Composition vs Inheritance

* Inheritance

Inheritance is a way to achieve code reusability when some objects have the same number of properties that can be shared across the app. Inheritance allows the app to do the coupling between the parent-child components and reuse properties such as state values and function in its child components.

React does not use inheritance except in the initial component class, which extends from the react package.

Inheritance uses the keyword extends to allow any component to use the properties and methods of another component connected with the parent. Using the extends keyword, you can allow the current component to access all the component's properties, including the function, and trigger it from the child component.

This example creates one component called **ParentClass.jsx**.

import React from "react";

class ParentClass extends React.Component {

constructor(props) {

super(props);

this.callMe = this.callMe.bind(this);

}

// ParentClass function

callMe() {

console.log("This is a method from parent class");

}

render() {

return false;

}

}

ParentClass extends the component from React as React.component, which means the newly created component itself is using the inheritance. After creating parent class/component, create one child component, **Example.jsx**.

export default class Example extends ParentClass {

constructor() {

super();

}

render() {

this.callMe();

return false;

}

}

The Example class extends ParentClass so the child class will access all the properties and methods created inside the parent component.

render() {

this.callMe();

return false;

}

Here in the child class, the **this.callMe()** function is called the part of parent class implementation, so the parent component's properties and its methods can be accessed by implementing inheritance in the child component.

* Using composition in React with Example

React has a powerful composition model, and we recommend using composition instead of inheritance to reuse code between components

Compared to the inheritance, the composition model is much more robust and less coupled because it does inheritance but doesn't extend the classes directly.

The composition model uses the parent-child relationship by passing the state, props, and function to the child component as a prop's value. The child component can access or trigger any changes to each other. For example, you have two-component ParentComponent and a ChildComponent, and the parent's properties get consumed in the child component.

Now create one file called **ParentComponent.jsx**.

import React from "react";

import ChildComponent from "./ChildComponent";

class ParentComponent extends React.Component {

constructor(props) {

super(props);

this.state = {

message: "This is a message from parent class"

};

}

render() {

return (

<div>

<ChildComponent message={this.state.message} />

</div>

);

}

}

export default ParentComponent;

From the parent component, the value this.state.message is passed to the child component without extending any component.

Create one file called **ChildComponent.jsx.**

import React from "react";

class ChildComponent extends React.Component {

render() {

const { message } = this.props;

return (

<div>

<p>

Message from Parent component : <b>{message}</b>

</p>

</div>

);

}

}

export default ChildComponent;

The child component accesses the parent component's property called message without extending the ParentComponent.

In the same way, any function can also be triggered by the child component.

clickMe() {

console.log("Action triggered from child component");

}

And pass it to the child component.

render() {

return (

<div>

<ChildComponent message={this.state.message} clickMe={this.clickMe} />

</div>

);

}

The function gets accessed as a prop from the child component and can also get triggered as below.

import React from "react";

class ChildComponent extends React.Component {

render() {

const { message } = this.props;

return (

<div>

<p>

Message from Parent component : <b>{message}</b>

</p>

<button onClick={this.props.clickMe}>Click Me</button>

</div>

);

}

}

export default ChildComponent;

The function clickMe() is triggered by the button click event, and the parent component will process the event.

<button onClick={this.props.clickMe}>Click Me</button>

The composition model allows you to restrict code overuse and decouple components, which cannot be possible with inheritance because it needs all the components in sequence. It may lead to code overuse or decomposition at any given point in time.

* Practice on Composition & Inheritance

Composition and inheritance are the approaches to use multiple components together in React.js . This helps in code reuse. React recommend using composition instead of inheritance as much as possible and inheritance should be used in very specific cases only.

Example to understand it −

Let’s say we have a component to input username.

**Inheritance**

class UserNameForm extends React.Component {

   render() {

      return (

         <div>

            <input type="text" />

         </div>

      );

   }

}

ReactDOM.render(

   < UserNameForm />,

   document.getElementById('root'));

This sis simple to just input the name. We will have two more components to create and update the username field.

With the use of inheritance we will do it like −

class UserNameForm extends React.Component {

   render() {

      return (

         <div>

            <input type="text" />

         </div>

      );

   }

}

class CreateUserName extends UserNameForm {

   render() {

      const parent = super.render();

      return (

         <div>

            {parent}

            <button>Create</button>

         </div>

      )

   }

}

class UpdateUserName extends UserNameForm {

   render() {

      const parent = super.render();

      return (

         <div>

            {parent}

            <button>Update</button>

         </div>

      )

   }

}

ReactDOM.render(

   (<div>

      < CreateUserName />

      < UpdateUserName />

   </div>), document.getElementById('root')

);

We extended the UserNameForm component and extracted its method in child component using super.render();

**Composition**

class UserNameForm extends React.Component {

   render() {

      return (

         <div>

            <input type="text" />

         </div>

      );

   }

}

class CreateUserName extends React.Component {

   render() {

      return (

         <div>

            < UserNameForm />

            <button>Create</button>

         </div>

      )

   }

}

class UpdateUserName extends React.Component {

   render() {

      return (

         <div>

            < UserNameForm />

            <button>Update</button>

         </div>

      )

   }

}

ReactDOM.render(

   (<div>

      <CreateUserName />

      <UpdateUserName />

   </div>), document.getElementById('root')

);

Use of composition is simpler than inheritance and easy to maintain the complexity.

* Difference b/n Composition & Inheritance

|  |  |
| --- | --- |
| Composition | Inheritance |
| Follows has-a relationship | Follows is-a relationship |
| Composition allows code-reuse. | Inheritance does not allow code-reuse. |
| In composition, you will no need to extend classes | In inheritance, you will need to extend classes. |
| In composition, you will no need to Mixin. | In inheritance, Mixin plays a major role |
| The composition is more flexible. | Inheritance is less flexible compared to composition. |