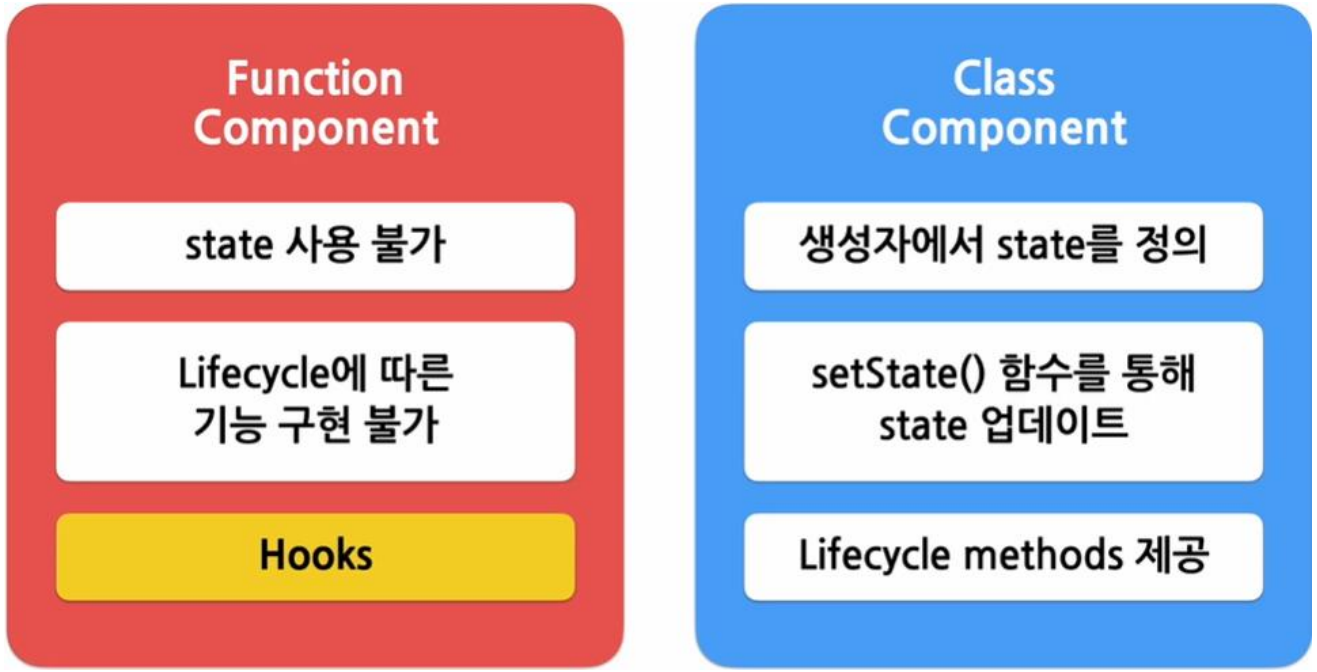
7. Hook

클래스 bye, 지금은 함수의 시대

7.1 Hook이란?

- Hook은 React 버전 16.8부터 새롭게 추가



- **Hook은 Class없이 React 기능들을 사용하는 방법을 제시**
  - 대표적으로 useState(), useEffect() 등

7.1 Hook이란?

- 원래 존재하는 어떤 기능에 마치 갈고리를 거는 것처럼 끼어 들어가 수행되는 것
- React의 state와 생명주기 기능에 갈고리를 걸어 원하는 시점에 정해진 함수를 실행되도록 만든 것
- 이 함수들을 Hook이라 부르고 이름은 모두 use로 시작함

7.2 useState(예제)

- 가장 대표적이고 많이 사용되는 Hook
- state를 생성하고 변경하기 위한 Hook
- 함수 컴포넌트에서는 기본적으로 state라는 것을 제공하지 않음
- 클래스 컴포넌트처럼 state를 사용하고 싶으면 useState()를 사용해야 함
- 사용법
  - const [변수명, set함수명] = useState(초기값);
  - 변수 각각에 대해 set 함수가 따로 존재함

