K-Digital Training KDT 풀스택 웹 개발자 양성 부트캠프 3기

React

WITH 팀 뤼쳐드

ref

HTML에서 요소 선택

- id 지정 후 요소 호출

```
<div id="id1"> 내용 </div>
```

React에서 요소 선택

-> React 컴포넌트가 아닌 **HTML Element에 직접 접근**해야 한다면..?

- **1) id 지정**

```
<div id="id1"> 내용 </div>
```

- **2) ref 지정**

React에서 요소 선택

-> React 컴포넌트가 아닌 **HTML Element에 직접 접근**해야 한다면..?

- 1번 방법) id 지정
  - 같은 컴포넌트를 반복해서 사용하면 **id가 중복**되는 문제
  - => 고유한 id가 되지 않음.

반면에 **ref** 는 **컴포넌트 내부에서만** 작동!!

ref 란?

- Reference
- 전역적으로 작동하지 않고 컴포넌트 내부에서 선언 및 사용
- 동일한 컴포넌트를 반복해 사용해도 **각 컴포넌트 내부에서만 동작**
  - 따라서 중복되지 않음!
- **DOM을 직접적으로 건드려야 할 때** 사용
  - 특정 input에 focus 주기, 스크롤 박스 조작 등

주의! **Class 형 컴포넌트에서만 기본 기능**으로 제공함

ref prop

- HTML 요소의 레퍼런스를 ref prop에 저장

```
<element ref={myRef}></element>
```

Ref 사용하기

Part1. 함수형 컴포넌트

useRef()

함수형 컴포넌트에서는 useRef()를 이용하여 DOM 요소에 직접 접근한다!

```
// 1. ref 객체 만들기
const myRef = useRef();

// 2. 선택하고 싶은 DOM 에 ref 값으로 설정
<element ref={myRef}></element>

// 3. useRef()로 만든 객체 안의 current가 실제 요소를 가르킴
myRef.current;
```

useRef() 의 용도

1. DOM 요소에 접근

ex. 버튼 클릭시 input 요소에 포커스 주기
=> **input 요소**에 **직접 접근** 필요!
document.querySelector()와 비슷!!

```jsx
import { useRef } from 'react';

const RefSample1 = () => {
  // 1. ref 객체 만들기
  const inputRef = useRef();

  const handleFocus = () => {
    // 3. useRef()로 만든 객체의 current 값에 focus() DOM API 사용
    // *focus(): 지정된 html 요소에 포커스 설정
    inputRef.current.focus();
  };

  return (
    <>
      <p>(함수형 컴포넌트) 버튼 클릭시 input에 focus 처리</p>
      {/* 2. 선택하고 싶은 DOM에 ref prop 설정 */}
      <input ref={inputRef} />
      <button onClick={handleFocus}>버튼</button>
    </>
  );
};

export default RefSample1;
```

useRef() 의 용도

2. 로컬변수로 사용

로컬변수 = 랜더링되어도 값이 그대로 유지
=> ref 안의 변경되어도 컴포넌트는 랜더링되지 않음!!!

```
import { useRef } from 'react';

const RefSample2 = () => {
  const idRef = useRef(1);

  const plusIdRef = () => {
    idRef.current += 1;
    console.log(idRef.current);
  };

  return (
    <div>
      <h1>Ref Sample</h1>
      <h2>{idRef.current}</h2>
      <button onClick={plusIdRef}>PLUS Ref</button>
    </div>
  );
};

export default RefSample2;
```

Ref 사용하기

Part2. 클래스형 컴포넌트

~~참고용으로만 알아두자..!~~

클래스형 컴포넌트에서 ref 사용 방법

1. 콜백 함수

```
<input ref={ (ref) => { this.input = ref } />
```

- 사용하고자 하는 DOM 요소에 ref 라는 콜백 함수 작성 및 props로 전달
- ref 를 컴포넌트의 멤버 변수로 설정하는데, 이때 ref는 원하는 대로 사용 가능
  - this.input을 this.orange 라고 사용해도 무관

클래스형 컴포넌트에서 ref 사용 방법

1. 콜백 함수

```jsx
import React from 'react';

class RefSample3 extends React.Component {
  handleFocus = () => {
    this.myInput.focus();
  };

  render() {
    return (
      <>
        <p>(클래스형 컴포넌트) 버튼 클릭시 input에 focus 처리</p>
        {/* 만들어진 변수 myInput 해당 요소의 ref prop로 넣어주면 ref 설정 완료 */}
        <input
          ref={(ref) => {
            this.myInput = ref;
          }}
        />
        <button onClick={this.handleFocus}>focus</button>
      </>
    );
  }
}

export default RefSample3;
```

클래스형 컴포넌트에서 ref 사용 방법

2. 내장 함수 createRef

```
input = React.createRef();
```

- React.createRef 함수를 이용해 컴포넌트 내부에서 변수에 ref 요소를 담아준다.
- 이때 만든 input요소는 실제 DOM 요소의 prop 에 연결해야 한다.
- 사용할 때는 this.input.current 를 이용한다.

