7강

컴포넌트 분리 및 Props 복습

➢ 핵심 포인트

- 입력값은 state로 관리한다.
- onChange 이벤트로 state를 갱신(수정)한다
- setState()로 리스트 추가/삭제를 수행한다.
- State 변화 → UI 자동 업데이트

입력 → 상태 → 화면 업데이트

➢ 컴포넌트 분리하고 props로 연결

- 컴포넌트로 분리하고 props로 연결한다.
- state는 부모에, UI는 자식에게
- 한 덩어리 코드를 레고 블록으로 나누는 그림
- Modal, PostList, Blog 컴포넌트로 나누기
- props로 데이터 전달
- 자식이 부모의 함수를 실행하도록 구조 잡기

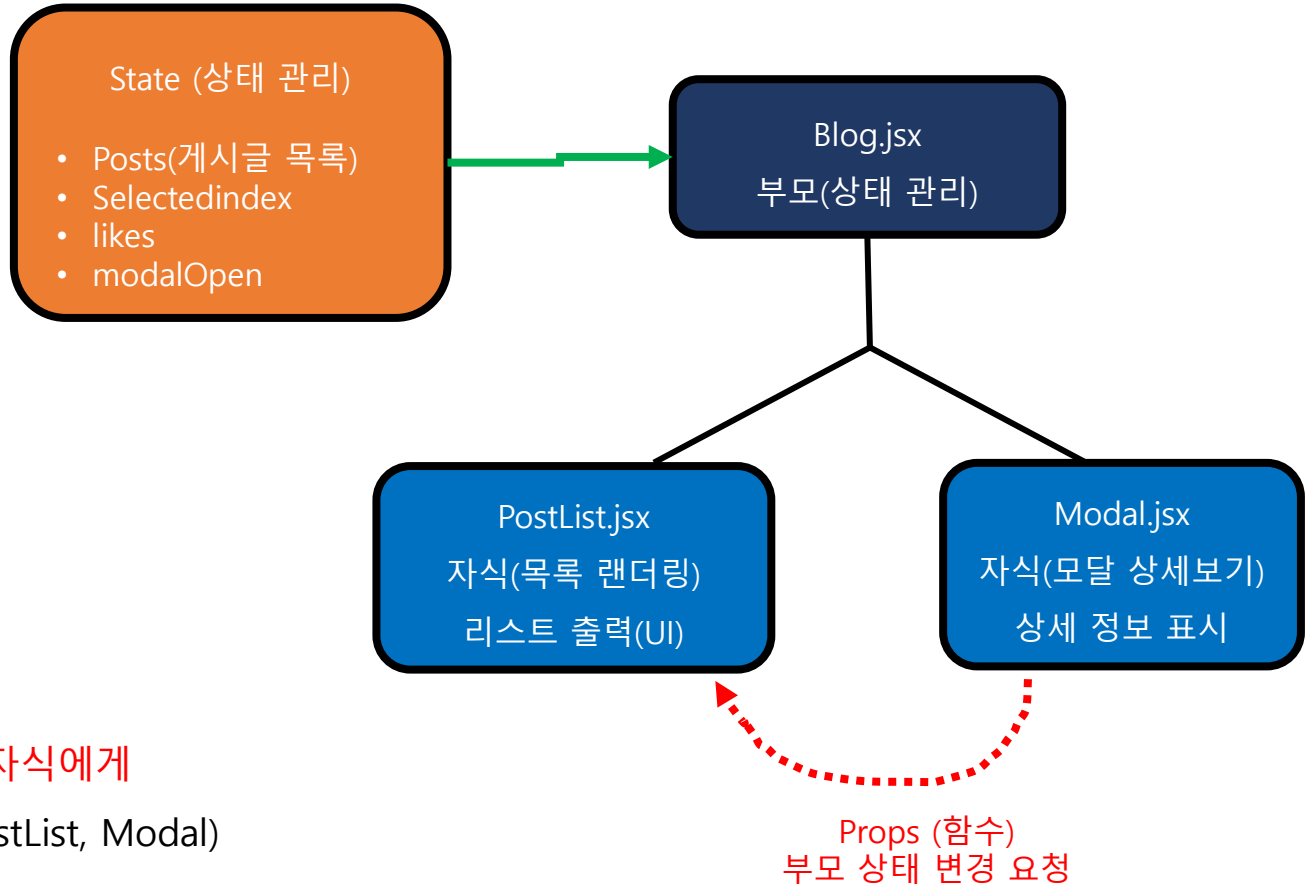
핵심

- 입력은 자식이 데이터는 부모가
- state는 부모가 갖는다
- 자식은 props로 전달받아 화면을 그린다.
- 자식이 부모의 함수를 props로 호출한다.

