8 - Let's get Classy

28 June 2023 07:43

Functional Code for the user

```
src > components > User.js > User
const User = () =>{
    return <div className="user-card">
        <h3>Name: Rizon Kumar Rahi</h3>
        <h3>Location: Bengaluru</h3>
        <h3>Contact: rizon.kumar.rahi@gmail.com</h3>
    </div>
}

export default User;
```

```
src > components > About.js > ...
import User from "./User";

const About = () => {
  return (
    <div>
      <h1>About</h1>
      <h2>This is Namaste React Web Series</h2>
      <User/>
    </div>
  );
};

export default About;
```

```
index.css > .user-card
.user-card{
    padding: 10px;
    border: 1px solid black;
}
```

Class components code for the same:

Class component is normal javascript class.

Syntax:

class NameOfTheComponent extends React.Component{ }

↓

this will make react to know it's a class based component

```
src > components > UserClass.js > UserClass
class UserClass extends React.Component{
    → inside this it will have render method (render())
}
```

which will display into the UI.

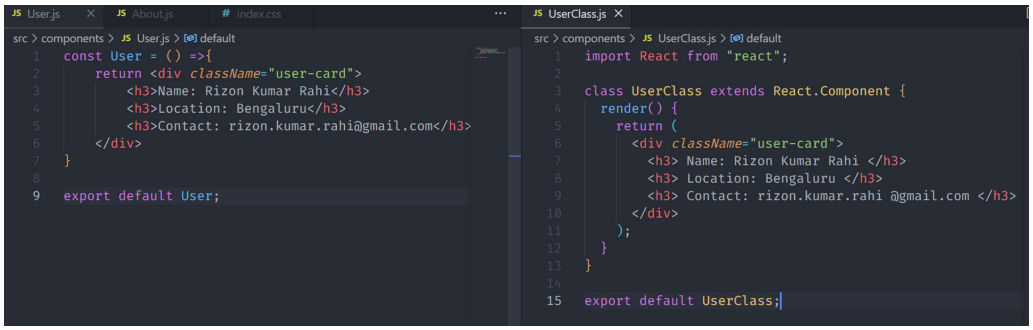
Functional components is a f() which will return a

pieces of JSX.

Class Based component is a class which extends the React.Component & it has render method that return some pieces of JSX.

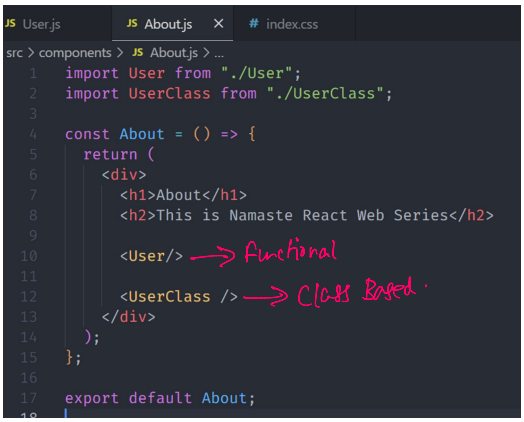
```
src > components > UserClass.js > default
import React from "react";

class UserClass extends React.Component {
  render() {
    return (
      <div className="user-card">
        <h3> Name: Rizon Kumar Rahi </h3>
        <h3> Location: Bengaluru </h3>
        <h3> Contact: rizon.kumar.rahi @gmail.com </h3>
      </div>
    );
  }
}

export default UserClass;
```