클래스형 컴포넌트에서 ref 사용 방법

2. 내장 함수 createRef()

```jsx
import React from 'react';

class RefSample4 extends React.Component {
  // createRef를 사용해 만들 때는 컴포넌트 내부에서 변수에 React.createRef()를 담아주어야 함
  myInput = React.createRef();

  handleFocus = () => {
    // ref를 설정한 DOM에 접근하기 위해서는 this.myInput.current 이용
    this.myInput.current.focus();
  };

  render() {
    return (
      <>
        <p>(클래스형 컴포넌트) 버튼 클릭시 input에 focus 처리</p>
        {/* 만들어진 변수 myInput 해당 요소의 ref prop로 넣어주면 ref 설정 완료 */}
        <input ref={this.myInput} />
        <button onClick={this.handleFocus}>focus</button>
      </>
    );
  }
}

export default RefSample4;
```

Life Cycle

라이프 사이클 이란?

- 라이프사이클 (LifeCycle)
- 수명 주기
- 모든 React Component 에 존재하는 것

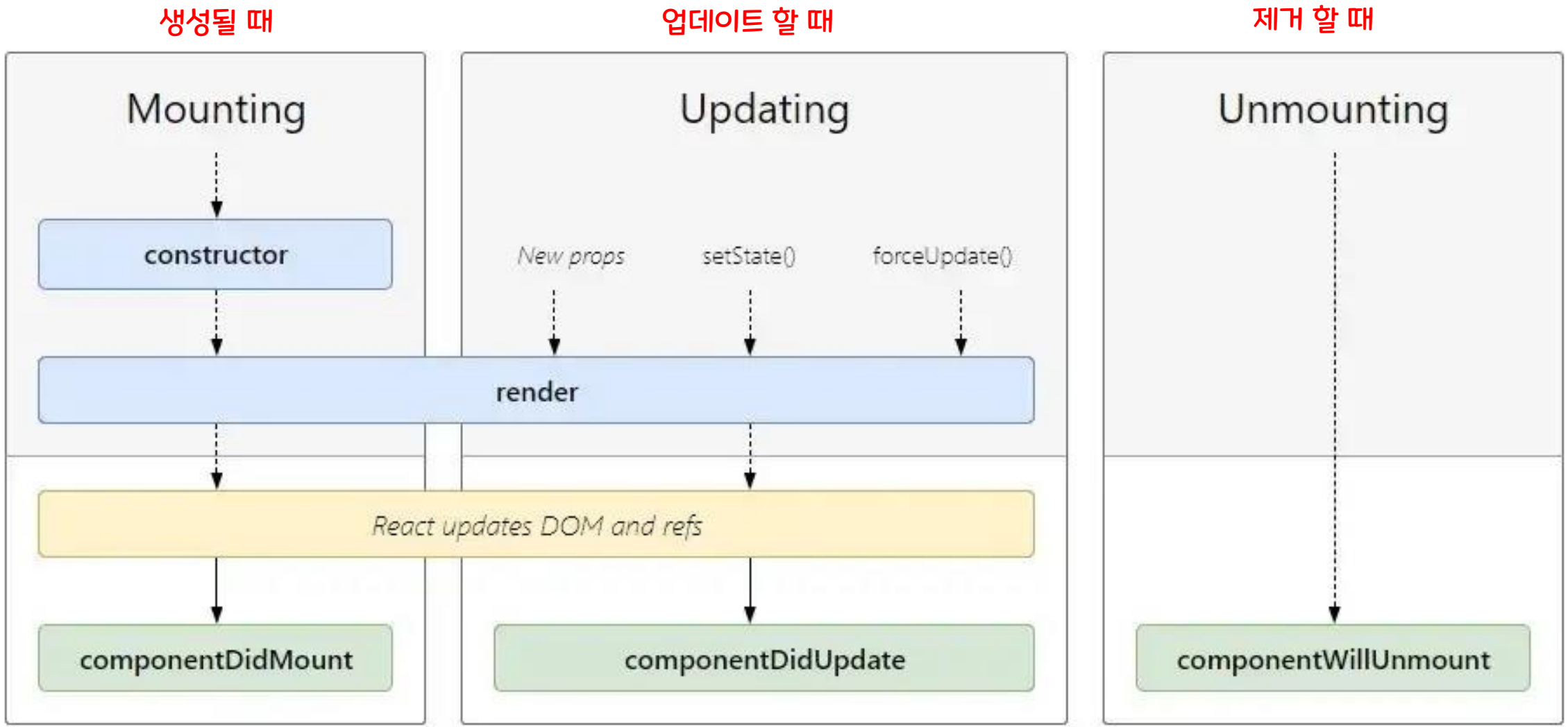
두산백과

생애 주기

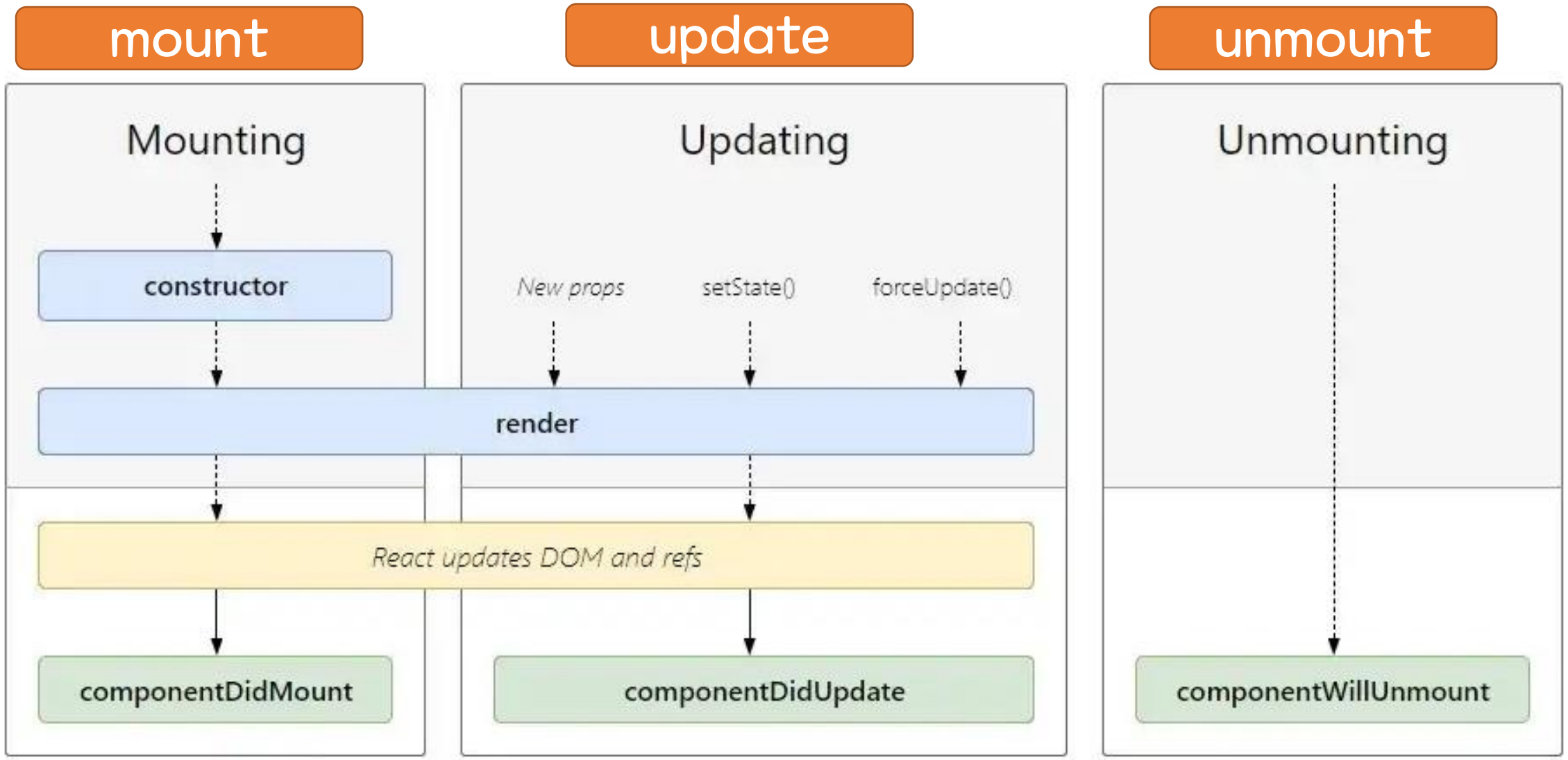
[life cycle , 生涯週期]

요약 사람의 생애를 개인이나 가족의 생활에서 발생하는 커다란 변화를 기준으로 하여 일정한 단계로 구분한 과정.