➢ Blog.jsx – 컴포넌트 분리하기

📣 폴더 구조

Blog.jsx
- PostList.jsx
- Modal.jsx



📣 핵심

- state는 부모에, UI는 자식에게
- 부모(Blog) → 자식(PostList, Modal)
- props → 데이터 전달
- props(함수) ← 자식에서 부모 상태 변경

➢ **Blog.jsx –** state를 보관하고, 자식 컴포넌트에 props로 전달한다.

```jsx
export default function Blog() {
  const [posts, setPosts] = useState([
    '남자코트 추천',
    '강남 우동맛집',
    '파이썬독학',
  ]);
  const [likes, setLikes] = useState([0, 0, 0]);
  //모달 창이 안보이는 상태를 false로 지정
  const [modalOpen, setModalOpen] = useState(false);
  // 선택된 글의 인덱스
  const [selectedIndex, setSelectedIndex] = useState(null);
  const [inputValue, setInputValue] = useState('');

  // 글 추가
  const addPost = () => {
    if (inputValue.trim() === '') {
      return alert('자료를 입력하세요');
    }
    let postCopy = [...posts];
    postCopy.unshift(inputValue);
    setPosts(postCopy);

    let likesCopy = [...likes];
    likesCopy.unshift(0);
    setLikes(likesCopy);

    setInputValue('');
  };
```

```jsx
// 글 삭제
const delPost = (index) => {
  let postCopy = [...posts];
  postCopy.splice(index, 1);
  setPosts(postCopy);

  let likesCopy = [...likes];
  likesCopy.splice(index, 1);
  setLikes(likesCopy);
};

//따봉 증가
const addLikes = (index) => {
  let likesCopy = [...likes];
  likesCopy[index] += 1;
  setLikes(likesCopy);
};
```

➢ **Blog.jsx –** state를 보관하고, 자식 컴포넌트에 props로 전달한다.

```
return (
  <div className="blog-container">
    <div className="nav-bar">
      <h4 style={{ color: 'white', fontSize: '16px' }}>ReactBlog</h4>
    </div>
    {/* PostList컴포넌트 분리 */}
    <PostList
      post={posts}
      likes={likes}
      setModalOpen={setModalOpen}
      setSelectedIndex={setSelectedIndex}
      delPost={delPost}
      addLikes={addLikes}
    />
    {/* 글 입력 창 */}
    <div className="input-area">
      <input type="text" onChange={(e) => setInputValue(e.target.value)} value={inputValue}/>
      <button onClick={addPost}>글 발행</button>
      {/* 모달 조건부 렌더링 부분 */}
      {modalOpen && (
        <Modal
          color={'lightgray'}
          title={posts}
          setPosts={setPosts}
          index={selectedIndex}
          onClose={() => setModalOpen(false)}
        />
      )}
    </div>
  </div>
);
```

➢ **PostList.jsx -** posts 배열을 map으로 렌더링 한다.

```jsx
export default function PostList(props) {
  return (
    <>
      <div className="post-list">
        {props.post.map((post, index) => (
          <div className="post-card" key={index}>
            <h4
              onClick={() => {
                props.setModalOpen(true);
                props.setSelectedIndex(index);
              }}
              style={{ cursor: 'pointer' }}
            >
              {post}
              <span
                className="like"
                onClick={(e) => {
                  e.stopPropagation();
                  props.addLikes(index);
                }}
              >
                👍
              </span>{props.likes[index]}
            </h4>
            <p className="post-date">🗓️11월 1일 발생</p>
            <button className="delete-btn"
              onClick={() => {
                props.delPost(index);
              }}
            >
              삭제
            </button>
          </div>
        ))}
      </div>
    </>
```

React 이벤트 버블링(Event Bubbling) 문제

1. 사용자가 👍 아이콘(span) 을 클릭함
2. span의 onClick 이벤트(props.addLikes(index))가 실행됨
3. 하지만 그 클릭 이벤트가 부모 요소인 <h4>로 "**전달(버블링)**" 됨
4. 그래서 <h4>의 onClick 이벤트(모달 열기)도 함께 실행됨
5. 해결 방법 : **e.stopPropagation()**