Functional Component VS Class Based Component

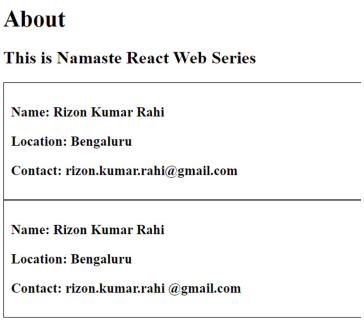


```
src > components > User.js > default
const User = () =>{
    return <div className="user-card">
        <h3>Name: Rizon Kumar Rahi</h3>
        <h3>Location: Bengaluru</h3>
        <h3>Contact: rizon.kumar.rahi@gmail.com</h3>
    </div>
}

export default User;
```

```
src > components > UserClass.js > default
import React from "react";

class UserClass extends React.Component {
  render() {
    return (
      <div className="user-card">
        <h3> Name: Rizon Kumar Rahi </h3>
        <h3> Location: Bengaluru </h3>
        <h3> Contact: rizon.kumar.rahi @gmail.com </h3>
      </div>
    );
  }
}

export default UserClass;
```



```
src > components > About.js > ...
import User from "./User";
import UserClass from "./UserClass";

const About = () => {
  return (
    <div>
      <h1>About</h1>
      <h2>This is Namaste React Web Series</h2>

      <User/> -> Functional

      <UserClass /> -> Class Based.
    </div>
  );
};

export default About;
```



About

This is Namaste React Web Series

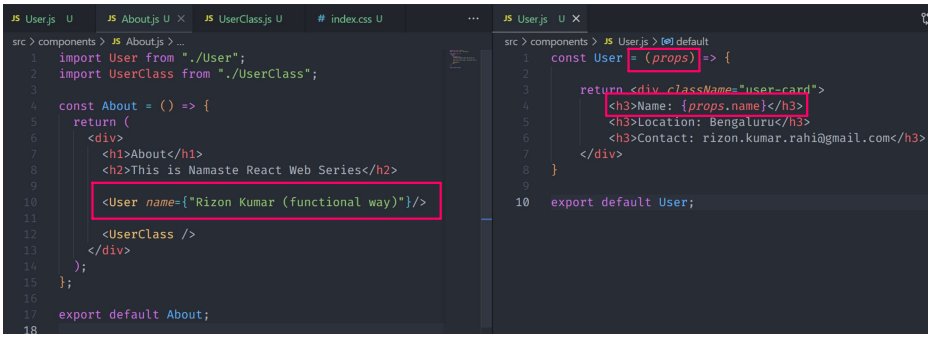
Name: Rizon Kumar Rahi
Location: Bengaluru
Contact: rizon.kumar.rahi@gmail.com

Name: Rizon Kumar Rahi
Location: Bengaluru
Contact: rizon.kumar.rahi @gmail.com

Passing Props:

In functional way:

Now suppose I want to pass name and location as a prop and receive it as a functional component.



```
// About.js
import User from "./User";
import UserClass from "./UserClass";

const About = () => {
  return (
    <div>
      <h1>About</h1>
      <h2>This is Namaste React Web Series</h2>

      <User name={"Rizon Kumar (functional way)"}/>

      <UserClass />
    </div>
  );
};

export default About;
```

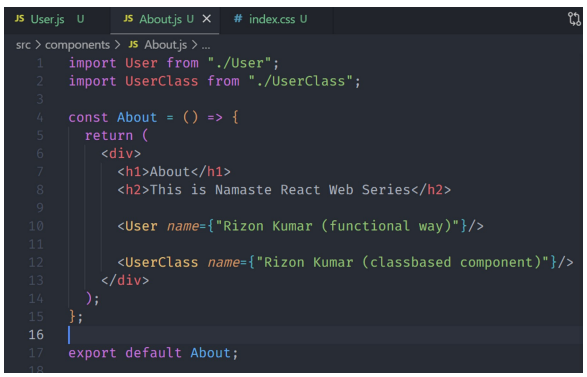
```
// User.js
const User = (props) => {

    return <div className="user-card">
        <h3>Name: {props.name}</h3>
        <h3>Location: Bengaluru</h3>
        <h3>Contact: rizon.kumar.rahi@gmail.com</h3>
    </div>
}

export default User;
```

we can destructure.


```
// About.js
import User from "./User";
import UserClass from "./UserClass";

const About = () => {
  return (
    <div>
      <h1>About</h1>
      <h2>This is Namaste React Web Series</h2>

      <User name={"Rizon Kumar (functional way)"}/>

      <UserClass />
    </div>
  );
};

export default About;
```

```
// User.js
const User = ({name}) => {

    return <div className="user-card">
        <h3>Name: {name}</h3>
        <h3>Location: Bengaluru</h3>
        <h3>Contact: rizon.kumar.rahi@gmail.com</h3>
    </div>
}

export default User;
```