7.3 useEffect

- useState와 같이 가장 많이 사용되는 Hook
- ~~사이드 이펙트를 수행하기 위한 Hook~~
  - 서버에서 데이터를 받아오거나 수동으로 DOM을 변경하는 등의 작업
  - 다른 컴포넌트에 영향을 미칠 수 있으며 렌더링이 끝난 이후에 실행되어야 하는 작업들
- 마운트/업데이트/언마운트 시 할 작업 설정
  - useEffect() Hook 을 사용하여 컴포넌트가 마운트 됐을 때 (처음 나타났을 때), 업데이트 될 때 (props 또는 state가 바뀔 때), 언마운트 됐을 때 (사라질 때) 특정 작업을 처리

7.3 useEffect

- useEffect() 만으로 클래스 컴포넌트의 생명주기 메서드들과 동일한 기능을 수행할 수 있음
  - componentDidMount()
  - componentDidUpdate()
  - componentWillUnmount()
  - 위 3개를 하나로 통합해서 제공

7.3 useEffect(예제)

- 사용법
  - useEffect(이펙트 함수, 의존성 배열);
  - **의존성 배열 안에 있는 변수 중에 하나라도 값이 변경되었을 때 이펙트 함수가 실행됨 ← 주로 사용**
  - 의존성 배열에 빈 배열([])을 넣으면 마운트와 언마운트 시에 단 한 번씩만 실행됨(어떠한 값에도 의존하지 않겠다는 의미!)
  - 의존성 배열 생략 시 마운트 될 때와 업데이트 될 때마다 호출됨 (처음 렌더링 될 때를 포함해서 매번 렌더링 될 때마다 실행)
  - 선언된 컴포넌트의 props와 state에 접근할 수 있음
  - useEffect()에서 리턴하는 함수는 컴포넌트 마운트가 해제될 때 호출됨
    - clean-up(정리) 함수

7.3 useEffect

- 사용법

```
useEffect(() => {
    // 컴포넌트가 마운트 된 이후,
    // 의존성 배열에 있는 변수들 중 하나라도 값이 변경되었을 때 실행됨
    // 의존성 배열에 빈 배열([])을 넣으면 마운트와 언마운트시에 단 한 번씩만 실행됨
    // 의존성 배열 생략 시 컴포넌트 업데이트 시마다 실행됨
    ...

    return () => {
        // 컴포넌트가 마운트 해제되기 전에 실행됨
        ...
    }
}, [의존성 변수1, 의존성 변수2, ...]);
```

7.4 useMemo

- Memoized value를 리턴하는 Hook
  - Memoization 개념: 연산량이 많은 동일한 계산을 반복해야 할 때, 이전 결과를 메모리에 저장해 두었다가 동일한 계산에서 쓰는 것
  - 즉, 이전에 계산 한 값을 재사용
- 연산량이 높은 작업이 매번 렌더링 될 때마다 반복되는 것을 피하기 위해 사용
- 최적화와 관련된 Hook

7.4 useMemo

- 사용법
  - const 변수명 = useMemo(값 생성 함수, 의존성 배열);
  - 의존성 배열에 들어있는 변수가 변했을 경우에만 새로 값 생성 함수를 호출하여 결과값을 반환함
  - 그렇지 않은 경우에는 기존 함수의 결과값을 그대로 반환함
  - 의존성 배열을 넣지 않을 경우 렌더링이 일어날 때마다 매번 값 생성 함수가 실행되므로 의미가 없음

7.4 useMemo

- 사용법

```
const memoizedValue = useMemo(
    () => {
        // 연산량이 높은 작업을 수행하여 결과를 반환
        return computeExpensiveValue(의존성 변수1, 의존성 변수2);
    },
    [의존성 변수1, 의존성 변수2]
);
```

7.5 useCallback

- useMemo() Hook과 유사하지만 값이 아닌 함수를 반환한다는 점이 다름
- 컴포넌트 내에 함수를 정의하면 매번 렌더링이 일어날 때마다 함수가 새로 정의되므로 useCallback() Hook을 사용하여 불필요한 함수 재정의 작업을 없애는 것
- 최적화와 관련된 Hook
- 사용법
  - const 함수명 = useCallback(콜백 함수, 의존성 배열);
  - 의존성 배열에 들어있는 변수가 변했을 경우에만 콜백 함수를 다시 정의해서 리턴함

