Javascript / React - Object Oriented Overview

Object oriented programming introduction::
https://www.w3schools.com/java/java_oop.asp

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

OOP is faster and easier to execute
OOP provides a clear structure for the programs
OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
OOP makes it possible to create full reusable applications with less code and shorter development time

Code-reuse

- Class template
- Object instance
- this keyword
- Constructor
- Encapsulation
    - Data/method hiding
    - Private/method fields
    - Public fields
    - static /shared field
- Inheritance - code reuse
- Polymorphism

    - Javascript object initializer syntax
        - Function constructor
    - Javascript Class keyword syntax

React Class / Functional programming example

.....
object literal
.....

```
var john = {
name: "John",
age: 30,
greet: function() {
console.log("Hello, my name is " + this.name + " and I am " + this.age + " years old.");
}
};

john.greet(); // outputs "Hello, my name is John and I am 30 years old."
```

.....
function constructor
.....

```
function Person(name, age) {
  this.name = name;
  this.age = age;

  this.greet = function() {
    console.log("Hello, my name is " + this.name + " and I am " + this.age + " years old.");
  }
}

var john = new Person("John", 30);
john.greet(); // outputs "Hello, my name is John and I am 30 years old."
```

.....
class syntax
.....

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  greet() {
    console.log("Hello, my name is " + this.name + " and I am " + this.age + " years old.");
  }
}

var john = new Person("John", 30);
john.greet(); // outputs "Hello, my name is John and I am 30 years old."
```

.....
inheritance
.....

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  greet() {
    console.log("Hello, my name is " + this.name + " and I am " + this.age + " years old.");
  }
}

class Employee extends Person {
  constructor(name, age, salary) {
    super(name, age);
    this.salary = salary;
  }

  displaySalary() {
    console.log("My salary is " + this.salary);
  }
}
```

```
var john = new Employee("John", 30, 50000);
john.greet(); // outputs "Hello, my name is John and I am 30 years old."
john.displaySalary(); // outputs "My salary is 50000"
```

.....

static method

.....

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  static hello() {  // static method
    return "Hello!!";
  }
}

mycar = new Car("Ford");

//Call 'hello()' on the class Car:
document.getElementById("demo").innerHTML = Car.hello();

//and NOT on the 'mycar' object:
//document.getElementById("demo").innerHTML = mycar.hello();
//this would raise an error.
```

—----------

React functional programming example

—----------

```
//--- Login.js ---

import { useState, useRef, useEffect } from "react";
import { Link, useNavigate } from "react-router-dom";

const Login = (props) => {

    const [msg, setMsg] = useState("");
    const txtusername = useRef("");
    const txtpassword = useRef("");

    let navigate = useNavigate();

    useEffect(()=>{

        //component mound/load

        return () =>{
            //component unmount/unload
        }
    },[]);

    const handleLogin = (e) =>{

        const _function_name = "handleLogin";
        let _msg = "";

        try {

            const _uid = txtusername.current.value;
            const _pwd = txtpassword.current.value; //optionally: encrypt password

            if(_uid === null || _uid === undefined || _uid.trim().length === 0)
            {
                _msg = "* invalid username";
            }


            if(_msg.length > 0)
```

```
            {
                setMsg(_msg)
                return false;
            }

            fetch(_url)
            .then((res)=>res.json())
            .then((data)=> {

                if(data.login === true)
                {
                    props.setUser(_uid);
                    navigate("/dashboard",{replace:true})
                }

                setMsg(data.msg)
            })
            .catch((error)=>{
                setMsg("* request error");
                console.log("* request error *");
                console.log(error);
            });

        } catch (error) {
            console.log(`** ${_function_name}::error`)
            console.log(error)
        }
    }

    return (
    <>
        <p>Login Page - v1.0.8</p>
        <div>
            <span>* username: </span><input ref={txtusername} type="text" maxLength={20}
placeholder="* username required"/> <br/>
            <span>* password: </span><input ref={txtpassword} type="password" maxLength={10}
placeholder="* password required"/> <br/>
            <p>{msg}</p>
            <p></p><button onClick={handleLogin}>login</button>
        </div>
        <p></p>
        <Link to="/register">register</Link>
    </>
    )
```

```
}

export default Login;
```

—----------

React Class example

—----------

```
class UserProfile extends React.Component {
  constructor(props) {
    super(props)
  }

  render(){
    const { name, age } = this.props
    return (
      <section>
        <div>UserName: {name}</div>
        <div>Age: {age}</div>
      </section>
    )
  }
}
```

……..

```
class App extends React.Component {
  constructor(props) {
    super(props)

    state = {
      likes: 0,
    }
  }

  componentWillMount() {
    console.log("Component want's to mount");
  }

  componentDidMount() {
    console.log('Component did mount');
  }

  componentDidUpdate(prevProps, prevState) {
    if (prevState.likes !== this.state.likes) {
      console.log('Likes got updated:', this.state.likes);
    }
  }

  render(){
    return <div>Hello world</div>
  }
}
```

https://dev.to/hasidicdevs/mastering-object-oriented-programming-in-javascript-best-practices-and-examples-31mc

.....

OOP in JavaScript: Encapsulation, Inheritance, Polymorphism, Abstraction, and Association

https://medium.com/codex/oop-in-javascript-encapsulation-inheritance-polymorphism-abstraction-and-association-2cbcd93bbb4f
.....

https://www.w3schools.com/react/react_class.asp

https://medium.com/@emmanuelodii80/everything-you-need-to-know-about-class-components-in-react-ae790220bdad

```
class Container extends React.Component {
  constructor(props) {
    super(props);
    this.state = {show: true};
  }
  delHeader = () => {
    this.setState({show: false});
  }
  render() {

    let myheader;

    if (this.state.show) {
      myheader = <Child />;
    };

    return (

      <div>
      {myheader}
      <button type="button" onClick={this.delHeader}>Delete Header</button>
      </div>
    );
  }
}
```

```
class Child extends React.Component {
  componentWillUnmount() {
    alert("The component named Header is about to be unmounted.");
  }
  render() {
    return (
      <h1>Hello World!</h1>
    );
  }
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Container />);
```