Class based component:



```
// About.js
import User from "./User";
import UserClass from "./UserClass";

const About = () => {
  return (
    <div>
      <h1>About</h1>
      <h2>This is Namaste React Web Series</h2>

      <User name={"Rizon Kumar (functional way)"}/>

      <UserClass name={"Rizon Kumar (classbased component)"}/>
    </div>
  );
};

export default About;
```

- Now to receive it we need a constructor.
- Constructor will receive the props

- then we use keyword super(props)

```
src > components > UserClass.js > ...
import React from "react";

class UserClass extends React.Component {

  constructor(props){
    super(props);

    console.log(props);

  }
  render() {
    return (
      <div className="user-card">
        <h3> Name: {this.props.name} </h3>
        <h3> Location: Bengaluru </h3>
        <h3> Contact: rizon.kumar.rahi@gmail.com </h3>
      </div>
    );
  }
}
export default UserClass;
```

```
▼ Object
    name: "Rizon Kumar (classbased component)"
  ▶ [[Prototype]]: Object
```

- You have to use this keyword in your class, so that props can be accessed anywhere in the class. to get accessed anywhere we use super(props) study about it more.

Name: Rizon Kumar (functional way)

Location: Bengaluru

Contact: rizon.kumar.rahi@gmail.com

Name: Rizon Kumar (classbased component)

Location: Bengaluru

Contact: rizon.kumar.rahi@gmail.com

passing more props.

```
src > components > About.js > ...
import User from "./User";
import UserClass from "./UserClass";

const About = () => {
  return (
    <div>
      <h1>About</h1>
      <h2>This is Namaste React Web Series</h2>

      <User name={"Rizon Kumar (functional way)"}/>

      <UserClass name={"Rizon Kumar (classbased component)"}
       location={"Bengaluru Class"} contact = {"rizon.kumar.rahi@gmail.com"}/

    </div>
  );
};

export default About;
```

```
src > components > UserClass.js > default
1  import React from "react";
2
3  class UserClass extends React.Component {
4
5    constructor(props){
6      super(props);
7
8      console.log(props);
9
10   }
11   render() {
12     return (
13       <div className="user-card">
14         <h3> Name: {this.props.name }</h3>
15         <h3> Location: {this.props.location}</h3>
16         <h3> Contact: {this.props.contact} </h3>
17       </div>
18     );
19   }
20 }
21
22 export default UserClass;
```

Name: Rizon Kumar (functional way)

Location: Bengaluru

Contact: rizon.kumar.rahi@gmail.com

Name: Rizon Kumar (classbased component)

Location: Bengaluru Class

Contact: rizon.kumar.rahi@gmail.com

Destructing it :

```
src > components > UserClass.js > default
1  import React from "react";
2
3  class UserClass extends React.Component {
4
5    constructor(props){
6      super(props);
7
8      console.log(props);
9
10   }
11   render() {
12     const{name, location, contact} = this.props;
13     return (
14       <div className="user-card">
15         <h3> Name: {name }</h3>
16         <h3> Location: {location}</h3>
17         <h3> Contact: {contact} </h3>
18       </div>
19     );
20   }
21 }
22
23 export default UserClass;
```

Creating state inside class based components: (State Variable)

in functional components :

```
src > components > User.js > default
1  import { useState } from "react";
2
3  const User = ({name, location, contact}) => {
4      const [count] = useState(0);
5
6      return <div className="user-card">
7          <h1>Count = {count}</h1>
8          <h3>Name: {name}</h3>
9          <h3>Location: {location}</h3>
10         <h3>Contact: {contact}</h3>
11     </div>
12 }
13
14 export default User;
```

in class based components :

- A state is created when instances of class is created.
- invoking a functional component i.e. you are loading/ mounting a f() into a webpages i.e. known as mounting/ invoking a f().
- creating a instance of class based components means loading a class based components.

