ECE PARIS
ÉCOLE D'INGÉNIEURS

Composant: événements, rendu conditionnel et listes

2018/2019

APERCU

Dans ce TP, vous découvrirez l'utilisation d'événements au sein d'un composant, ainsi que le rendu conditionnel et la mise en place de listes.

OBJECTIFS

1. Créer un événement
2. Afficher conditionnellement un élément
3. Afficher une liste d'éléments

Dans ce TP, vous améliorerez l'utilisation de votre composant **Clock**.

1. Créer un événement

Tout d'abord, vous allez créer deux boutons dans le composant principal afin de pouvoir y attacher des événements et ainsi contrôler notre horloge. Voici à quoi doit ressembler la méthode **render**:

```
render() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <h1 className="App-title">Welcome to React</h1>
      </header>
      <Clock date={new Date()}/>
      <button>Start</button>
      <button>Stop</button>
    </div>
  );
}
```

Vous allez désormais attacher un événement pour chaque bouton. Pour cela, il suffit simplement d'ajouter la propriété **onClick** à notre élément et d'y associer une fonction. Pour cela, vous allez tout d'abord ajouter l'état "**running**" au composant **App**:

```
constructor(props) {
  super(props);

  this.state = { running: true };
}
```

Créez deux méthodes permettant de modifier l'état:

```
start() {
  this.setState({ running: true });
}

stop() {
  this.setState({ running: false });
}
```

Puis appelez cette méthode dans la propriété **onClick**:

```
<button onClick={this.start}>Start</button>
<button onClick={this.stop}>Stop</button>
```

Attention: Afin de pouvoir utiliser le mot-clef **this** dans les méthodes **start** et **stop** lorsqu'elles sont utilisées en tant que callback, il faut lier **this** grâce à la méthode **bind**. Ajoutez les lignes suivantes dans votre constructeur:

```
this.start = this.start.bind(this);
this.stop = this.stop.bind(this);
```

Maintenant, il faut modifier notre composant **Clock** pour qu'il continue ou s'arrête en fonction de sa nouvelle propriété **run**:

```
componentDidMount() {
  this.timerID = setInterval(() => {
    if (this.props.run) {
      this.setState({
        date: new Date()
      });
    }
  }, 1000);
}
```

2. Afficher conditionnellement un élément

Vous allez désormais afficher un message lorsque le composant **Clock** est arrêté. Pour cela, il faut donc pratiquer l'affichage conditionnel. Vous allez créer deux méthodes renvoyant du JSX dans le composant **Clock**, une qui affichera l'heure, et une qui affichera le message d'arrêt. Puis dans la méthode **render**, l'une des deux méthodes sera appelée en fonction de la propriété **run**:

```
displayHour() {
  return (
    <div>
      {this.state.date.toLocaleTimeString()}
    </div>
  );
}
```

```
displayMessage() {
  return (
    <div>
      Clock is stopped
    </div>
  );
}
```

Et finalement dans la méthode **render**:

```
render() {
  if (this.props.run) {
    return this.displayHour();
  } else {
    return this.displayMessage();
  }
}
```

3. Affichage d'une liste d'élément

Vous allez désormais gérer l'affichage de plusieurs composant **Clock** grâce à un principe de liste. Pour cela, remplacez l'état **running** dans le composant **App** par **listClock** qui contiendra un objet **Array**:

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

Il faut désormais créer deux méthodes afin d'ajouter et de retirer des éléments dans ce tableau. La méthode d'ajout créera un objet ayant deux propriétés, **date** (Date) et **run** (Booléen), puis l'ajoutera dans le tableau. La méthode de suppression retirera le dernier objet créé dans le tableau.

```
add() {
  this.setState({
    listClock: this.state.listClock.concat({ date: new Date(), run: true })
  });
}

remove() {
  this.setState({
    listClock: this.state.listClock.slice(0, -1)
  })
}
```

Maintenant, ajoutez deux boutons qui seront liés à ces deux méthodes:

```
<button onClick={this.add}>New</button>
<button onClick={this.remove}>Delete</button>
```

Attention: N'oubliez pas l'utilisation du **bind** dans le constructeur !

Vous allez désormais afficher des composants **Clock** en fonction de cette liste. Pour cela, créez une méthode qui générera l'affichage des composants:

```
displayClocks() {
  let listItem = this.state.listClock.map((clock, index) =>
    <li key={index}>
      <Clock date={clock.date} run={clock.run}/>
      <button>Start</button>
      <button>Stop</button>
    </li>
  );

  return (<ul>{listItem}</ul>);
}
```

Et finalement, la fonction **render** du composant **App**:

```
render() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <h1 className="App-title">Welcome to React</h1>
      </header>

      {this.displayClocks()}

      <button onClick={this.add}>New</button>
      <button onClick={this.remove}>Delete</button>
    </div>
  );
}
```

Remarque: La propriété **key** est obligatoire lors de l'affichage d'une liste https://reactjs.org/docs/lists-and-keys.html#keys

Mais il faut désormais faire fonctionner correctement les boutons **Start** et **Stop** qui ont été modifiés. Pour cela, rajoutez un paramètre dans les deux méthodes **start** et **stop** permettant de sélectionner le composant visé:

```
start(index) {
  let clocks = this.state.listClock.slice(); // Copy array
  clocks[index].run = true; // Edit array
  this.setState({ listClock: clocks }); // Save array
}

stop(index) {
  let clocks = this.state.listClock.slice(); // Copy array
  clocks[index].run = false; // Edit array
  this.setState({ listClock: clocks }); // Save array
}
```

Et finalement, pour pouvoir les utiliser, modifiez les boutons dans la méthode **displayClocks**:

```
<button onClick={() => {this.start(index)}}>Start</button>
<button onClick={() => {this.stop(index)}}>Stop</button>
```

Remarque: Une fonction flèche est utilisé dans ce cas pour deux raisons: ne pas avoir à **bind** le mot-clef **this** à la fonction, et pouvoir utiliser un paramètre chargé dans le contexte actuel (**index**). Pour plus d'informations, voir le principe de **Closure**: https://javascript.info/closure