➢ **Modal.jsx -** props로 선택된 글과 닫기 함수를 받아 모달을 표시 한다.

```jsx
// 자식 컴포넌트 Modal 생성하기
export default function Modal(props) {
  const update = () => {
    let titleCopy = [...props.title];
    titleCopy[props.index] =
      prompt('새 제목을 입력하세요', titleCopy[props.index]) ||
      titleCopy[props.index];
    props.setPosts(titleCopy);
  };
  return (
    <>
      <div className="modal-overlay">
        <div className="modal" style={{ backgroundColor: props.color }}>
          <h4>{props.title[props.index]}</h4>
          <p>날짜: 11월 1일</p>
          <p>상세내용: 여기에 내용을 넣어보세요</p>
          <div className="modal-buttons">
            <button onClick={update}>글수정</button>
            <button onClick={props.onClose}>닫기</button>
          </div>
        </div>
      </div>
    </>
  );
}
```

문제1] 다음의 조건에 만족하도록 React를 작성 하시오.

조건

① 영화 제목 + 리뷰 입력 → 목록에 추가

② 각 영화는 "상세보기" 버튼 클릭 시 모달로 세부 내용 표시

③ 리뷰 없으면 "아직 작성된 리뷰가 없습니다." 출력

④ 모달은 props를 통해 부모 → 자식으로 데이터 전달

⑤ 컴포넌트 구조

<MovieApp /> // 부모

├ <MovieList /> // 영화 목록

├ <MovieItem /> // 각 영화 항목

└ <MovieModal /> // 선택된 영화 상세정보 모달

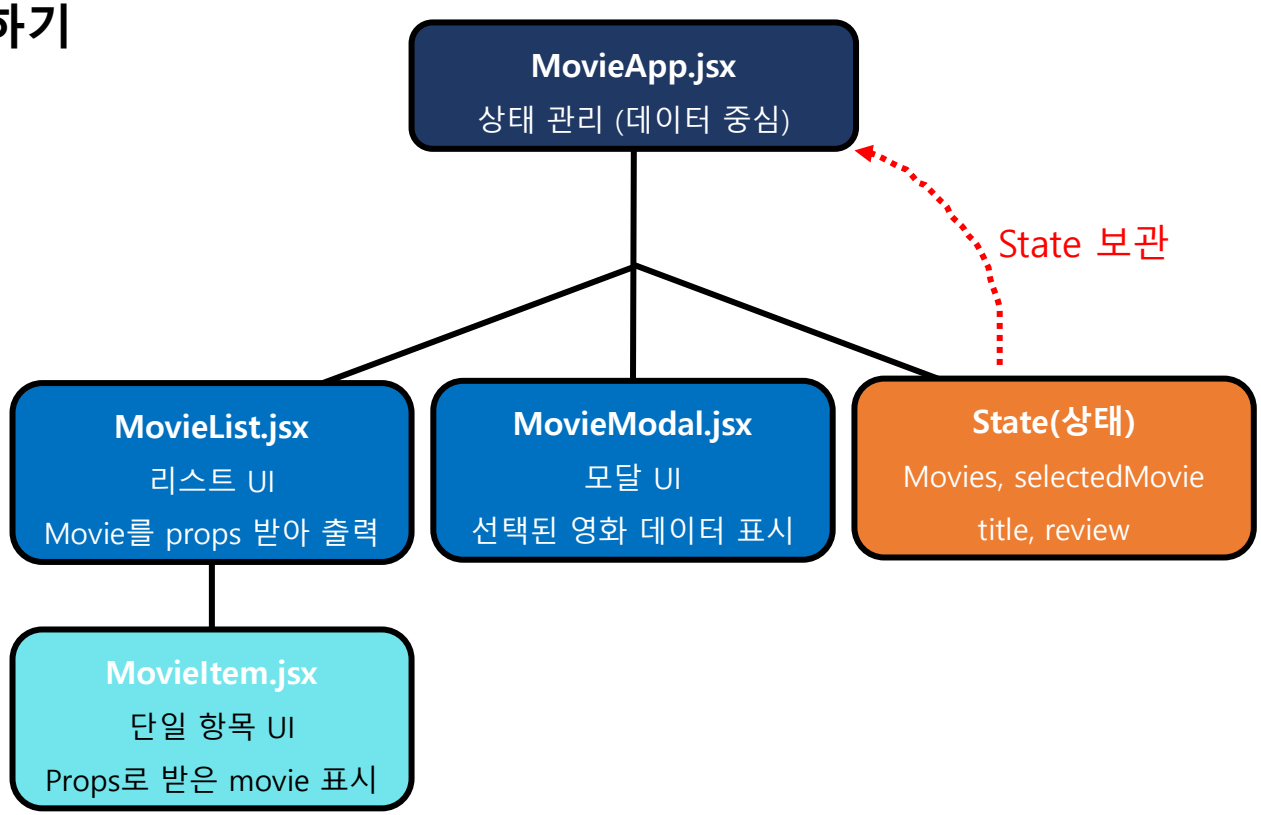
⑥ src 폴더안에 -> stateUI 폴더 생성 -> movie 폴더 생성 -> MovieApp.jsx, MovieList.jsx, MovieItem.jsx, MovieModal.jsx 파일을 생성하여 작성하시오.