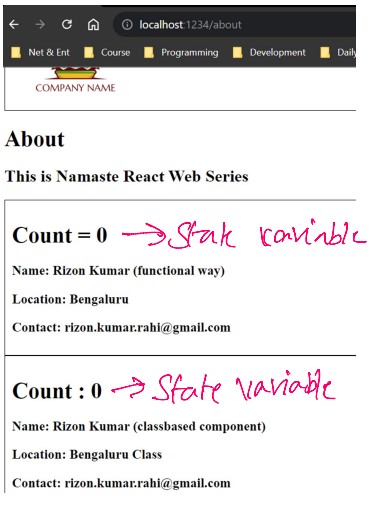
  that means whenever class based components called constructor is called & that the best place to receive the props, create a state variable.

```
constructor (props){
    super (props);
    this.props = {
        count : 0,
    };
}
```

super (props); → big whole object which contains state variable.

count : 0, → inital value

```
UserClass.js U
src > components > UserClass.js > UserClass > render
import React from "react";

class UserClass extends React.Component {

  constructor(props){
    super(props);
    this.state = {
      count: 0,
    };
  }

  render() {
    const{name, location, contact} = this.props;
    return (
      <div className="user-card">
        {/* THis is how you use count */}
        <h1>Count : {this.state.count}</h1>
        <h3> Name: {name }</h3>
        <h3> Location: {location}</h3>
        <h3> Contact: {contact} </h3>
      </div>
    );
  }
}

export default UserClass;
```



→ State variable

→ State variable

Destructing :

```
src > components > UserClass.js > UserClass > render
1  import React from "react";
2
3  class UserClass extends React.Component {
4
5    constructor(props){
6      super(props);
7      this.state = {
8        count: 0,
9      };
10   }
11
12   render() {
13     const{name, location, contact} = this.props;
14     const{count} = this.state;
15
16     return (
17       <div className="user-card">
18         <h1>Count : {count}</h1>
19         <h3> Name: {name }</h3>
20         <h3> Location: {location}</h3>
21         <h3> Contact: {contact} </h3>
22       </div>
23     );
24   }
25 }
26
27 export default UserClass;
```

Now creating '2' state variable:

Functional way:

```
src > components > User.js > default
1  import { useState } from "react";
2
3  const User = ({name, location, contact}) => {
4    const [count] = useState(0);
5    const [count2] = useState(1);
6
7      return <div className="user-card">
8          <h1>Count = {count}</h1>
9          <h1>Count2 = {count2}</h1>
10         <h3>Name: {name}</h3>
11         <h3>Location: {location}</h3>
12         <h3>Contact: {contact}</h3>
13     </div>
14 }
15
16 export default User;
```

→ BTS React create a one big object & put all the state variable.

Class based way:

```
src > components > UserClass.js > UserClass > render > count1
1  import React from "react";
2
3  class UserClass extends React.Component {
4
5    constructor(props){
6      super(props);
7      this.state = {
8        count: 0,
9        count1: 1,
10     };
11   }
12
13   render() {
14     const{name, location, contact} = this.props;
15     const{count, count1} = this.state;
16
17     return (
18       <div className="user-card">
19         <h1>Count : {count}</h1>
20         <h1>Count1 : {count1}</h1>
21         <h3> Name: {name }</h3>
22         <h3> Location: {location}</h3>
23         <h3> Contact: {contact} </h3>
24       </div>
25     );
26   }
```

→ Destructure

How we can update this state variable :

Class Based Way :

- We create a button, which has onClick event & it takes call back function
- On click of the button, we update our state variable.

```
15    return (
16      <div className="user-card">
17        <h1>Count : {count}</h1>
18        <button
19          onClick={() => {
20            this.state.count = this.state.count + 1;
21          }}
22        >
23          Count Increase
24        </button>
```

this is wrong, we don't update our state variable directly.

- Never update the state variable directly.
- React give access to "this.setState()" to do that

we will pass object.