라이프사이클


생성될 때
업데이트 할 때
제거 할 때
Mounting
Updating
Unmounting
constructor
New props
setState()
forceUpdate()
render
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount

라이프 사이클


mount
update
unmount
Mounting
Updating
Unmounting
constructor
New props
setState()
forceUpdate()
render
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount

라이프사이클 용어 정리

- **Mount**
  - DOM이 생성되고 웹 브라우저 상에 나타남
- **Update**
  - props or state 바뀌었을 때 업데이트함
- **Unmount**
  - 컴포넌트가 화면에서 사라질(제거될) 때

LifeCycle 이해하기

Part1. 함수형 컴포넌트

useEffect()

- 함수형 컴포넌트에서는 useEffect()를 사용해
  Mount, Update, Unmount 시 특정 작업을 처리

```
useEffect(effect, deps);
// effect: 랜더링 이후 실행할 함수 (Mount, Update)
//     - effect 함수에서 함수를 return시 반환하는 함수가 컴포넌트 Unmount 될 때 실행됨
// deps: 배열 값이 변하면 effect 함수 실행
//     - 생략: Mount & Update 시 동작
//     - []: 빈배열 -> Mount 시에만 동작
//     - [data]: 배열 안의 data 값이 Update 시 동작
```

useEffect()

```
// 컴포넌트 Mount or Update 될 때 동작
useEffect(() => {});


// 컴포넌트 Mount 될 때 동작
useEffect(() => {}, []);


// data 값이 Update 될 때 동작
const [data, setData] = useState();
useEffect(() => {}, [data]);
```

useEffect()

```jsx
import { useState, useEffect } from 'react';

const MyComponent = (props) => {
  const { number } = props;
  const [text, setText] = useState('');

  useEffect(() => {
    // Mount 시점에 실행
    console.log('Functional Component | ⭕ mount!');

    // Unmount 시점에 실행
    return () => {
      console.log('Functional Component | ❌ unmount');
    };
  }, []);

  // Mount & Update 시점에 실행
  useEffect(() => {
    console.log('Functional Component | ✅ update!');
  });

  // text가 바뀔 때만 실행
  useEffect(() => {
    console.log('Functional Component | ✅ text update!');
  }, [text]);

  return (
    <>
      <p>MyComponent {number}</p>
      <input
        type="text"
        value={text}
        onChange={(e) => setText(e.target.value)}
      />
    </>
  );
};
```

```jsx
const LifeCycleFunc = () => {
  const [number, setNumber] = useState(0);
  const [visible, setVisible] = useState(true);

  const changeNumberState = () => {
    setNumber(number + 1);
  };

  const changeVisibleState = () => {
    setVisible(!visible);
  };

  return (
    <>
      <button onClick={changeNumberState}>PLUS</button>
      <button onClick={changeVisibleState}>ON/OFF</button>
      {visible && <MyComponent number={number} />}
    </>
  );
};

export default LifeCycleFunc;
```

LifeCycle 이해하기

Part2. 클래스형 컴포넌트

~~참고용으로만 알아두자..!~~

마운트(mount)

마운트 될 때 호출되는 메서드

- constructor
- render
- getDerivedStateFromProps
- componentDidMount

업데이트(update)

업데이트 될 때 호출되는 메서드

- getDerrivedStateFromProps
- shouldComponentUpdate
- componentDidUpdate

언마운트(unmount)

- comopnentWillUnmount

라이프 사이클

- 앞에 소개한 모든 메서드를 완벽히 알아야 하는 것은 아님. 심지어 이제는 클래스형 컴포넌트를 사용하지도 않음..
- 실제로 공식 문서에서도 자주 사용되는 메서드와 자주 사용하지 않는 메서드를 구분하고 있음
- 자주 사용하는 메서드
  - componentDidMount
  - componentDidUpdate
  - componentWillUnmount

https://ko.reactjs.org/docs/react-component.html#commonly-used-lifecycle-methods

라이프 사이클


```jsx
import React from 'react';

class MyComponent extends React.Component {
  componentDidMount() {
    console.log('Class Component | ⭕ mount!');
  }

  componentDidUpdate() {
    console.log('Class Component | ✅ update!');
  }

  componentWillUnmount() {
    console.log('Class Component | ❌ unmount!');
  }

  render() {
    return <h1>MyComponent {this.props.number}</h1>;
  }
}
```

```jsx
class LifeCycleClass extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      number: 0,
      visible: true,
    };
  }

  changeNumberState = () => this.setState({ number: this.state.number + 1 });
  changeVisibleState = () => this.setState({ visible: !this.state.visible });

  render() {
    return (
      <>
        <button onClick={this.changeNumberState}>PLUS</button>
        <button onClick={this.changeVisibleState}>ON/OFF</button>
        {this.state.visible && <MyComponent number={this.state.number} />}
      </>
    );
  }
}

export default LifeCycleClass;
```