ECE PARIS
ÉCOLE D'INGÉNIEURS

Composant: propriétés, état et cycle de vie

2018/2019

APERCU

Dans ce TP, vous découvrirez la création de composant et comment interagir avec. Vous étudierez l'utilisation des propriétés, le changement d'état et le cycle de vie de ces composants.

OBJECTIFS

1. Créer des composants
2. Faire des opérations dans les composants
3. Utiliser les propriétés d'un composant
4. Manier les états d'un composant
5. Gérer le cycle de vie d'un composant

Dans ce TP, vous créerez un composant permettant d'afficher une horloge numérique.

1. Créer un composant

Dans un premier temps, créez un dossier **Clock** dans **src** qui contiendra votre composant. Dans ce dossier, créez le fichier **Clock.js** qui contiendra le code de votre composant. Pour créer un composant, rien de plus simple:

```
import React, { Component } from 'react';

class Clock extends Component {
  render() {
    return (<div></div>);
  }
}
export default Clock;
```

Il suffit simplement d'importer la classe **Component** de la librairie **React** puis de créer une classe **Clock** qui hérite de cette classe. Il est aussi obligatoire de créer une méthode **render** dans la classe **Clock** qui retourne du JSX.

Il suffit ensuite d'importer ce composant dans notre composant principal **App** puis de l'intégrer dans le JSX:

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
import Clock from './Clock/Clock.js';

class App extends Component {
  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 />
      </div>
    );
  }
}
export default App;
```

2. Exécutez une action lors de la construction

Vous allez maintenant créer un objet JavaScript **Date** lors de la construction de la classe **Clock** afin de l'afficher. Pour cela:

```
import React, { Component } from 'react';

class Clock extends Component {
  constructor(props) {
    super(props);
    this.date = new Date();
  }

  render() {
    return (<div>{this.date.toLocaleTimeString()}</div>);
  }
}
export default Clock;
```

Il suffit simplement de surcharger le constructeur de la classe mère **Component** afin de rajouter la propriété **date**. La fonction **super** permet d'appeler le constructeur de la classe mère afin de surcharger sans changer le comportement initial du composant.

3. Utiliser les propriétés d'un composant

Plutôt que de créer une propriété, il est aussi possible de passer des arguments lors de la création de la classe grâce aux **props**. Pour cela, il suffit simplement de l'indiquer dans le JSX:

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
import Clock from './Clock/Clock.js';

class App extends Component {
  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()}/>
      </div>
    );
  }
}
export default App;
```

Puis, mettre à jour la classe **Clock** pour afficher la propriété:

```
import React, { Component } from 'react';

class Clock extends Component {
  render() {
    return (<div>{this.props.date.toLocaleTimeString()}</div>);
  }
}
export default Clock;
```

Remarque: Les propriétés d'un composants (**props**) sont en lecture seule ! Une fois la classe initialisée, ses propriétés ne peuvent pas changer.

4. Manier les états d'un composant

L'horloge affiche désormais l'heure mais ne s'actualise pas avec le temps. La date étant stockées dans une propriété, il est impossible de la changer. Pour cela, il existe "les états" (appelés **states**).

Pour associer une value à une variable d'état, il suffit d'utiliser la méthode **setState** qui prend un objet en paramètre, sauf dans le constructeur où il est possible d'initialiser l'objet **state** directement. Pour utiliser une variable d'état, il suffit de l'appeler grâce à l'objet **state** dans la classe. **Attention:** Il est interdit de changer la valeur d'un état autrement que par la méthode **setState** en dehors du constructeur !

```
import React, { Component } from 'react';

class Clock extends Component {
  constructor(props) {
    super(props);
    this.state = { date: this.props.date }
  }

  render() {
    return (
      <div>
        {this.state.date.toLocaleTimeString()}
      </div>
    );
  }
}
export default Clock;
```

Remarque: Lorsque l'état d'un composant est modifié, alors la fonction **render** est appelée automatiquement afin de mettre à jour l'affichage du composant.

5. Gérer le cycle de vie d'un composant

La date est désormais stockée dans une variable d'état mais elle ne se met pas à jour automatiquement. Pour cela, vous allez exécuter un chronomètre pour mettre à jour la date lors de la création du composant puis stopper le chronomètre lorsque le composant est détruit.

Lorsqu'un composant est affiché à l'écran dans React, on dit qu'il est "monter" ("mounting"), et lorsqu'il est détruit car il n'y a plus besoin de l'afficher, on dit qu'il est "démonter" ("unmounting"). Il est possible de surcharger deux méthodes afin d'exécuter des actions lorsqu'un composant est monté ou démonté:

```
import React, { Component } from 'react';

class Clock extends Component {
  constructor(props) {
    super(props);
    this.state = { date: this.props.date }
  }

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

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  render() {
    return (
      <div>
        {this.state.date.toLocaleTimeString()}
      </div>
    );
  }
}
export default Clock;
```

La méthode **componentDidMount** est exécutée lorsque le composant est affiché à l'écran et permet de créer un interval qui met à jour la variable d'état **date** toute les secondes.

La méthode **componentWillUnmount** est exécutée lorsque le composant n'est plus affiché à l'écran et permet de stopper l'interval.