Reminders

• React is new and very different! Start early and ask questions

Upcoming Deadlines

• HW7 due 11pm Thursday (8/04)
Last Time...
• Intro to React
• Many examples!

Today’s Agenda
• HW8 Overview
• React Examples
• Using Leaflet for Maps in React
What's difference between Java and JavaScript?

Jay Prakash
It is like "car and carpet".

Faisal
It's like "moon and honeymoon".

1.2k Likes

210 Replies
Node and NPM

- Used to manage our React development environment

- Install Node.js: [https://nodejs.org/en/](https://nodejs.org/en/)
  - This will also install NPM

- Install the LTS version (not the current version)
  - Windows Users: Make sure you “Add to PATH” (should be automatically selected by default)
  - MacOS Users: may get a warning about the installer not coming from a “verified developer.”
    To resolve this, open System Preferences and navigate to Security & Privacy > General. There, you'll be able to click “Open” to run the Node/NPM installer.
React (JavaScript library)

- React (also known as React.js or ReactJS) is an open-source front-end JavaScript library

- React code is made of entities called components, which allow you to implement different UI in different classes
  - Think of a component like a synthetic HTML tag

- Allow direct addition of HTML to the code

- Check HTML syntax (refer to the lecture material for this)
React Components

- Each component has a render method to determine what it looks like on the page

- Components form a tree:

- Components can have **state**, which is local information used for rendering

- Components can receive information from its parent using **props**
  - Use functions as props as **callbacks**
React Components

```
<MyComponentName value="Hello World"
onChange={() => doSomething()}/>
```

- **MyComponentName** is the name of your component/class
- In this case, the **props** are **value** and **onChange**
- **onChange** takes in a function, which we call a **callback**
  - this is how we can pass information up the tree, from a child to a parent
React Developer Tools

• You should download the React Developer Tools!

• This is a Chrome/Edge extension that allows you to view additional details about your React app
IntelliJ Ultimate Edition

**Community**
- No Javascript/Typescript support

**Ultimate**
- Has Javascript/Typescript support

No documentation on hover!
HW8 Overview

- Draw lines on a map in React
- Starter code has (most of) the pieces, but not much functionality.
  - Lots of hard-coded values, placeholders (\texttt{console.log} instead of doing stuff), etc..
- Your job: "wire all the pieces together"
  - Accept user input
  - Process/parse the data
  - Error check – users do weird stuff, make sure you can't crash
  - Move data between components as necessary
  - Add the actual functionality in response to user input.
- Structure:
  - Top-level \texttt{<App>} component, with two child components.
HW8 Component Structure
Running a React App

**npm**: Similar to gradle, but we need to install manually the first time.

In the terminal, change directory until you're in the same place as the "package.json" file for the project you want to run.

To Install (first time): `npm install`
To Run (every time): `npm start`

Once started, you can edit and save files and the page will automatically reload – no need to restart. Use Control-C to shut down when you're done developing.
Section Demo
Running The Section Demo

• Download and unzip the section demo.
• IntelliJ: File > Open...

• From the IntelliJ terminal:
  - `npm install`

• Success!
  (These warnings are normal).
Running The Section Demo

After installation finishes, run `npm start`

A browser window should open up automatically.

Doesn’t work?

- Did you install the correct version (LTS)?
- Try running `npm audit fix --force` and then run `npm start`
Example 1:

**React Boilerplate**

- This is a React component with minimum parts needed to display a Hello World message.

```javascript
render() {
  return (
    <p>Hello World</p>
  );
}
```
Example 2:
Rendering an Array of Elements

- This shows you how to render an array of JSX Elements
- Recall:
  ```javascript
  let myParagraph: JSX.Element = <p>Hello World</p>;
  render() {
    let arr: JSX.Element[] = [<p>Hello World!</p>,
                               <p>Hola Mundo!</p>,
                               <p>Bonjour Monde</p>];

    return (  
      <div>
        {arr}
      </div>
    );
  }
  ```
Example 2:
Rendering an Array of Elements

• What happens if you don’t put curly-braces around arr?
  – It gets interpreted as plain text!

render() {
  let arr: JSX.Element[] = [<p>Hello World!</p>,
    <p>Hola Mundo!</p>,
    <p>Bonjour Monde</p>];

  return (
    <div>
      arr
    </div>
  );
}

• Curly braces {} are special syntax in JSX, used to evaluate a JavaScript expression during compilation.
Example 2: Rendering an Array of Elements

- When rendering any **array** of JSX elements, each element needs a unique “key” **prop**. Keys can be anything as long as they are **unique**.

```jsx
render() {
  let arr: JSX.Element[] = [<p key={1}>Hello World!</p>,
    <p key={2}>Hola Mundo!</p>,
    <p key={3}>Bonjour Monde</p>];

  return (
    <div>
      {arr}
    </div>
  );
}
```

⚠️ Warning: Each child in a list should have a unique "key" prop.

