Chapter 13. 합성 vs 상속

교재: 처음만난 리액트 (저자: 이인제, 한빛출판사)

Contents

- CHAPTER 13: 합성 vs 상속

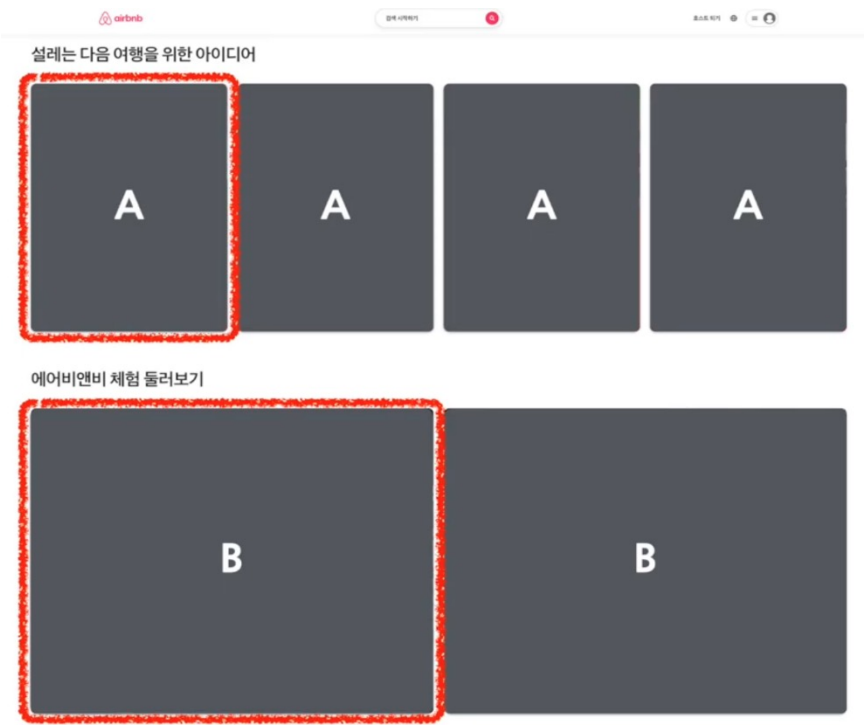
13.1 합성에 대해 알아보기

13.2 상속에 대해 알아보기

13.3 (실습) Card 컴포넌트 만들기

SECTION 13.1 합성에 대해 알아보기

- **합성(Composition)**
  - 여러 개의 컴포넌트를 합쳐서 새로운 컴포넌트를 만드는 것



컴포넌트 A + 컴포넌트 B = 페이지 컴포넌트

  - 여러 개의 컴포넌트를 어떻게 조합할 것인가?

SECTION 13.1 합성에 대해 알아보기

- **Containment**
  - 하위 컴포넌트를 포함하는 형태의 합성 방법
  - Sidebar, Dialog 같은 박스 형태의 컴포넌트는 어떤 자식 엘리먼트가 들어올지 예상할 수 없음

    ex. 동일한 사이드바 컴포넌트를 사용하는 두 개의 쇼핑몰 – 의류 메뉴, 식료품 메뉴

  - 이러한 컴포넌트에서는 Containment 합성 방법을 사용
    - 특수한 children prop을 사용하여 자식 엘리먼트를 출력에 그대로 전달

```jsx
function FancyBorder(props) {
  return (
    <div className={`FancyBorder FancyBorder-${props.color}`}>
      {props.children}
    </div>
  )
}
```

SECTION 13.1 합성에 대해 알아보기

- **Containment 예제**
  - 자식 엘리먼트를 prop으로 받아서 테두리로 감싸주는 컴포넌트

```jsx
import './FancyBorder.css';

function FancyBorder(props) {
  return (
    <div className={`FancyBorder FancyBorder-${props.color}`}>
      {props.children}
    </div>
  )
}

export default FancyBorder;
```

```css
.FancyBorder {
  padding: 10px 10px;
  border: 10px solid;
}

.FancyBorder-blue {
  border-color: blue;
}

.FancyBorder-yellow {
  border-color: yellow;
}
```

SECTION 13.1 합성에 대해 알아보기

◦ Containment 예제

- FancyBorder 컴포넌트를 이용한 Dialog 컴포넌트

```jsx
import FancyBorder from './FancyBorder';

function WelcomeDialog(props) {
  return (
    <FancyBorder color="blue">
      <h1>
        어서오세요
      </h1>
      <p>
        우리 사이트에 방문하신 것을 환영합니다!
      </p>
    </FancyBorder>
  )
}

export default WelcomeDialog;
```

```jsx
import FancyBorder from './FancyBorder';
import Calculator from '../12/Calculator';

function WelcomeDialog(props) {
  return (
    <FancyBorder color="yellow">
      <h2>
        Calculator
      </h2>
      <Calculator />
    </FancyBorder>
  )
}

export default WelcomeDialog;
```

