State 관리하기

명령형 vs 선언형 with 라면

1. 냄비에 물 550ml를 붓는다.
2. 가스레인지 앞으로 이동하여 그 위에 냄비를 올려놓는다.
3. 가스불을 켠다.
4. 물이 끓는지 확인한다. (반복)
5. (만약)물이 끓으면 라면 봉지를 뜯고 내용물을 냄비안에 넣는다.
6. 3분 동안 기다린다.
7. 가스불을 끈다.

1. 라면 하나요~
2. 요리사가 알아서 준비

UI를 선언적인 방식으로 생각하려면...

1. 컴포넌트의 다양한 시각적 state를 확인하기
2. 무엇이 state 변화를 트리거하는지 알아내기
3. useState를 사용해서 메모리의 state를 표현하기
4. 불필요한 state 변수를 제거하기
5. state 설정을 위해 이벤트 핸들러 연결하기

```
const [answer, setAnswer] = useState('');
const [error, setError] = useState(null);
const [status, setStatus] = useState('typing');

if (status === 'success') {
  return <h1>That's right!</h1>
}

async function handleSubmit(e) {
  e.preventDefault();
  setStatus('submitting');
  try {
    await submitForm(answer);
    setStatus('success');
  } catch (err) {
    setStatus('typing');
    setError(err);
  }
}

function handleTextareaChange(e) {
  setAnswer(e.target.value);
}
```

State 구조화

1. 연관된 state 그룹화 하기

```
const [x, setX] = useState(0);
const [y, setY] = useState(0);
```

```
const [position,setPosition] = useState({
  x: 0,
  y: 0
})
```

2. state 모순 피하기

```
const [isSending,setIsSending] = useState(false);
const [isSent,setIsSent] = useState(false);
```

```
const [status, setStatus] = useState('typing'); // typing or sending or sent
```

State 구조화

3. 불필요한 state 피하기

```
const [firstName, setFirstName] = useState('');
const [lastName, setLastName] = useState('');
const [fullName, setFullName] = useState('');

function handleFirstNameChange(e) {
  setFirstName(e.target.value);
  setFullName(e.target.value + ' ' + lastName);
}

function handleLastNameChange(e) {
  setLastName(e.target.value);
  setFullName(firstName + " " + e.target.value);
}
```



```
const [firstName, setFirstName] = useState('');
const [lastName, setLastName] = useState('');

const fullName = firstName + " " + lastName;
```

State 구조화

4. State 중복 피하기

```
const initialItems = [
  { title: 'pretzels', id: 0 },
  { title: 'crispy seaweed', id: 1 },
  { title: 'granola bar', id: 2 },
];

export default function Menu() {
  const [items, setItems] = useState(initialItems);
  const [selectedItem, setSelectedItem] = useState(
    items[0]
  );

}
```

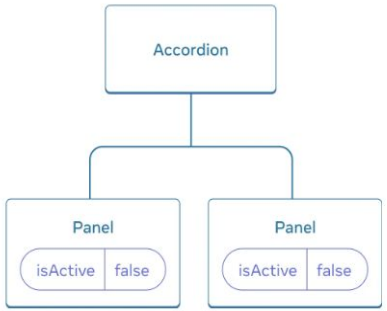


```
const initialItems = [
  { title: 'pretzels', id: 0 },
  { title: 'crispy seaweed', id: 1 },
  { title: 'granola bar', id: 2 },
];

export default function Menu() {
  const [items, setItems] = useState(initialItems);
  const [selectedId, setSelectedId] = useState(0);

  const selectedItem = items.find(item =>
    item.id === selectedId
  );

}
```

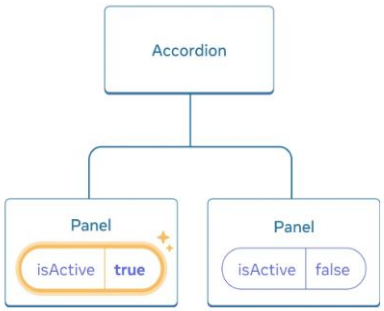
5. 깊게 중첩된 state 피하기 => 평탄화 작업 하기

컴포넌트 간 state 공유하기

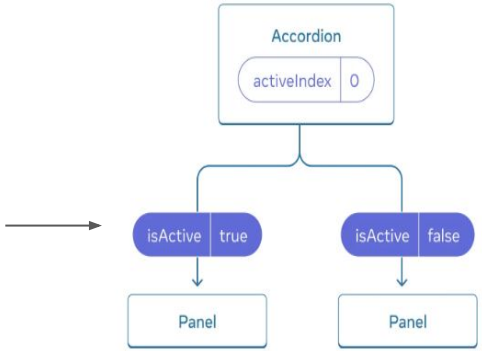
1. 공유하고자 하는 state 제거하기
2. 공유하려는 컴포넌트 사이에 가장 가까이에 위치한 상위 컴포넌트 찾기
3. 상위 컴포넌트에 state 추가하기
4. 하위 컴포넌트에 값 뿌려주기



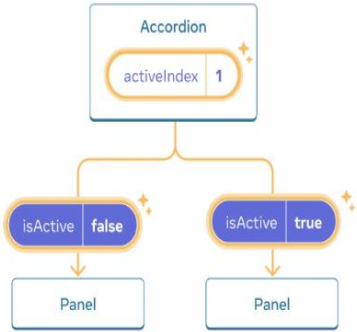
처음에는 각 `Panel`의 `isActive` state가 `false`이기 때문에 두 컴포넌트 모두 닫힌 상태로 보입니다.