Check the render method of `App`. See [https://reactjs.org/link/warning-keys](https://reactjs.org/link/warning-keys) for more information.

at p
at App (http://localhost:3000/main.a5b9a06...hot-update.js:37:208)
Example 3:

**Drawing on a Map**

- We will use the React Leaflet plugin to display an interactive map of the campus using React.
- `<Map>` tag: creates an instance of the map component. This component is also provided with your HW8 starter code.
- We’re using `<Map>` in HW8 and HW9 to draw lines/paths on top of images (like a map of campus!)
- `<MapContainer>` – Creates a container for the map with properties such as the default position and zoom level.
- `<MapLine>` – Represents an edge on the map.
  - Takes the source and destination coordinates as well as the color of each edge.
  - Map should be in the format provided in HW7.
Example 3: Drawing on a Map

```javascript
render() {
  return (
    <div>
      <h1 id="app-title">Line Mapper!</h1>
      <div>
        <Map edgeList={[]}/>
      </div>
    </div>
  )
}
```

We pass in an empty array into `Map` as the `edgeList` prop.
Example 3:

**Drawing on a Map**

- Why did we need to pass in the `edgeList` prop into the `Map` element? `<Map edgeList=[] />`

**Map.tsx:**

```typescript
interface MapProps {
    edgeList: ColoredEdge[]; // edges to be drawn
}

class Map extends Component<MapProps, {}> {

    All `Map` elements **must** have the `props` defined in the interface passed in on the left.
Example 4: State

- We are initializing the information about our lines in our constructor.
  - Initialize `state` with `this.state = {...}`

- We are storing our lines and the color of our lines in our `state`.

- App's `state` in this example is never getting updated after initialization.
Example 4:

State

constructor(props: any) {
    super(props);
    // initialize tempLines
    // and color_
    this.state = {
        color: color_,
        lines: tempLines
    };
}

render() {
    return (
        ...
        <Map edgeList={this.state.lines} />
        ...
    )
}

We created lines in App's constructor, passed them through this.state into Map as the edgeList prop.
Example 4:

State

interface AppState {
    lines: ColoredEdge[];
    color: string
}

class App extends Component<{}, AppState> {

    App's state object must follow the interface passed in on the right.

    this.state = {};

    Compiler Error: Type '{} is missing the following properties from type 'Readonly': lines, color
Aside: Interfaces

Interfaces define what properties an object is required to have.
- Conceptually: the “shape” of an object

```typescript
interface HasLabel { interface Empty {
  label: string; // nothing
}
}

let obj1: HasLabel = { label: "label1" }; let obj2: Empty = { label: "label2" };

console.log(obj1.label); console.log(obj2.label);

Compiler Error: Property 'label' does not exist on type 'Empty'.
```
Example 5:
Changing State

- **App** still stores a current color and a list of edges

- We have 3 buttons to update the color to **red**, **blue**, or **green**.

- Button’s `onClick` event listener calls `setState` in **App** to change the color and trigger a **re-render** when the button is clicked.
  - Initialize state using `this.state = { ... }`
  - Use `this.setState` to update the `state` after initialization
    - Otherwise, React might not notice the `state` update and not update the UI!
Example 5: 
Changing State

Line Mapper!
Example 5:  
Changing State

```jsx
<button onClick={this.onGreenClick}>Green</button>

onGreenClick = () => {
    const tempLines = this.state.lines;
    for (let i in tempLines) {
        tempLines[i].color = "green";
    }
    let newState = {
        color: "green",
        lines: tempLines
    };
    this.setState(newState);
};
```

When the button is clicked, we grab the old state, modify it, and then replace the old state with our new state!
Example 5:  
Changing State

React’s re-renderer watches for state updates. When it detects a state update, a re-render is queued. It does not happen instantly, as React might group multiple state updates in one re-render.

