CSE 331
Software Design & Implementation

Topic: Intro to React

💬 Discussion: What website do you visit most frequently?
Reminders

• Pre-req for today: Watch TS Introduction video

Upcoming Deadlines

• Prep. Quiz: HW7 due Monday (7/31)
• HW7 due Thursday (8/03)
Last Time...

• Event-driven Programming
• A Short History of Web
• HTML, CSS, JS
• React, TypeScript

• TS Introduction

Today’s Agenda

• Review
• Examples
  • Simplest React application
  • Character Counter
  • Schedule Picker
Event-driven programming

Register Event

```java
public void myFunction() {
    System.out.println("I was here");
}
button1.addOnClickListener(myFunction);
```

Event loop:

```java
do {
    e = getNextEvent();
    process event e;
} while (e != quit);
```
Event-driven programming

Register Event

```java
public void myFunction() {
    System.out.println("I was here");
}
button1.addOnClickListener(myFunction);
```

Event loop:

```java
do {
    e = getNextEvent();
    process event e;
} while (e != quit);
```
The Allen School

The Allen School is a Computer Science school at UW. The best course in the Allen School is [CSE 331](https://cs.uw.edu/331).
Tags form a Tree

This tree data structure, which lives in the browser, is often called the "DOM" – *Document Object Model*
Demo: Callbacks in JS

```
<html lang="en">
  <head>
    <title>HTML Button</title>
  </head>
  <body>
    <script type="text/javascript">
      function sayHello() {
        alert("Hello, CSE 331!");
      }
    </script>
    <button onclick="sayHello()">Click Me!</button>
  </body>
</html>
```
The Road So Far...

Done:
• First, look at basic HTML on its own
  – No scripting, no dynamic content
  – Just how content/structure is communicated to the browser

• Second, look at basic TypeScript (& JavaScript) on its own
  – No browser, no HTML, just the language
  – Get a feel for what’s different from Java

• Third, a quick look at very basic user interactions
  – Events, event listeners, and callbacks (more depth later)

Now:
• Fourth, use TypeScript with React with HTML
  – Write TypeScript code, using the React library
  – Generates the page content using HTML-like syntax
Reminder: Our Stack

(we write these)

TypeScript
React

HTML Template

Compiled/Combined by the Development Tooling

HTML
JavaScript

(sent to browser to execute)
Making the Jump to React

• Write mostly TS, which is responsible for dynamically generating the HTML on-the-fly.
  • Fundamentally different way of thinking about websites.
  • Allows code reuse (more or less impossible in HTML)
  • Improves modularity.
  • Designed to reduce coupling, increase cohesion. (Yay!)

• The webpage is made up of Components
  • Component = a class that extends the Component class
  • Components contain each other & form a tree structure
    • Just like HTML tags
Tags form a Tree

This tree data structure, which lives in the browser, is often called the "DOM" – *Document Object Model*
Components

• We will have many components
  – e.g. Application, Column, LoginForm, Input, Button
Components

- We will have many components
  - e.g. Application, Column, LoginForm, Input, Button
Components

- We will have many components
  - e.g. Application, Column, LoginForm, Input, Button
Components

- We will have many components
  - e.g. Application, Column, LoginForm, Input, Button
The Contract

• React is "in charge" of the creation of the webpage.
  – It calls methods in your components to do that
  – You override those methods to control the behavior

• React can understand the data used to display the website
  – When data changes, it updates the page (efficiently!)

• You can create multiple components
  – Can reuse a single component multiple times
  – Each component is a single "part" of the webpage
Example 1

- The simplest source code to create a React website is these 3 files:
  - index.html
    - A very small amount of "necessary" HTML
    - Most of the actual web content will be generated by the TS/React code
  - index.tsx
    - Starting point of code – runs when the page loads
    - Starts React
  - App.tsx
    - Our first component – the App component

- When we build the React app, all these files will be incorporated into what is sent to the browser
Example 2

register-react/...
Passing Data from Parent -> Child

- We will have many components
  - e.g. Application, Column, LoginForm, Input, Button
React

- Regain modularity by allowing custom tags

```javascript
let app = (
  <div>
    <TitleBar name="My App"/>
    <EditPane rows="80" />
  </div>);
```

- TitleBar and EditPane can be separate modules
  - their HTML gets substituted in these positions
React

