10. React Hooks

다중 선택	react 입문

1. React Hooks란?

- 함수형 컴포넌트에서도 상태(state)와 생명주기(lifecycle) 기능을 사용할 수 있도록 해주는 React의 기능
- React 16.8에서 도입됨
- Hooks를 사용하면 클래스형 컴포넌트 없이도 상태 관리와 로직 재사용이 가능해 코드가 간결해지고 유지보수성이 향상됨

2. React 내장 Hooks

(1) 상태 관리와 렌더링

- `useState`: 상태 관리를 위한 Hook

```
const [count, setCount] = useState(0);
```

  - `count`: 현재 상태 값
  - `setCount`: 상태를 업데이트하는 함수

- `useReducer`: 복잡한 상태 로직 관리

```
const [state, dispatch] = useReducer(reducer, initialSta
te);
```

(2) 컴포넌트 생명주기 관련

- `useEffect`: 사이드 이펙트를 처리
  - 렌더링 후 수행되며, 클린업 함수로 정리 가능

```
useEffect(() => {
  console.log("Component mounted or updated");
  return () => console.log("Cleanup on unmount or updat
e");
}, [dependency]);
```

- `useLayoutEffect`: DOM 업데이트 직후 실행 (렌더링 차이점 주의)

(3) 성능 최적화

- `useMemo`: 계산 결과를 캐싱

```
const memoizedValue = useMemo(() => computeExpensiveValu
e(a, b), [a, b]);
```

- `useCallback`: 함수 캐싱

```
const memoizedCallback = useCallback(() => doSomething
(a, b), [a, b]);
```

(4) 참조와 DOM 접근

- `useRef`: DOM 요소 또는 변수 참조

```
const inputRef = useRef(null);
inputRef.current.focus();
```

(5) Context와 연동

- `useContext`: Context API와 함께 사용

```
const value = useContext(MyContext);
```

3. 커스텀 Hooks

- **재사용 가능한 로직**을 만들기 위해 사용
- 이름이 "use"로 시작해야 함
- 예시: API 요청 로직을 위한 커스텀 Hook

```
import { useState } from "react";

// 3가지 Hook 관련된 팁
// 1. 함수 컴포넌트, 커스텀 훅 내부에서만 호출 가능
// 2. 조건문 또는 반복문 안에서 호출될 수 없음
// 3. 나만의 훅(커스텀 훅) 만들 수 있음

// 따로 떼서 저장하는 게 일반적
function useInput() {
  const [input, setInput] = useState("");

  const onChange = (e) => {
    setInput(e.target.value);
  };

  return [input, onChange];
}

const HookExam = () => {
  const [input, onChange] = useInput();
  const [input2, onChange2] = useInput();

  return (
    <>
      <input type="text" value={input} onChange={onChang
e} />
      <input type="text" value={input2} onChange={onChan
ge2} />
    </>
  );
};
```

```
export default HookExam;
```

→ `src/hooks` 폴더 안에 커스텀 훅 이름으로 저장해서 사용하는 게 일반적임. 위처럼 다 사용하지 않고.

4. Hooks 사용 시 주의사항

- **최상위 레벨에서만 호출**: 조건문이나 반복문 내부에서 호출하지 말 것
- **React 함수형 컴포넌트에서만 호출**: 클래스형 컴포넌트나 일반 함수에서 사용 불가
- **의존성 배열 관리**: `useEffect`나 `useMemo` 등에서 의존성 배열을 정확히 설정

5. 장점

- 클래스형 컴포넌트에 비해 코드가 간결하고 읽기 쉬움
- 로직을 훅으로 추상화해 재사용성 향상
- 생명주기 메서드 간의 의존성 문제 감소

6. 단점

- 복잡한 상태 로직은 다루기 어려울 수 있음
- 의존성 배열 관리 실수가 발생할 가능성 있음