Notes

CS50's Mobile App Development with React Native on EdX

Sparsh Jain

January 21, 2021

Contents

Chapter 1

React, Props, State

1.1 Classes

- Syntax introduced in ES6
- Simplifies the defining of complex objects with their own prototypes
- Classes vs instances
- Methods vs static methods vs properties
- new, constructor, extends, super

```
// Set should maintain a list of unique values and should
↪  support add, delete, and inclusion
// It should also have the ability to get its size

class Set {
  constructor(arr) {
    this.arr = arr
  }

  add(val) {
    if (!this.has(val)) this.arr.push(val)
  }

  delete(val) {
    this.arr = this.arr.filter(x => x !== val)
  }
```

```javascript

  has(val) {
    return this.arr.includes(val)
  }

  get size() {
    return this.arr.length
  }
}

const s = new Set([1,2,3,4,5])

// trying to add the same value shouldn't work
s.add(1)
s.add(1)
s.add(1)
console.log('s should have 5 members and actually has:',
 ↪  s.size)

console.log('s should contain 5:', s.has(5))

s.add(6)
console.log('s should contain 6:', s.has(6))
console.log('s should have 6 members and actually has:',
 ↪  s.size)

s.delete(6)
console.log('s should no longer contain 6:', !s.has(6))
console.log('s should have 5 members and actually has:',
 ↪  s.size)
```

Program 1.1: Class Example (Set) in JavaScript

```javascript
// We can also extend the native implementation of Set if we
 ↪  wanted to do something
// like log on addition or create new methods

class MySet extends Set {
  constructor(arr) {
    super(arr)
    this.originalArray = arr
  }

  add(val) {
    super.add(val)
    console.log(`added ${val} to the set!`)
  }

  toArray() {
    return Array.from(this)
  }

  reset() {
    return new MySet(this.originalArray)
  }
}

const s = new MySet([1,2,3,4,5])
s.add(6)
s.add(7)
console.log(s.toArray())

const reset = s.reset()
console.log(reset.toArray())
```

Program 1.2: Extending JS Set Class

```
class Todo {
  constructor(configuration) {
    this.text = configuration.text  'New TODO'
    this.checked = false
  }

  render() {
    return (
      <li>
        <input type="checkbox" checked={this.checked} />
        <span>{this.text}</span>
      </li>
    )
  }
}
```

Program 1.3: Using Class for Todo App

1.2 React

- Allows us to write delcarative views that "react" to changes in data
- Allows us to abstract complex problems into smaller components
- Allows us to write simple code that is still performant

1.2.1 Imperative vs Declarative

- How vs What
- Imperative programming outlines a series of steps to get to what you want
- Declarative programming just declares what you want

```javascript
// assume createElement() exists, similar in abstraction to
↪  document.createElement()

const strings = ['E', 'A', 'D', 'G', 'B', 'E']

function Guitar() {
  // create head and add pegs
  const head = createElement('head')
  for (let i = 0; i < 6; i++) {
    const peg = createElement('peg')
    head.append(peg)
  }


  // create neck and add frets
  const neck = createElement('neck')
  for (let i = 0; i < 19; i++) {
    const fret = createElement('fret')
    head.append(fret)
  }


  // create body and add strings
  const body = createElement('body')
  body.append(neck)
  strings.forEach(tone => {
    const string = createElement('string')
    string.tune(tone)
    body.append(string)
  })

  return body
}
```

Program 1.4: Building Guitar - The Imperative Way

```
const strings = ['E', 'A', 'D', 'G', 'B', 'E']

function Guitar() {
  return (
    <Guitar>
      {strings.map(note => <String note={note} />)}
    </Guitar>
  )
}
```

Program 1.5: Building Guitar - The Declarative Way

1.2.2 React is Declarative

- Imperative vs Declarative
- The browser APIs aren't fun to work with
- React allows us to write what we want, and the library will take care of the DOM manipulation

```
const SLIDE = {
  title: 'React is Declarative',
  bullets: [
    'Imeritive vs Declaraive',
    "The browser APIs are't fun to work with",
    'React allows us to write what we want, and the library
     ↪ will take care of the DOM manipulation',
  ],
}

const CLASSNAMES = {title: 'title', bullet: 'bullet'}

function createSlide(slide) {
  const slideElement = document.createElement('div')

  const title = document.createElement('h1')
  title.className = CLASSNAMES.title
  title.innerHTML = slide.title
  slideElement.appendChild(title)

  const bullets = document.createElement('ul')
  slide.bullets.forEach(bullet => {
    const bulletElement = document.createElement('li')
    bulletElement.className = CLASSNAMES.bullet
    bulletElement.innerHTML = bullet
    bullets.appendChild(bulletElement)
  })
  slideElement.appendChild(bullets)

  return slideElement
}
```