- Custom tags implemented using classes (like TS)
  ```javascript
  class TitleBar extends React.Component {
  ```
- Attributes (name="My App") passed in props arg
- Method `render` produces the HTML for component
- Framework joins all the HTML into one blob
  - can update in a single call to `innerHTML = ...`
Structure of a React Application

- **Model**: Data and invariants
- **Listeners**: Events
- **HTML**: Presentation

Flow:
- Updates from Model to Listeners
- Events from Listeners to HTML
- Presentation from HTML to Model
React State

- Components become dynamic by maintaining state
  - stored in fields of `this.state`
  - call `this.setState({field: value})` to update

- React will respond by calling `render` again
  - will automatically update the HTML to match the HTML produced by this call
React State

- Also very efficient!
  - Tracks updates in virtual DOM
  - Only updates the necessary elements in browser
Callbacks in JS

0 - web page is loaded into browser

1 - JS sayHello function embedded in web page inside <script> tag

2 - Button created on page load; sayHello() function registered to be called on click event

3 - when button is clicked function sayHello() is called and alert box is displayed
Event Listeners

Three ways to do this properly:

1. `onClick={this.handleClick.bind(this)}`

2. `onClick={(e) => this.handleClick(e)}`

3. **Make `handleClick` a prop rather than a method:**

   ```javascript
   handleClick: (e) => { ... };
   ```

   Then `this.handleClick` is okay. (The homework assignment does this)
Example 3

register-react2/...
Passing Data from Child -> Parent

• We will have many components
  – e.g. Application, Column, LoginForm, Input, Button
Structure of Example React App

App

State:
- quarter

onPick

Quarter Picker

Props:
- quarter

State:
- classes

onBack

Class Picker
React State

• Custom tag also has its own events

• Updating data in a parent:
  – sends parent component new data via event
  – parent updates state with `setState`
  – React calls parent’s `render` to get new HTML
    • result can include new children
    • result can include changes to child props
Passing Around Information

• React terminology uses the term **passing in** (instead of registering) a callback function when we supply a function as a prop to a child component.

• We can propagate information upwards from child component.
  – Parent passes down a callback function from a parent component as a prop.
  – When called, the callback function can then update the fields (state) of the parent component from the child component.

Source: www.dotnettricks.com
Splitting the Model

• State should exist in the **lowest common parent** of all the components that need it
  – sent down to children via *props*

• Children change it via *events*
  – sent up to the parent so it can change its state

• Parent’s render creates new children with new props
Structure of a React Application

- **Model**: data and invariants
- **Listeners**: events
- **HTML**: presentation
- **React Components**: updates
Structure of a React Application

• Model must store all data necessary to generate the exact UI on the screen
  – react may call `render` at any time
  – must produce identical UI

• Any state in the HTML components must be mirrored in the model
  – e.g., every text field's `value` must be part of some React component's state
  – `render` produces
    
    `<input type="text" value={...}>`
React setState

- `setState` does not update state instantly:
  
  ```javascript
  // this.state.x is 2
  this.setState({x: 3});
  console.log(this.state.x); // still 2!
  ```

- Update occurs after the event finishes processing
  - `setState` adds a new event to the queue
  - Work is performed when that event is processed

- React can batch together multiple updates
React Gotchas

• render should not have side-effects
  – only read this.state in render

• Never modify this.state
  – use this.setState instead

• Never modify this.props
  – read-only information about parent’s state

• Not following these rules may introduce bugs that will be hard to catch!
React Performance

• React re-computes the tree of HTML on state change
  – can compute a “diff” vs last version to get changes

• Surprisingly, this is not slow!
  – slow part is calls into browser methods
  – pure-JS parts are very fast in modern browsers
  – processing HTML strings is also incredibly fast
React Tools

- Use of compilers etc. means new tool set

- `npm` does much of the work for us
  - installs third-party libraries
  - runs the compiler(s)

- Much more in sections tomorrow...
Before next class...

1. Start on the Prep. Quiz: HW7 to get practice with generics
   - Will need to apply generics and implement Dijkstra’s algorithm

2. If you are uncomfortable with generics, start HW7 early
   - Will need to apply generics
   - Useful for implementing Dijkstra’s algorithm on a `Graph<Double>`