두 `Panel`의 버튼 중 어느 것을 클릭하더라도 클릭한 해당 `Panel`의 `isActive` state만 변경됩니다.



처음에 `Accordion`의 `activeIndex`는 `0`이므로 첫 번째 `Panel`은 `isActive = true`를 받습니다.



`Accordion`의 `activeIndex` state가 `1`로 변경되면 두 번째 `Panel`은 `isActive = true`를 받게 됩니다.

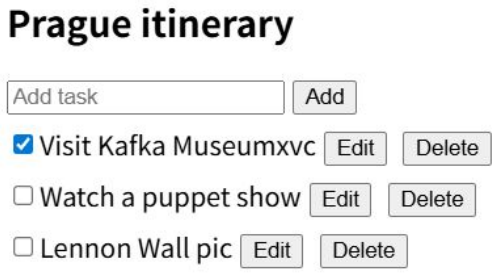
state 보존하고 초기화하기

1. 같은 자리 같은 컴포넌트 => state 보존
2. 같은 자리 다른 컴포넌트 => state 초기화
3. 같은 위치에서 state를 초기화
   a. 다른 위치에 컴포넌트를 렌더링
   b. key를 이용해 state를 초기화

Reducer

한 컴포넌트에서 state 업데이트가 여러 이벤트 핸들러로 분산되어 있을때, 하나로 묶어서 관리하기 위해 사용된다.

```
const [tasks, setTasks] = useState(initialTasks);

function handleAddTask(text) {
  setTasks([...tasks, {
    id: nextId++,
    text: text,
    done: false
  }]);
}

function handleChangeTask(task) {
  setTasks(tasks.map(t => {
    if (t.id === task.id) {
      return task;
    } else {
      return t;
    }
  }));
}

function handleDeleteTask(taskId) {
  setTasks(
    tasks.filter(t => t.id !== taskId)
  );
}
```



Reducer

```
const [tasks, setTasks] = useState(initialTasks);
```

```
function handleAddTask(text) {
  setTasks([...tasks, {
    id: nextId++,
    text: text,
    done: false
  }]);
}
```

```
function handleChangeTask(task) {
  setTasks(tasks.map(t => {
    if (t.id === task.id) {
      return task;
    } else {
      return t;
    }
  }));
}
```

```
function handleDeleteTask(taskId) {
  setTasks(
    tasks.filter(t => t.id !== taskId)
  );
}
```

```
const [tasks, tasksDispatch] = useReducer(
  tasksReducer,
  initialTasks
);
```

```
function handleAddTask(text) {
  tasksDispatch({
    type: 'added',
    id: nextId++,
    text: text,
  });
}
```

```
function handleChangeTask(task) {
  tasksDispatch({
    type: 'changed',
    task: task
  });
}
```

```
function handleDeleteTask(taskId) {
  tasksDispatch({
    type: 'deleted',
    id: taskId
  });
}
```

```
export default function tasksReducer(tasks, action) {
  switch (action.type) {
    case 'added': {
      return [...tasks, {
        id: action.id,
        text: action.text,
        done: false
      }];
    }
    case 'changed': {
      return tasks.map(t => {
        if (t.id === action.task.id) {
          return action.task;
        } else {
          return t;
        }
      });
    }
    case 'deleted': {
      return tasks.filter(t => t.id !== action.id);
    }
    default: {
      throw Error('Unknown action: ' + action.type);
    }
  }
}
```

Context를 사용해 데이터 전달하기

1. Context 생성하기

```
export const DarkModeContext = createContext(null);
```

2. Context 제공하기

```
export default function App() {
  const [isDay,setIsDay] = useState(true)
  const value = {
      isDay: isDay,
      clickBtn: ()=> setIsDay(!isDay)
    }

  return (
    <DarkModeContext value={value}>
      ...
    </DarkModeContext>
  )
}
```

3. Context 사용하기

```
const {isDay, clickBtn} = useContext(DarkModeContext);
```

Reducer와 Context로 앱 확장하기

```jsx
const TasksContext = createContext(null);
const TasksDispatchContext = createContext(null);

export function TasksProvider({children}) {
  const [tasks, dispatch] = useReducer(tasksReducer,initialTasks);

  return (
    <TasksContext value={tasks}>
      <TasksDispatchContext value={dispatch}>
        {children}
      </TasksDispatchContext>
    </TasksContext>
  );
}

export function useTasks() {
  return useContext(TasksContext);
}

export function useTasksDispatch() {
  return useContext(TasksDispatchContext);
}
```