You can use anywhere in the class state.

```
JS UserClass.js U ×
src > components > JS UserClass.js > UserClass > render
1   import React from "react";
2
3   class UserClass extends React.Component {
4     constructor(props) {
5       super(props);
6       this.state = {
7         count: 0,
8       };
9     }
10
11    render() {
12      const { name, location, contact } = this.props;
13      const { count } = this.state;
14
15      return (
16        <div className="user-card">
17          <h1>Count : {count}</h1>
18          <button
19            onClick={() => {
20              // Never update the state variable directly
21              this.setState({
22                count: this.state.count + 1,
23              })
24            }}
25          >
26            Count Increase
27          </button>
```

React will render

Count : 3

Count Increase

Name: Rizon Kumar (classbased component)

Location: Bengaluru Class

Contact: rizon.kumar.rahi@gmail.com

To update the count 2,

```
return (
  <div className="user-card">
    <h1>Count : {count}</h1>
    <button
      onClick={() => {
        // Never update the state variable directly
        this.setState({
          count: this.state.count + 1,
          count2: this.state.count2 + 1,
        })
      }}
    >
      Count Increase
    </button>
```

Life cycle of React :

loading == mounting

When you render about page → (Functional Based Component)

About.js → Parent component is render and go line by line once it's see UserClass component, now new instance of class is created & constructor is called once it's called then render is called.

Constructor → Render

Now suppose if About.js (Class Based Component) (Parent

UserClass → Children

then the way it's called is.

Suppose this is the code:

About.js

```
class About extends Component {
  constructor (props) {
    super (props)
    console.log ("Parent Constructor")
  }
  render () {
    console.log (" Parent Render")
    return (
      <div>
        <UserClass/>
      </div>
    )
  }
}
```

UserClass.js

```
class About extends Component {
  constructor (props) {
    super (props)
    console.log (" Child Constructor")
  }
  render () {
    console.log (" Child Render")
    return (
      <div>
        ...
      </div>
    )
  }
}
```

Life cycle :

Parent Constructor → Parent Render → Child Constructor → Child Render

Class Based Component has one more method which is

componentDidMount () { }

The way it's called is (in Child case)

constructor → render → componentDid Mount

This is the life cycle of React.

Now suppose componentDid Mount is in parent :

The cycle is

Parent constructor → Parent Render → Child constructor → Child Render → Child componentDid Mount → Parent Component Did Mount.

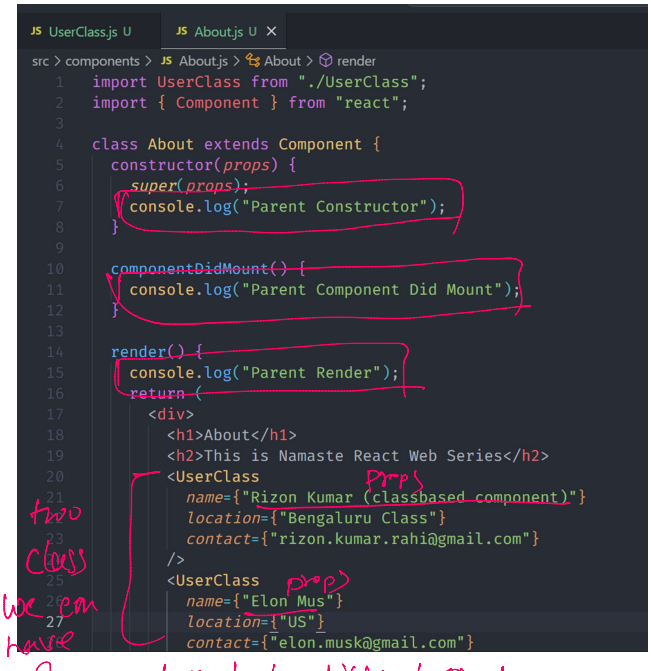
Use case of ComponentDidMount():

- There are something we do once component is mounted successfully
- they are used to make API call.
  using API called inside called Based component, let's go to f() component first
- Reason for ^ in the f() component we call API inside useEffect(), we do here because ↓

loads → Render → API → Render

now similar in class Based component we use componentDidMount.

