

COMPONENTDIDMOUNT

- Good place to use 3rd party libs or attach listeners or use ajax
- First chance to interact with the DOM node

COMPONENTWILLRECIEVEPROPS

- invocked when props from parent change
- can react to prop transistion before the rerender is called
- can still access old props via this.props and new props via the first parameter
- calling setState here does not cause an additional render but rather changes state before the render occurs

SHOULDCOMPONENTUPDATE

- see PureRenderMixin
- can return false to stop a rerender and gain that bit of performance
- defaultly always returns true

COMPONENTWILLUPDATE

- not called for initial render
- can not call this.setState
- perform prep before rerender

COMPONENTDIDUPDATE

- not called for initial render
- perform work on domNode

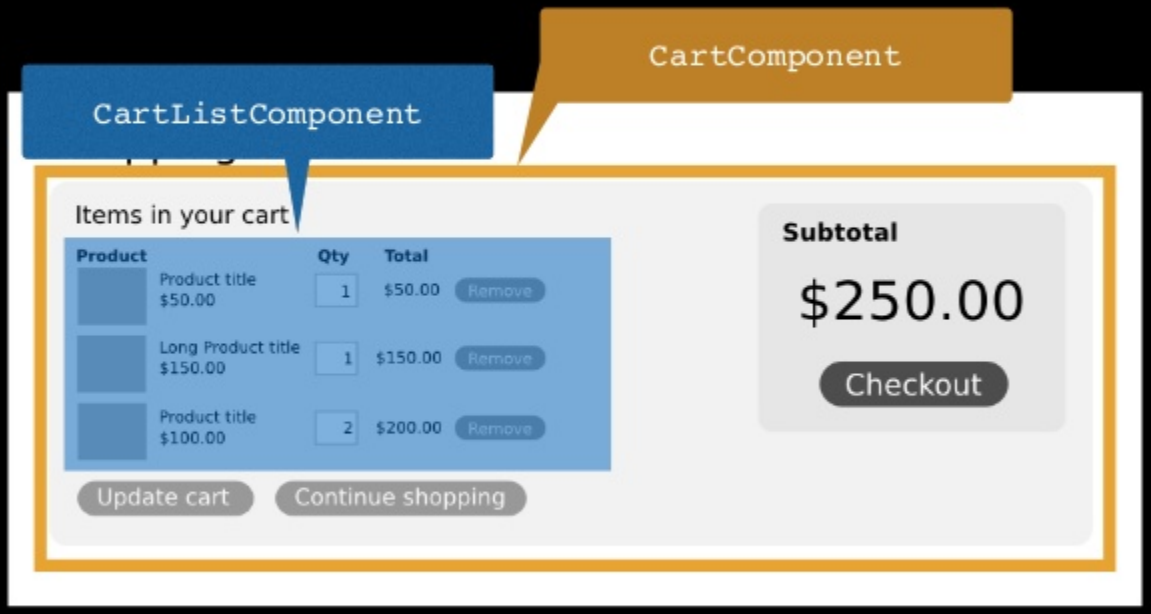
COMPONENTWILLUNMOUNT

- place to perform teardown code and remove listeners etc

COMPOSIBILITY

- React components can be nested in each other and pass information down to children

What if we "separate" another way?



- Notice how Cart Component is a parent to CartListComponent
- CartListComponent may contain child components to render each list item
- With this scheme the list items can own their state of qty and total while the cart list is only concerned with having x amount of generic items

- An owner is the component that sets the props of other components.
- React components can be created and rerendered with props from parent components
- Children usually rendered in the order they appear in the DOM but can dynamically be added like search results

PASSING PROPS

```
<div classname="wrapper">
  <childcomponent proptopass="{this.propToPass}">
</childcomponent></div>
```

REFS

- Reference to a component in that view
- Can be accessed in parent component by this.refs.refName

```
<div classname="wrapper">
  <childcomponent ref="childComponent">
</childcomponent></div>
```

- Can reach out to components public facing methods
- Can reach out to native components like an input tag
- NEVER access a ref during render
- Automagically created and destroyed for you