⑦ src 폴더안에 -> App.jsx에 작성한 MovieApp.jsx를 import하여 실행 하시오.

➢ MovieApp.jsx – 컴포넌트 분리하기

📣 폴더 구조

MovieApp.jsx
- MovieList.jsx
- MovieItem.jsx
- MovieModal.jsx



📣 핵심

state는 부모에, UI는 자식에게

역할	담당 컴포넌트	설명
상태 관리 (데이터 중심)	MovieApp	movies, selectedMovie, title, review 등 모두 여기 있음
리스트 UI	MovieList	부모의 movies를 props로 받아 리스트로 출력
단일 항목 UI	MovieItem	props로 받은 movie 표시 및 버튼 이벤트 전달
모달 UI	MovieModal	선택된 영화 데이터 표시 및 닫기 버튼 이벤트 전달

<초기 출력 화면 >

<추가 화면>

<상세보기 화면>

문제2] 다음의 조건에 만족하도록 React를 작성 하시오.

조건

① 상품 목록 map() 렌더링

② "담기" 클릭 시 장바구니 수량 증가

③ "장바구니 보기" 클릭 시 모달 오픈

④ 장바구니가 비어있으면 "비어있습니다" 출력

⑤ 컴포넌트 구조

<ShopApp /> // 부모

├ <ProductList /> // 상품 목록

├ <ProductItem /> // 각 상품

└ <CartModal /> // 장바구니 모달

아래 오브젝트 배열을 이용해 작성하세요

```
const [products] = useState([
    { id: 1, name: '노트북', price: 1200000 },
    { id: 2, name: '마우스', price: 25000 },
    { id: 3, name: '키보드', price: 50000 },
  ]);
```