```javascript
this.setState(someNewState)
```

```
render() {
  return(
    ...
    <div>
      <Map edgeList={this.state.lines} />
    </div>
    ...
  )
}
```

Queue a re-render!

Updated state is passed in!
Aside: Passing Functions Around

```javascript
render() {
  let text: string = "Hello!";
  return (
    <p>{text}</p>
  
  
}

Notice how these two are *pretty much* equivalent!

render() {
  return (  
    return (  
      <p>Hello!</p>  
    
  
  
}
```
Aside: Passing Functions Around

```javascript
onGreenClick = () => {
    // function body
};

render() {
    return (
        <button onClick={this.onGreenClick}>Green</button>
    
    
};

Similarly, these two are also *pretty much* equivalent!

```javascript
<button onClick={() => {
    // function body
}}>Green</button>
```
Example 6: Children and Props

• We have a new component that puts a title above the Map, called `ColorTitle`.
  – `ColorTitleProps` includes a color that it will display.

• We must include `ColorTitle` in `App`’s render method.

• Current color is passed to child component in `props`.
**Example 6: Children and Props**

We pass in `this.state.color` as the `color prop` of our `ColorTitle` element.

**App.tsx:**
```javascript
render() {
  return (
    <div>
      ...
      <ColorTitle color={this.state.color} />
      ...
    </div>
  );
}
```
Example 6: Children and Props

The `ColorTitle` element takes the `color` prop and displays it!

```javascript
render() {
  return (
    <h1 id="app-title"
        style={{color: this.props.color}}>
      Your favorite color is {this.props.color}!
    </h1>
  );
}
```
Example 7: Callbacks

- We factor out the three buttons into `ButtonGroup`

- `ButtonGroup` uses a `callback` function to notify `App` that a new color has been chosen
  - Remember: `ButtonGroup` is a child of `App`

- `Callback` function is passed in via `props` also
Example 7:

Callbacks

In our `App` component:

```jsx
update_color = (color_: string) => {
  // create newState by getting the old state and modifying
  // it using the color_ parameter, then replacing the old
  // state with our new state!
  this.setState(newState);
}
```

We pass this `update_color` function as a `prop` into our `ButtonGroup` element. This function updates `App`’s `state`.

```jsx
<ButtonGroup onColorChange={this.update_color} />
```
Example 7:

**Callbacks**

In the `ButtonGroup` component:

```javascript
onGreenClick = () => {
  this.props.onColorChange("green");
};

... render() {
  return (
    <div>
      <button onClick={this.onGreenClick}>Green</button>
    </div>
  );
}
```

We pass **information** from `ButtonGroup` to `App` when we call the **callback** function.

When `ButtonGroup`'s button is clicked, it calls `onGreenClick`, which calls the **callback** function that we passed in as a **prop**!
**Example 7:**

**Callbacks**

`update_color` updates App’s state using the information received through the `color_` parameter ("green").

```jsx
render() {
  return (
    <div>
      <ButtonGroup onColorChange={this.update_color} />
      <br />
      <ColorTitle color={this.state.color} />
      <div>
        <Map edgeList={this.state.lines} />
      </div>
    </div>
  );
}
```

When `ButtonGroup`’s button is clicked, it calls `onGreenClick`, which calls the `callback` function that we passed in as a prop, which updates App’s state, and re-renders the `ColorTitle` and `Map` elements using App’s updated state as props!
The Flow

```html
<App />

<Map />
  <MapContainer />
    <MapLine />
    <MapLine />

<ColorTitle />
  <h1>

<ButtonGroup />
  <button>
  <button>
  <button>
```

updateColor()
Aside: `console.log` output

- Kebab menu > More tools > Developer tools

`console.log` will get output here
Using React Developer Tools

- 🌐 Components Tab
  - See the component structure!
  - Verify the props and state!
Components are reusable blocks of code that allow **modular design** and **proper cohesion**.

Components contain other components and HTML tags to determine how they appear on a webpage.

- React is responsible for managing the underlying webpage.

Data owned/controlled by a component is stored in that component’s **state**.

Data flows *down* from parent to child through **props**.

Data flows *up* from child to parent through **callbacks** from the child into the parent’s code.

React notifies components of changes to their data, and re-renders happen accordingly.
Before next lecture...

1. Do HW6 by tonight!
   - No written portion
   - Coding portion (push and tag on GitLab)

2. Feel free to add additional JUnit tests or script tests!