7.5 useCallback(예제)

- 사용법

```
const memoizedCallback = useCallback(
    () => {
        doSomething(의존성 변수1, 의존성 변수2);
    },
    [의존성 변수1, 의존성 변수2]
);
```

7.6 useRef(예제1)

- 레퍼런스를 사용하기 위한 Hook
  - 레퍼런스(참조)란 특정 컴포넌트에 접근할 수 있는 객체
- 매번 렌더링 될 때마다 **항상 같은 레퍼런스 객체를 반환**
- 반환된 객체는 컴포넌트의 전 생애주기에 걸쳐서 유지됨
  - 컴포넌트가 마운트 해제 되기 전까지 계속 유지
- 부득이하게 특정 DOM을 선택해야 하는 경우에만 사용
  - In JavaScript, getElementById(), querySelector() 같은 DOM Selector 함수를 사용
  - In React, useRef() Hook 사용
  - 예를 들어 특정 엘리먼트의 크기가 필요할 때, 스크롤바 위치를 가져오거나 설정해야 될 때, 또는 input 같은 상호작용 가능한 엘리먼트에 포커스를 줘야 할 때 등

7.6 useRef(예제2)

- 또 다른 용도로 컴포넌트 안에 변수 만들 때 사용
  - 컴포넌트 안에서 조회 및 수정 할 수 있는 지역 변수
  - useRef 로 관리하는 변수는 값이 바뀐다고 해서 컴포넌트가 리렌더링 되지 않음
  - 렌더링이 될 때 값이 초기화 되지 않음
    (일반 변수로 선언 시 렌더링이 될 때 마다 값이 초기화 됨)
- 사용법
  - const 변수명 = useRef(초기값);
  - 변수명.current라는 속성을 통해서 접근
    - current: 현재 참조하고 있는 엘리먼트(또는 값)
    - current 속성에 변경 가능한 값을 담고 있는 "상자"와 같음

7.7 Hook의 규칙

- 무조건 최상위 레벨에서만 호출해야 함
  - 함수 컴포넌트 안에서 최상위 레벨을 의미
  - 반복문이나 조건문 또는 중첩된 함수들 안에서 Hook을 호출하면 안됨
    - Hook의 실행 순서가 바뀔 수 있음
      → state가 꼬이거나 렌더링 문제 유발
- 함수 컴포넌트에서만 Hook을 호출해야 함
  - 또는 직접 만든 커스텀 Hook에서만 호출 가능

```
function HookRules(props) {
  const [name, setName] = useState('Goni');

  useEffect(() => {
    // "Good"
  });

  if (name !== '') {
    useEffect(() => {
      // ... Bad ...
    });
  }

  for (let index = 0; index < 3; index++) {
    useEffect(() => {
      // ... Bad ...
    });
  }

  function func() {
    useEffect(() => {
      // ... Bad ...
    });
  }

  return (
    <div>
    </div>
  )
};
```

7.8 Custom Hook 만들기(예제)

- 기본적으로 제공되는 Hook 이외에 추가적으로 필요한 기능이 있다면 직접 만들어 사용
- 이것을 커스텀 훅이라 부름
- 여러 컴포넌트에서 반복적으로 사용되는 로직을 훅으로 만들어 재사용

7.8 Custom Hook 만들기

- 정리
  - 이름이 use로 시작하여 React Hook 임을 알리고,
    내부에서 다른 Hook을 호출하는 단순한 자바스크립트 함수
  - 함수를 만들 때 파라미터로 무엇을 받을지, 어떤 것을 리턴 할지는 필요에 따라 직접 정함
  - 중복되는 로직을 커스텀 훅으로 추출하여 재사용성 높이기
  - 만약 여러 개의 컴포넌트에서 하나의 커스텀 훅을 사용할 때 훅 내부에 있는 state를 공유하는 것 일까..?
    - 그렇지 않다.
    - 커스텀 훅을 호출할 때마다 별개의 state를 얻게 됨!

7.9 훅을 사용한 컴포넌트 개발(실습)

- my-app/src/chapter6 실습
- 1) useCounter() 커스텀 훅 만들기
- 2) Accommodate 컴포넌트 만들기
- 3) 실행하기