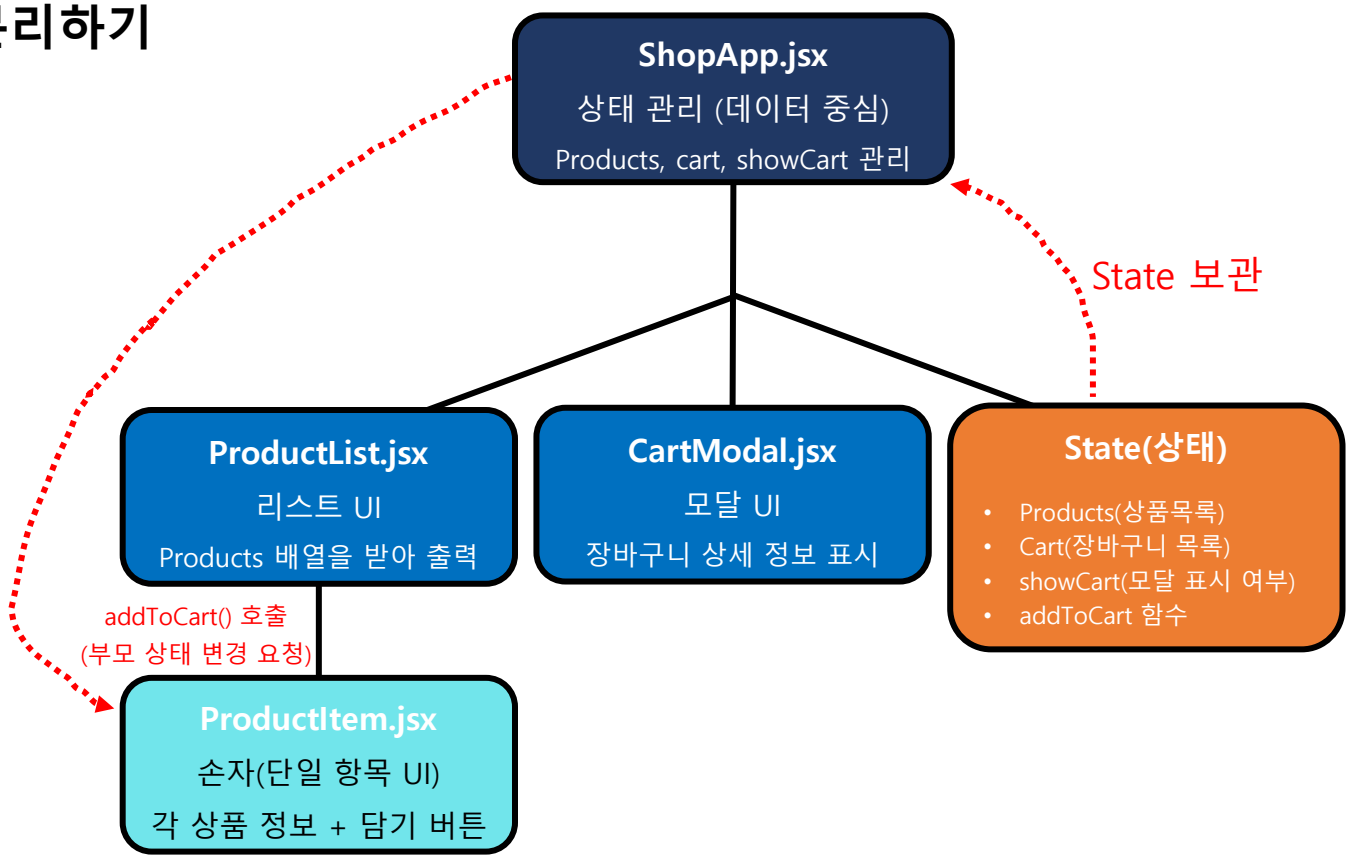
⑥ src 폴더안에 -> stateUI 폴더 생성 -> shop 폴더 생성 -> ProductList.jsx, ProductItem.jsx, CartModal.jsx 파일을 생성하여 작성하시오.

⑦ src 폴더안에 -> App.jsx에 작성한 ProductList.jsx를 import하여 실행 하시오.

➢ ShopApp.jsx – 컴포넌트 분리하기

📣 폴더 구조

ShopApp.jsx
- ProductList.jsx
- ProductItem.jsx
- CartModal.jsx



📣 핵심

ProductList → **Cart**

id	name	price
1	노트북	120000
2	마우스	25000
3	키보드	50000

id	name	price	quantity
1	노트북	120000	2
2	마우스	25000	1

<초기 출력 화면 >

쇼핑몰

장바구니 보기

- 노트북 - 1200000원 담기
- 마우스 - 25000원 담기
- 키보드 - 50000원 담기

<노트북 담기>

쇼핑몰

장바구니 보기

- 노트북 - 1200000원 담기
- 마우스 - 25000원 담기
- 키보드 - 50000원 담기

<장바구니 보기>

<노트북 두 번 담기>

다음의 조건에 만족하도록 React를 작성 하시오.

조건

① 음식 주문 하기

② foods : [{ id, name, price }] 형태로 초기 음식 데이터를 담는다. (각 개인들이 만들어 사용한다.)

③ cart : [{ id, name, price, quantity }] 형태로 장바구니에 담고, 담긴 음식 관리 한다.

④ isCartOpen : flase(Boolean), 장바구니 모달 열림/닫힘 기능을 구현한다.

⑤ 음식 목록에서 "담기" 버튼 클릭 시 장바구니에 추가하는 기능 구현한다.

⑥ 이미 담긴 음식이면 수량만 +1을하는 기능 구현한다.

⑦ 장바구니 버튼 클릭 시 CartModal이 열리도록 기능 구현한다

⑧ CartModal에서 수량 증가/감소 버튼 클릭 시 cart 상태가 업데이트되도록 기능 구현한다.

⑨ 수량감소는 1까지만, 수량증가는 10개까지만 가능하다.

