Framework Training
React
London July 2018
Exercise E-state

- This exercise adds **state** to components.
- Install and run the starter project.

```
npm install
npm start
```

Review the current state of the project

- The Shop component render method maps over the basket props to draw up four panels within a FlexBox.

```
{this.props.basket.map((name, n) => {
    return (
        <section className="panel" key={n}>
            <h2>0</h2>
            <h4>{name}</h4>
            <p>Up</p>
            <p>Down</p>
        </section>
    );
})}
```

- We can use **destructuring** to write more concise syntax.

```
let {basket} = this.props;
    {basket.map((item, n) ....}
```

Component composition

- We can use **composition** by moving each item into its own Panel component.

```
<section className="shop">
    {basket.map((name,n) => <Panel key={n} desc={name} /> )}
</section>
```

- Note that **Panel is a stateless component**: a single function, not a class with methods.

```
    let Panel = (props) => {

        let {desc} = props;

        return (
            <section className="panel">
                <h2>0</h2>
                <h4>{desc}</h4>
                <p>Up</p>
                <p>Down</p>
            </section>
        )
    }
```

Generating a random key

- We should use true unique keys, not just a sequence of numbers for the key attribute.
- Add a getKey method which creates a unique key from the item name and a random number.

```
  getKey( s ) {
      return s + "-" + Math.floor( Math.random() * 1024 * 1024
);
  }
```

- Use this method in each Panel instance.

```
<Panel key={this.getKey(name)} desc={name} />
```

Add state to the Panel component.

- We want to add state to the Panel component.
- Clicking the UP or DOWN buttons should change the number displayed.
- The component is currently a stateless function.
- Convert it to a component class with methods.

```
class Panel extends Component {
    render() {
        let {desc} = this.props;
        return (
```

```
            <section className="panel">..</section>
        )
    }
}
```

- Add a constructor method.

```
constructor( props ) {
    super( props );
    console.log( this.props );
}
```

Component state

- State can be only defined in the **constructor**.
- It can be changed indirectly using **setState** in other methods.
- State is defined as an object.
- It is then visible in all methods as **this.state**.

```
constructor( props ) {
super( props );
this.state = { total:0 };
```

}

- We can define an UP method to increase the total.

```
up() {
    let n = this.state.total + 1;
    this.setState({ total: n });
}
```

- Define a DOWN method to decrease the total and avoid minus numbers.

```
  down() {
    let n = Math.max(this.state.total - 1, 0);
    this.setState({ total: n });
  }
```

- Add event-based code to the render method to call these methods when the user clicks UP or DOWN.

```
<p onClick={this.up}>Up</p>
```

Down

- Clicking UP or DOWN causes a runtime error.

- Javascript changes the runtime value of THIS to undefined.
- Expressions like this.state.total cause a run-time error.
- One solution is to **explicitly bind** the run-time value of THIS in the constructor.

```
this.up = this.up.bind( this );
this.down = this.down.bind( this );
```

- To see changes in state, we need to update the render method.

```
        <h2>{ this.state.total }</h2>
```

- We can make this more concise with **destructuring**

```
let {total} = this.state;
<h2>{total}</h2>
```