Now suppose inside parent there are multiple children,



```
name={"Elon Mus"}
location={"US"}
contact={"elon.musk@gmail.com"}
```

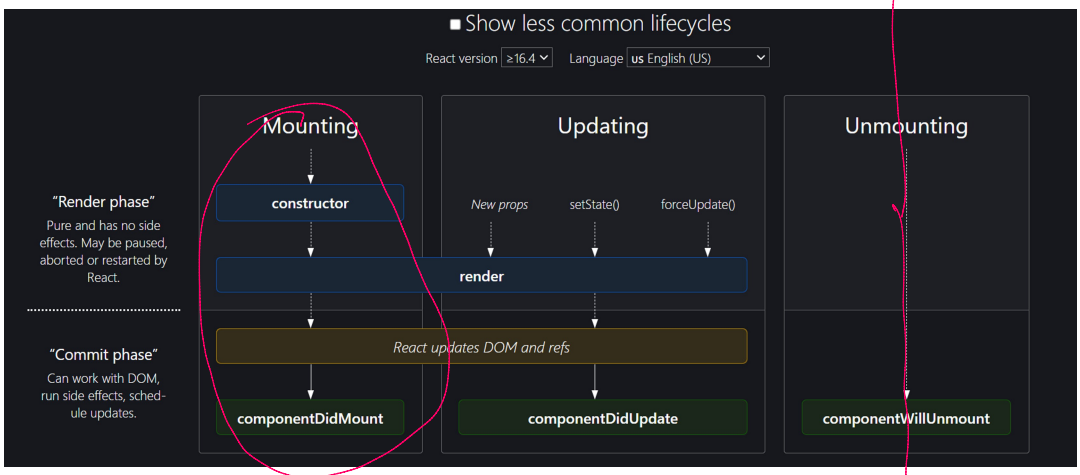
Same class but different props

```
this.setState({
  count: this.state.count + 1,
})
```

Order of the Render :

```
Parent Constructor                                                    About.js:7
Parent Render                                                         About.js:15
Rizon Kumar (classbased component)Child Constructor                   UserClass.js:9
Rizon Kumar (classbased component)Child Render                        UserClass.js:19
Elon MusChild Constructor                                             UserClass.js:9
Elon MusChild Render                                                  UserClass.js:19
Rizon Kumar (classbased component)Child Component Did Mount           UserClass.js:13
Elon MusChild Component Did Mount                                     UserClass.js:13
Parent Component Did Mount                                            About.js:11
```

→ Reason for this



The reason why react is fast :

1) Render phase → constructor called, rendered called, react will update the DOM

2) Commit phase → then component Did Mount is called

As there are two child, react will not call component Did Mount, instead it optimise it by batching the render phase of these two child & then commit phase is called.

- Parent constructor
- Parent Render
  - first constructor

- First constructor
- First render
- Second Constructor
- Second render

<DOM Updated - In Single Batch>

- First Component DidMount
- Second Component Did Mount

- Parent Component Did Mount.

Let's see how to make API call:

Calling github user API:

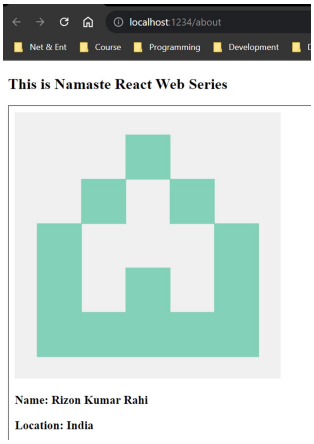
```
JS UserClass.js U ×
src > components > JS UserClass.js > UserClass > render
1  import React from "react";
2
3  class UserClass extends React.Component {
4    constructor(props) {
5      super(props);
6      this.state = {
7      };
8      // console.log(this.props.name + "Child Constructor")
9    }
10
11   async componentDidMount(){
12     // console.log(this.props.name + "Child Component Did Mount");
13     // API Call
14     const data = await fetch("https://api.github.com/users/rizonkumar");
15     const json = await data.json();
16
17     console.log(json)
18   }
19   render() {
20     const { name, location, contact } = this.props;
21     const { count } = this.state;
22     // console.log(this.props.name + "Child Render");
```

```
▼ {login: 'rizonkumar', id: 78218974, node_id: 'MDQ6VXNlcjc4MjE4OTc0', avatar_url: 'https://avatars.githubusercontent.com/u/78218974?v=4', gravatar_id: '', …}   UserClass.js:17
    avatar_url: "https://avatars.githubusercontent.com/u/78218974?v=4"
    bio: "SFCC Developer @ Merkle | Full Stack Developer | Tech Community | Problem Solver | E-Sport Athlete. "
    blog: "https://linktr.ee/rizonkumarrahi"
    company: "Merkle"
    created_at: "2021-01-29T14:40:11Z"
    email: null
    events_url: "https://api.github.com/users/rizonkumar/events{/privacy}"
    followers: 4
```