⑩ 컴포넌트 구조

```
food/
    ├─ FoodApp.jsx      // 최상위 컴포넌트
    ├─ FoodList.jsx     // 음식 목록 렌더링
    └─ CartModal.jsx    // 장바구니 모달
```

⑥ src 폴더안에 -> stateUI 폴더 생성 -> food 폴더 생성 -> FoodApp.jsx, FoodList.jsx , CartModal.jsx 파일을 생성하여 작성하시오.

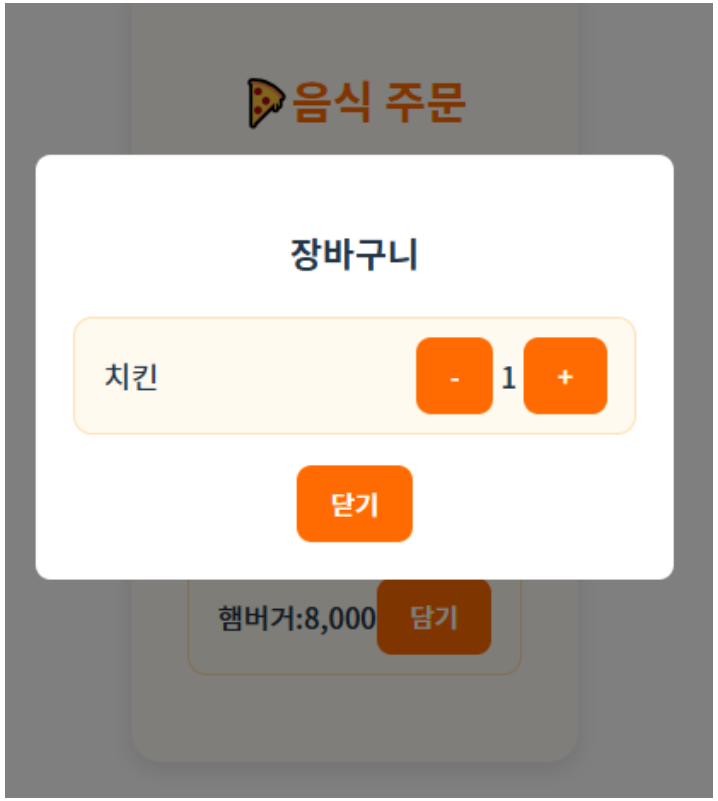
⑦ src 폴더안에 -> App.jsx에 작성한 FoodApp.jsx를 import하여 실행 하시오.

힌트

- 장바구니에 담긴 수량 증가(+) / 감소(-) 버튼을 클릭할 때마다 증가 감소되는 update 함수를 부모에 작성한다.
- 생성한 update 함수를 cartModal로 데이터를 전달한다.
- 전달받은 uapdate 함수를 이용하여 cartModal.에 수량 증가함수와 수량 감소 함수를 작성한다.

<주문 화면>

<장바구니 화면>



<장바구니 수량 증가 화면>

추가 이벤트 UI 적용하기

onMouseEnter, onMouseLeave, Hover 효과

➢ 마우스 이벤트 종류와 차이점

```
// onMouseOver - 요소나 자식 요소 위로 마우스가 올라갈 때마다 발생
   const handleMouseOver = () => {setMsg('마우스가 올라왔습니다 (Over)'); };
```

```
// onMouseEnter - 요소 위로 마우스가 처음 진입할 때만 발생 (자식 요소 무시)
  const handleMouseEnter = () => { setMsg('마우스가 진입했습니다 (Enter)'); };
```

```
// onMouseLeave - 요소에서 마우스가 벗어날 때 발생
  const handleMouseLeave = () => { setMsg('마우스가 벗어났습니다 (Leave)'); };
```

```
// onMouseOut - 요소나 자식 요소에서 마우스가 벗어날 때 발생
  const handleMouseOut = () => { setMsg('마우스가 나갔습니다 (Out)'); };
```

➢ 호버 효과 (Hover Effect)