Program 1.6: Imperative Slide

```
const SLIDE = {
  title: 'React is Declarative',
  bullets: [
    'Imeritive vs Declaraive',
    "The browser APIs are't fun to work with",
    'React allows us to write what we want, and the library
     ↪  will take care of the DOM manipulation',
  ],
}

function createSlide(slide) {
  return (
    <div>
      <h1>{SLIDE.title}</h1>
      <ul>
        {SLIDE.bullets.map(bullet => <li>{bullet}</li>)}
      </ul>
    </div>
  )
}
```

Program 1.7: Declarative Slide

1.2.3 React is Easily Componentized

- Breaking a complex problems into discrete components
- Can reuse these components
  - Consistency
  - Iteration speed
- React's declarative nature makes it easy to customize components

```html
<div>
  <div>
    <h1>React</h1>
    <ul>
      <li>Allows us to write declarative views that "react" to changes in data</li>
      <li>Allows us to abstract complex problems into smaller components</li>
      <li>Allows us to write simple code that is still performant</li>
    </ul>
  </div>
  <div>
    <h1>React is Declarative</h1>
    <ul>
      <li>Imerative vs Declarative</li>
      <li>The browser APIs aren't fun to work with</li>
      <li>React allows us to write what we want, and the library will take care of the DOM manipulation</li>
    </ul>
  </div>
  <div>
    <h1>React is Easily Componentized</h1>
    <ul>
      <li>Breaking a complex problem into discrete components</li>
      <li>Can reuse these components
      <li>React's declarative nature makes it easy to customize components</li>
    </ul>
  </div>
</div>
```

Program 1.8: HTML Slideshow

```
const slides = [
  {
    title: 'React',
    bullets: [
      'Allows us to write declarative views that "react" to
       ↪  changes in data',
      'Allows us to abstract complex problems into smaller
       ↪  components',
      'Allows us to write simple code that is still
       ↪  performant',
    ],
  },
  {
    title: 'React is Declarative',
    bullets: [
      'Imerative vs Declarative',
      "The browser APIs aren't fun to work with",
      'React allows us to write what we want, and the library
       ↪  will take care of the DOM manipulation',
    ],
  },
  {
    title: 'React is Easily Componentized',
    bullets: [
      'Breaking a complex problem into discrete components',
      'Can reuse these components',
      "React's declarative nature makes it easy to customize
       ↪  components",
    ],
  },
]

// TODO implement slideshow
const slideShow = (
  <div>
    {slides.map(slide => <Slide slide={slide} />)}
  </div>
)

// note that this pseudocode differs from react.
// in react, accessing the slide title would be done with
↪  {slide.slide.title}
// and accessing the bullets would be {slide.slide.bullets}
const Slide = slide => (
  <div>
    <h1>{slide.title}</h1>
    <ul>
      {slide.bullets.map(bullet => <li>{bullet}</li>)}
    </ul>
  </div>
)
```

Program 1.9: React Slideshow

1.2.4 React is Performant

- We write what we want and React will do the hard work
- Reconciliation - the process by which React syncs changes in app state to DOM
  - Reconstructs the virtual DOM
  - Diffs the virtual DOM against the DOM
  - Only makes the changes needed*

1.3 Writing React

- JSX
  - XML-like syntax extension of JavaScript
  - Transpiles to JavaScript
  - Lowercase tags are treated as HTML/SVG tags, uppercase are treated as custom components

- Components are just functions
  - Returns a node (something React can render, e.g. a <div />)
  - Receives an object of the properties that are passed to the element

Note: Can run/try react on codesandbox.io .

1.4 Props

- Passed as an object to a component and used to compute the returned node
- Changes in these props will cause a recomputation of the returned node ("render")
- Unlike in HTML, these can be any JS value (use {to let react know})

```
import React from 'react';
import { render } from 'react-dom';
import Hello from './Hello';

const styles = {
  fontFamily: 'sans-serif',
  textAlign: 'center',
};

const App = (props) => (
  <div style={styles}>
    <h2>{props.count}</h2>
  </div>
);

const App2 = function(props) {
  return (
    <div style={styles}>
      <h2>{props.count}</h2>
    </div>
  )
}
let count = 0
```

```

setInterval(
  function() {render(<App2 count={count++} />,
  ↪ document.getElementById('root'))},
  1000
)
```