Let's show this data. (JSON data to webpage)

- create a state variable & update the state (which will get our data)

```
UserClass.js U
src > components > UserClass.js > UserClass > constructor > userInfo
  async componentDidMount(){
    // console.log(this.props.name + "Child Component Did Mount");
    // API Call
    const data = await fetch("https://api.github.com/users/rizonkumar");
    const json = await data.json();

    this.setState({
      userInfo: json,
    })
    console.log(json)
  }
  render() {
    // const { name, location, contact } = this.props;
    // const { count } = this.state;
    // console.log(this.props.name + "Child Render")

    const {name, location, avatar_url} = this.state.userInfo;
    return (
      <div className="user-card">
        <img src={avatar_url} />
        <h3> Name: {name}</h3>
        <h3> Location: {location}</h3>
        {/* <h3> Contact: {contact} </h3> */}
      </div>
    );
  };
};
```



- setState update the state variable & it's in the updating stage in the life cycle. so react again render once again and react will updat the DOM with new values and after that there is something known as componentDidUpdate() which updat later on.

Life Cycle :

Mounting life cycle

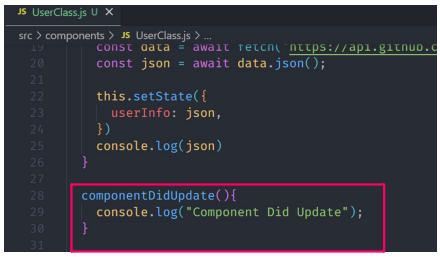
- Constructor
- Render (dummy)
  - <HTML Dummy> // for few milliseconds
- Component Did Mount
  - <API Call>
    - <this.setState> (when setState is called update cycle is called)

Update Cycle

- render (API data new data)

- <HTML (new API data → name, img, location)

- Component Did Update.



```
JS UserClass.js U
src > components > JS UserClass.js > ...
    const data = await fetch( https://api.github.c
    const json = await data.json();

    this.setState({
      userInfo: json,
    })
    console.log(json)
  }

  componentDidUpdate(){
    console.log("Component Did Update");
  }
```

```
FirstChild Constructor                                                    UserClass.js:13
FirstChild Render                                                         UserClass.js:35
FirstChild Component Did Mount                                            UserClass.js:17
▶ {login: 'rizonkumar', id: 78218974, node_id: 'MDQ6VXNlcjc4MjE4OTc0', avatar_url: 'https://avatars.githubusercontent.com/u/78218974?v=4', gravatar_id: '', …}   UserClass.js:25
FirstChild Render                                                         UserClass.js:35
Component Did Update                                                      UserClass.js:29
```

- There is something known as componentWillUnMount() method this is called just before unmounting it means when the component will be removed from the page for eg when you go from About.js → ContactUs Page then componentWillUnMount will be called.

- Never Ever compare React life cycle with Functional Component.

In functional componet if you want to unmount then you will return it in useEffect.

E.g.

```
useEffect(() => {
// API call
```

```
const timer = Set Interval (() => {
    console.log ("Namaste React")
}, 1000);
return () => {
  clear Interval (timer);   // Clearing
  };
}, []);
```