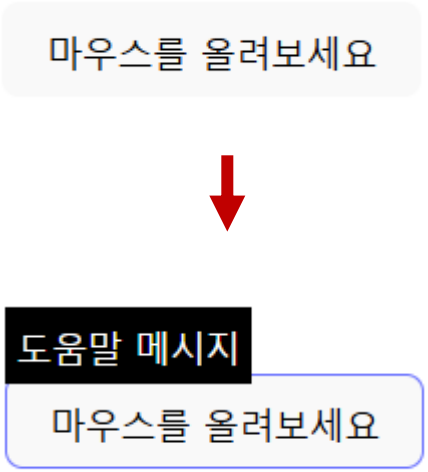
<예시 코드 >

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


export default function HoverCard() {
  const [isHovered, setIsHovered] = useState(false);


  return (
    <div
      onMouseEnter={() => setIsHovered(true)}
      onMouseLeave={() => setIsHovered(false)}
      style={{
        backgroundColor: isHovered ? 'blue' : 'gray',
        padding: '20px',
        transition: 'all 0.3s',
        cursor: 'pointer',
      }}
    >
      호버하면 색이 변합니다
    </div>
  );
}
```

➢ 툴팁 (Tooltip)

<예시 코드 > 실무에서는 onMouseEnter + onMouseLeave 조합을 주로 사용!

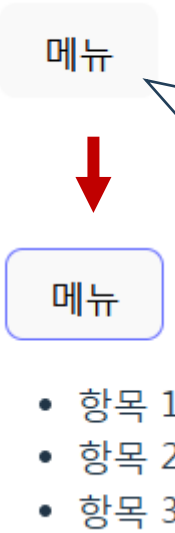
```
export default function TooltipExample() {
  const [showTooltip, setShowTooltip] = useState(false);
  return (
    <div style={{ position: 'relative' }}>
      <button
        onMouseEnter={() => setShowTooltip(true)}
        onMouseLeave={() => setShowTooltip(false)}
      >
        마우스를 올려보세요
      </button>
      {showTooltip && (
        <div
          style={{
            position: 'absolute',
            top: '-30px',
            backgroundColor: 'black',
            color: 'white',
            padding: '5px',
          }}
        >
          도움말 메시지
        </div>
      )}
    </div>
  );
}
```



➢ 드롭다운 메뉴

<예시 코드 >

```
export default function Dropdown() {
  const [isOpen, setIsOpen] = useState(false);


  return (
    <div
      onMouseEnter={() => setIsOpen(true)}
      onMouseLeave={() => setIsOpen(false)}
    >
      <button>메뉴</button>
      {isOpen && (
        <ul>
          <li>항목 1</li>
          <li>항목 2</li>
          <li>항목 3</li>
        </ul>
      )}
    </div>
  );
}
```



- 드롭다운이 로봇처럼 깜빡거리는 이유는 마우스 움직임에 너무 빨리 반응하기 때문
- 실제로는 마우스가 벗어나도 0.3초 정도 기다렸다가 닫히게 만들면 훨씬 자연스럽다. 이걸 **디바운스(Debounce)**라 한다.

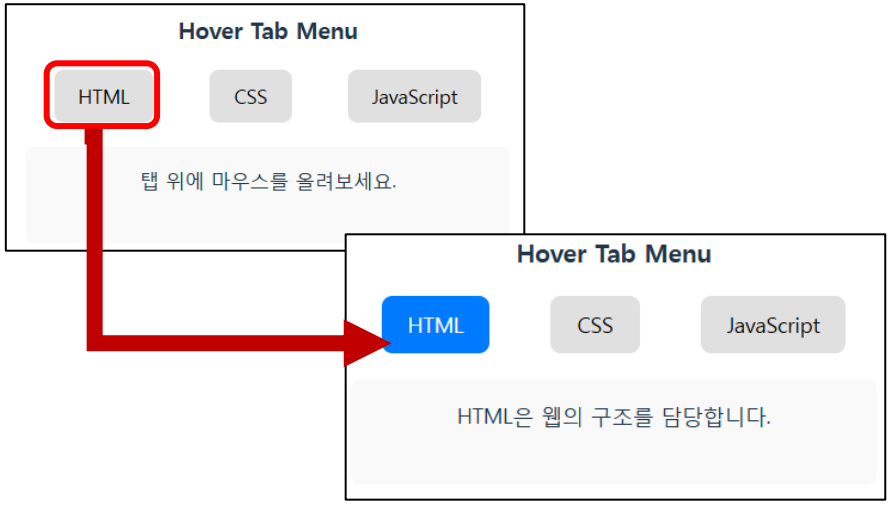
➤ 드롭다운 메뉴 디바운스 해결

<예시 코드 >

