Kihwan Cho (http://www.kihwancho.com)

Creating a project

Create React App and Typescript are used for this test.

```
npx create-react-app kablamo --typescript
```

Installation

```
yarn install
yarn start
```

Trouble shooting

I made 3 versions of implementation depending on approaches.

Version 1 (StopwatchV1)

- The main issue was all of the handlers used in `onClick` function didn't recognize `this` context. Basically, we can bind `this` context to handler functions using `bind` as the example below.

```
onClick={this.handleStartClick.bind(this)}
```

- In `onDelete` function, we can also pass `index` value after `this` context in order to remove a clicked item.

```
onDelete={this.handleDeleteClick.bind(this, i)}
```

- In `handleDeleteClick` function, because `SampleArray.splice(index, 1)` will return an removed value and `splice` function will mutate an array, we can make a duplication of `this.laps` as the codes below. Otherwise, lodash `remove` function would be handy to maniplate array values.

```
const updated = [...this.laps];
updated.splice(index, 1);

this.laps = updated;
this.forceUpdate();
```

- Also, we need to use `this.forceUpdate();` in order to re-render the component because `this.laps` is a static prop.
- The default value of `lastClearedIncrementer` should be `undefined` instead of `null` in order to render appropriate buttons when we set `not 0` into `initialSeconds`. Otherwise, it will render `stop` button at first render even though `setInterval` is stopped because `this.incrementer === lastClearedIncrementer` condition is `false`. (undefined === null // false)

```
<StopwatchV1 initialSeconds={1} />
```

- In `handleStartClick` function, we can add `if (this.incrementer === this.state.lastClearedIncrementer)` condition in order to prevent double-click or multiple-click. Otherwise, it will generate unreachable `setInterval` functions.

Version 2 (StopwatchV2)

- We can try to avoid using `forceUpdate()` since it is not a natural behavior in React. Even React dev team doesn't recommend using it.

> Normally you should try to avoid all uses of forceUpdate() and only read from this.props and this.state in render().

- In order to avoid `forceUpdate()`, we can set `laps` as a state. Then it will re-render the component when `this.state.laps` changes.

```
this.state = {
  ...
  laps: []
};

...

this.setState({
  laps: updated
});
```

Version 3 (StopwatchV3)

- In order to build solid applications, we need to find out how to improve `reusability`, `readability`, `maintainability` and `testability`. Since React already supports `HOC` and `Function component`, we can follow Functional Programming principles avoiding `shared states` and `state mutations`. Then our applications will be easy to reuse and easy to test.

StopwatchV3.view

- The view component only takes control of rendering and all the dynamic states and handlers will come from `enhancer`. In this case, we can make a test easily because we can inject different context anytime and anywhere.

StopwatchV3.enhancer

- This function will take a view component and return it with all the states and handlers. Using React Hooks APIs such as useState, useRef, we can manage shared states and mutable values in this function.