



WarsawJS Workshop #57

Wstęp do React.js

Jarosław Kowalczyk
GitHub: Wyxuch



Klasy

Klasy są wzorem na
podstawie którego
tworzone są
obiekty

```
class templateClass {  
  constructor(arg1, arg2) {  
    this.arg1 = arg1;  
    this.arg2 = arg2;  
  }  
  
  method1() {  
    console.log(this.arg1)  
  }  
  
  method2() {  
    console.log(this.arg2)  
    // do something else  
  }  
}
```

```
const object = new templateClass('arg1', 'arg2')  
object.method1() // displays "arg1"
```

Klasy można rozszerzać o nowe argumenty albo metody lub je nadpisać

```
class extendedTemplateClass extends templateClass{
  constructor(arg1, arg2, arg3) {
    super(...arguments);
    this.arg3 = arg3
  }

  method2() {
    console.log(this.arg3)
    // do something else
  }
}
```

```
const extendedObject = new extendedTemplateClass('arg1', 'arg2', 'arg3')
extendedObject.method1() // displays "arg1"
extendedObject.method2() // displays "arg3"
```

Czas na zadania

Spread

Tablice

```
const array1 = [1, 2, 3];  
const array2 = ['a', 'b', 'c'];  
const arg1 = 'arg1';  
  
const concat = [...array1, ...array2, arg1]  
console.log(concat) // [1, 2, 3, 'a', 'b', 'c', 'arg1']
```

Obiekty

```
const obj1 = {  
  arg1: 1,  
  arg2: 2,  
};
```

```
const obj2 = {  
  ...obj1,  
  arg2: 'two',  
  arg3: 3  
}
```

```
console.log(obj2)
```

Console.log zwróci

```
{  
  arg1: 1,  
  arg2: 'two',  
  arg3: 3  
}
```



```
const obj2 = obj1
```

!=

```
const obj2 = {...obj1}
```

Pass by reference

Kopiuje obiekt

Czas na zadania

Destrukturyzacja


```
const arg1 = 'arg1';  
const arg2 = 'arg2';
```



```
const [arg1, arg2] = ['arg1', 'arg2']
```

```
const data = {  
  first: 1  
}
```

```
const displayFirst = (data) => {  
  console.log(data.first)  
}
```



```
const data = {  
  first: 1  
}
```

```
const displayFirst = ({ first }) => {  
  console.log(first)  
}
```

Można też tak

```
const [arg1, arg2, ...rest] = ['arg1', 'arg2', 'arg3', 'arg4'];  
console.log(arg1) // arg1  
console.log(arg2) // arg2  
console.log(rest) // ['arg3', 'arg4']
```

Czas na zadania

Czym jest React?

Najważniejsze zalety React'a

- State management
- Łatwy podział na komponenty
- Refresh komponentów
- Masa bibliotek i rozszerzeń
- Szybki i łatwy setup nowego projektu
- Przystosowany do używania TypeScript'a

Komponenty klasowe

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      arg1: 0,
      array: [1, 2, 3]
    };

    this.handleClick = this.handleClick.bind(this)
  }

  arg2 = 'argument'

  componentDidMount() {
    //do something
  }

  componentDidUpdate(prevProps, prevState, snapshot) {
    //do something
  }

  handleClick() {
    this.setState({...this.state, arg1: this.state.arg1 + 1})
  }

  render() {
    return (
      <div>
        <button onClick={this.handleClick}></button>
        <p>{this.arg2}</p>
        <p>{this.state.arg1}</p>
      </div>
    )
  }
}

export default App;
```

Import komponentów



App.jsx

```
import React from 'react';
import Add from './components/Add'

import './App.css';

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      counter: 0
    };

    this.add = this.add.bind(this)
  }

  add() { // setter
    this.setState({counter: this.state.counter + 1})
  }

  render() {
    return (
      <div>
        <Add add={this.add} />
        <p>{this.counter}</p>
      </div>
    )
  }
}

export default App;
```

Add.jsx

```
import React from 'react';

class Add extends React.Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <button onClick={this.props.add}>
        Add
      </button>
    )
  }
}

export default Add;
```

Renderowanie dynamicznych elementów i conditionals

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      hidden: true,
      tasks: ['task1', 'task2', 'task3']
    };
  }

  render() {
    return (
      <div className={`tasks ${this.state.hidden && 'hidden'}`} >
        {
          this.state.tasks.map((task, index) => (
            <div key={`task${index}`} >
              <p>{task}</p>
            </div>
          ))
        }
      </div>
    )
  }
}
```

Czas na zadania

Komponenty funkcyjne

Obydwa te elementy działają tak samo

App klasowy

```
import React from 'react';
import Add from './components/Add'

import './App.css';

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      counter: 0
    };

    this.add = this.add.bind(this)
  }

  add() { // setter
    this.setState({counter: this.state.counter + 1})
  }

  render() {
    return (
      <div>
        <Add add={this.add} />
        <p>{this.counter}</p>
      </div>
    )
  }
}

export default App;
```

App funkcyjny

```
import React, { useState } from 'react';
import Add from './components/Add'

import './App.css';

const App = (props) => {
  const [counter, setCounter] = useState(0)

  return (
    <div>
      <Add add={setCounter} />
      <p>{counter}</p>
    </div>
  )
}

export default App;
```

Dwa najbardziej podstawowe Hooks

```
const [value, setter] = useState(0)
```

- pozwala używać state'u w komponentach funkcyjnych

```
useEffect(() => {  
    if(prop) {  
        // Do something  
    }  
}, [prop]) // Na jaki argument powinien reagować
```

- Zastępuje metody z cyklu życia komponentu.
- Może reagować na różne zmienne

Czas na projekt!

Animacje

```
@keyframes fromLeft {  
  from {  
    left: -100vw;  
    transform: scaleX(2);  
  }  
  to {  
    transform: scaleX(1);  
    left: 0;  
  }  
}
```

Dodaj do elementu

```
#element {  
  animation: drop linear 0.3s;  
}
```

Lub

```
@keyframes fromLeft {  
  0% {  
    left: -100vw;  
    transform: scaleX(2);  
  }  
  50% {  
    background-color: white;  
  }  
  100% {  
    transform: scaleX(1);  
    left: 0;  
  }  
}
```

Dodaj do elementu

```
#element {  
  animation: drop linear 0.3s;  
}
```

Do zobaczenia!