Program 1.10: Props in React

1.5 State

- Adds internally-managed configuration for a component
- ‘this.state’ is a class property on the component instance
- Can only be updated by invoking ‘this.setState()’
  - Implemented in React.Component
  - setState() calls are batched and run asynchronously
  - Pass an object to be merged, or a function of previous state
- Changes in state also cause re-renders

```
import React from 'react';
import { render } from 'react-dom';
import Hello from './Hello';

const styles = {
  fontFamily: 'sans-serif',
  textAlign: 'center',
};

class App extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      count: 0,
    }
  }

  increaseCount() {
    this.setState(prevState => ({count: prevState.count + 1}))
    this.setState(prevState => ({count: prevState.count + 1}))
    console.log(this.state.count)
  }

  render() {
    return (
      <div style={styles}>
        <div>
          <button onClick={() =>
           ↪  this.increaseCount()}>Increase</button>
        </div>
        <h2>{this.state.count}</h2>
      </div>
    )
  }
}

render(<App />, document.getElementById('root'))
```

Program 1.11: React States

1.6 Todo App

1. Layout what you need

```
const list = document.getElementById('todo-list')
const itemCountSpan =
  ↪ document.getElementById('item-count')
const uncheckedCountSpan =
  ↪ document.getElementById('unchecked-count')

//  <li>
//    <input type="checkbox" />
//    <button>delete</button>
//    <span>text</span>
//  </li>

function newTodo() {
  // get text
  // create li
  // create input checkbox
  // create button
  // create span
  // update counts
}

function deleteTodo() {
  // find the todo to delete
  // delete
  // update the counts
}
```

2. Componentize

```
const list = document.getElementById('todo-list')
const itemCountSpan =
  ↪ document.getElementById('item-count')
const uncheckedCountSpan =
  ↪ document.getElementById('unchecked-count')

//  <li>
//    <input type="checkbox" />
//    <button>delete</button>
//    <span>text</span>
//  </li>

function createTodo() {
  // make li

  // make input

  // make button

  // make span
}

function newTodo() {
  // get text

  // invoke createTodo()

  // update counts

  // apend to list
}

function deleteTodo() {
  // find the todo to delete
  // remove
  // update counts
}
```

3. Write Declaratively (Inner HTML)

```
const list = document.getElementById('todo-list')
const itemCountSpan =
 ↪  document.getElementById('item-count')
const uncheckedCountSpan =
 ↪  document.getElementById('unchecked-count')

//  <li>
//    <input type="checkbox" />
//    <button>delete</button>
//    <span>text</span>
```

```javascript
//  </li>

function createTodo() {
  const li = document.createElement('li')
  li.innerHTML = `
    <input type="checkbox" />
    <button>delete</button>
    <span>text</span>
  `
  return li
}

function newTodo() {
  // get text

  // invoke createTodo()

  // update counts

  // apend to list
}

function deleteTodo() {
  // find the todo to delete
  // remove
  // update counts
}
```

4. Store todo list in a Data Structure

```javascript
// store todos in memory
let todos = []

function renderTodo(todo) {
  // render a single todo
}

function render() {
  // render the todos in memory to the page
  list.innerHTML = ''
  todos.map(renderTodo).forEach(todo => list.appendChild(todo))

  // update counts

  return false
}

function addTodo(name) {
  const todo = new Todo(name)
  todos.push(todo)
  return render()
}

function removeTodo(todo) {
  todos = todos.filter(t => t !== todo)
  return render()
}
```

5. React it

```
import React from 'react';
import { render } from 'react-dom';

let id = 0

const Todo = props => (
  <li>
    <input type="checkbox" checked={props.todo.checked}
     ↪  onChange={props.onToggle} />
    <button onClick={props.onDelete}>delete</button>
    <span>{props.todo.text}</span>
  </li>
)

class App extends React.Component {
  constructor() {
    super()
    this.state = {
      todos: [],
    }
  }

  addTodo() {
    const text = prompt("TODO text please!")
    this.setState({
      todos: [
        ...this.state.todos,
        {id: id++, text: text, checked: false},
      ],
    })
  }

  removeTodo(id) {
    this.setState({
      todos: this.state.todos.filter(todo => todo.id !== id)
    })
  }

  toggleTodo(id) {
    this.setState({
      todos: this.state.todos.map(todo => {
        if (todo.id !== id) return todo
        return {
          id: todo.id,
          text: todo.text,
          checked: !todo.checked,
        }
      })
    })
  }

  render() {
    return (
      <div>
        <div>Todo count: {this.state.todos.length}</div>
        <div>Unchecked todo count: {this.state.todos.filter(todo => !todo.checked).length}</div>
        <button onClick={() => this.addTodo()}>Add TODO</button>
        <ul>
          {this.state.todos.map(todo => (
            <Todo
              onToggle={() => this.toggleTodo(todo.id)}
              onDelete={() => this.removeTodo(todo.id)}
              todo={todo}
            />
          ))}
        </ul>
      </div>
    )
  }
}


render(<App />, document.getElementById('root'));
```

Program 1.12: Todo App in React

1.7 React Native

Why limit React to just web? Bring it to mobile!

- A framework that relies on React core
- Allows us build mobile apps using only JavaScript
  - Learn once, write anywhere
- Supports iOS and Android