```jsx
export default function Dropdow02() {
  const [isOpen, setIsOpen] = useState(false);
  return (
    <div
      onMouseEnter={() => setIsOpen(true)}
      onMouseLeave={() => setIsOpen(false)}
      style={{ position: 'relative', display: 'inline-block' }}
    >
      <button>메뉴 ▼</button>
      {/* 조건부 렌더링 대신 항상 렌더링하되 CSS로 숨김 */}
      <ul
        style={{
          display: isOpen ? 'block' : 'none',
          position: 'absolute',
          top: '100%',
          left: 0,
          backgroundColor: 'white',
          border: '1px solid #ddd',
          listStyle: 'none',
          padding: '8px 0',
          margin: '0',
          minWidth: '150px',
        }}
      >
        <li style={{ padding: '8px 16px' }}>항목 1</li>
        <li style={{ padding: '8px 16px' }}>항목 2</li>
        <li style={{ padding: '8px 16px' }}>항목 3</li>
      </ul>
    </div>
  );
}
```

➢ 탭 메뉴

<예시 코드 >




```
export default function HoverTabMenu() {
  const [activeTab, setActiveTab] = useState(null);


  const tabs = [
    { id: 1, title: 'HTML', content: 'HTML은 웹의 구조를 담당합니다.' },
    { id: 2, title: 'CSS', content: 'CSS는 스타일을 담당합니다.' },
    { id: 3, title: 'JavaScript', content: 'JS는 동작을 담당합니다.' },
  ];


  return (
    <div style={{ width: '400px', margin: '50px auto' }}>
      <h3>Hover Tab Menu</h3>
      <div style={{ display: 'flex', justifyContent: 'space-around' }}>
        {tabs.map((tab) => (
          <div
            key={tab.id}
            onMouseEnter={() => setActiveTab(tab.id)} // 마우스 올리면 활성화
            onMouseLeave={() => setActiveTab(null)} // 마우스 벗어나면 비활성화
            style={{
              padding: '10px 20px',
              borderRadius: '8px',
              backgroundColor: activeTab === tab.id ? '#007bff' : '#e0e0e0',
              color: activeTab === tab.id ? 'white' : 'black',
              cursor: 'pointer',
              transition: '0.3s',
            }}
          >
            {tab.title}
          </div>
        ))}
      </div>
```

```
      {/* 탭 내용 영역 */}
      <div
        style={{
          marginTop: '20px',
          padding: '15px',
          backgroundColor: '#f9f9f9',
          borderRadius: '8px',
          minHeight: '50px',
        }}
      >
        {activeTab
          ? tabs.find((tab) => tab.id === activeTab)?.content
          : '탭 위에 마우스를 올려보세요.'}
      </div>
    </div>
  );
}
```

onMouseEnter, onMouseLeave, Hover 효과 연습 문제

문제1] [이미지 변경]버튼 클릭 시 이미지 변경되도록 작성

단, poodle.png, pome.png 이용해 작성

이미지 변경

이미지 변경

문제2] 기본 이미지 위에 마우스 hover 시 다른 이미지로 변경되도록 작성

단, tree-5.jpg ,tree-5.jpg 이용해 작성

문제3] 3장의 이미지를 클릭할 때마다 순서대로 바뀌도록 작성
단, imgs 배열을 이용해 작성

```
const imgs = [
    '/images/coffee-gray.jpg',
    '/images/coffee-blue.jpg',
    '/images/coffee-pink.jpg',
  ];
```



문제4] 아래 주어진 imgs 객체배열 이용해 category가 1번이면 나무, category가 2번이면 배경 이미지로 전환 되도록 작성 (단, 배열의 filter() 함수 이용해 작성)

배열이름.filter()
무엇을 반환하나: **조건을 만족하는 모든 요소들의 새로운 배열**. 만족하는 요소가 없으면 **빈 배열 [ ] 반환**.
용도: 여러 항목을 추출할 때(예: 삭제, 검색 결과, 특정 조건의 모든 요소 추출).
원본 배열 변경 여부: 원본을 변경하지 않고 새 배열을 반환.

배열이름.filter((요소, 인덱스, 배열) => { return 조건식; });

```
const imgs = [
    { id: 1, name: 'pic-1.jpg', category: 1 },
    { id: 2, name: 'pic-2.jpg', category: 1 },
    { id: 3, name: 'pic-3.jpg', category: 1 },
    { id: 4, name: 'pic-4.jpg', category: 1 },
    { id: 5, name: 'tree-1.jpg', category: 2 },
    { id: 6, name: 'tree-2.jpg', category: 2 },
    { id: 7, name: 'tree-3.jpg', category: 2 },
    { id: 8, name: 'tree-4.jpg', category: 2 },
  ];
```