➢ 리액트에서는 props.children을 통해 하위 컴포넌트를 하나로 모아서 제공해 줌

SECTION 13.1 합성에 대해 알아보기

- **Containment**
  - 여러 개의 children 집합이 필요한 경우
    - props를 정의해서 각각 원하는 컴포넌트를 전달

```
function SplitPane(props) {
  return (
    <div className="SplitPane">
      <div className="SplitPane-left">
        {props.left}
      </div>
      <div className="SplitPane-right">
        {props.right}
      </div>
    </div>
  );
}
```

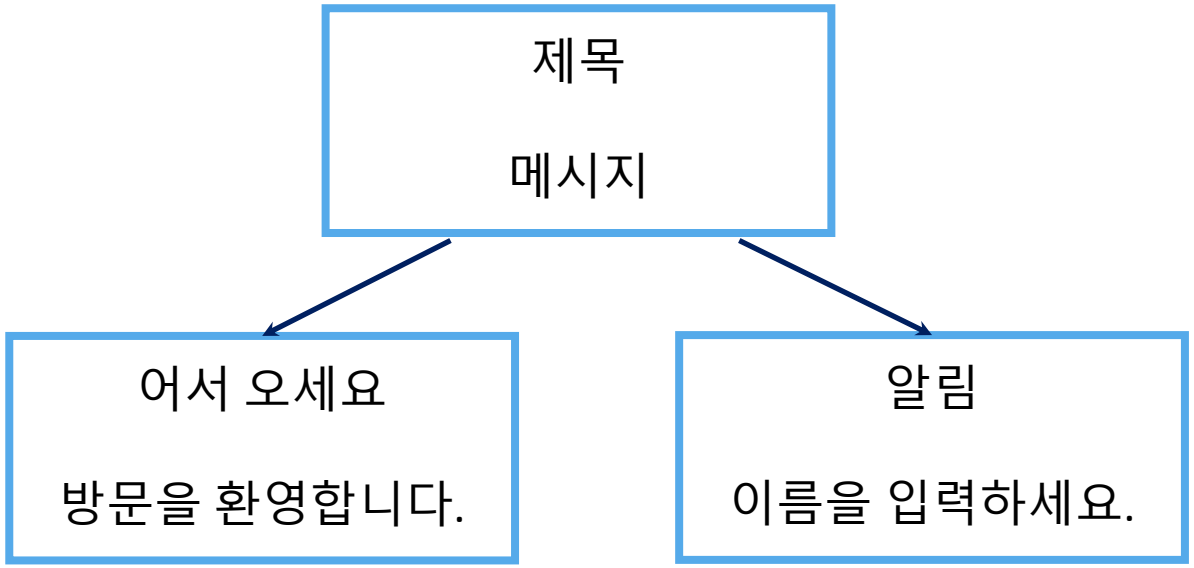
```
function App(props) {
  return (
    <SplitPane
      left={
        <Contacts />
      }
      right={
        <Chat />
      }
    />
  );
}
```

  - props.children이나 직접 정의한 props를 이용하여 하위 컴포넌트를 포함하는 합성 방법

SECTION 13.1 합성에 대해 알아보기

- **Specialization**
  - 범용적인 컴포넌트를 만들어 놓고 이를 특수화(구체화)시켜서 컴포넌트를 합성하는 방식

  - 범용적인 Dialog 컴포넌트 – 모든 종류의 Dialog를 포함하는 개념
  - Welcome Dialog는 환영을 위한 구체화된 Dialog 컴포넌트
  - Alert Dialog는 경고를 위한 구체화된 Dialog 컴포넌트



SECTION 13.1 합성에 대해 알아보기

- **Specialization 예제**

```jsx
import FancyBorder from './FancyBorder';

function Dialog(props) {
  return (
    <FancyBorder color="blue">
      <h1>
        {props.title}
      </h1>
      <p>
        {props.message}
      </p>
    </FancyBorder>
  )
}
```

범용적인 Dialog 컴포넌트

```jsx
function WelcomeDialog(props) {
  return (
    <Dialog
      title="어서 오세요"
      message="우리 사이트에 방문하신 것을 환영합니다!!"
    />
  );
}

export default WelcomeDialog;
```

구체화된 WelcomeDialog 컴포넌트

SECTION 13.1 합성에 대해 알아보기

- **Containment와 Specialization을 같이 사용하기**

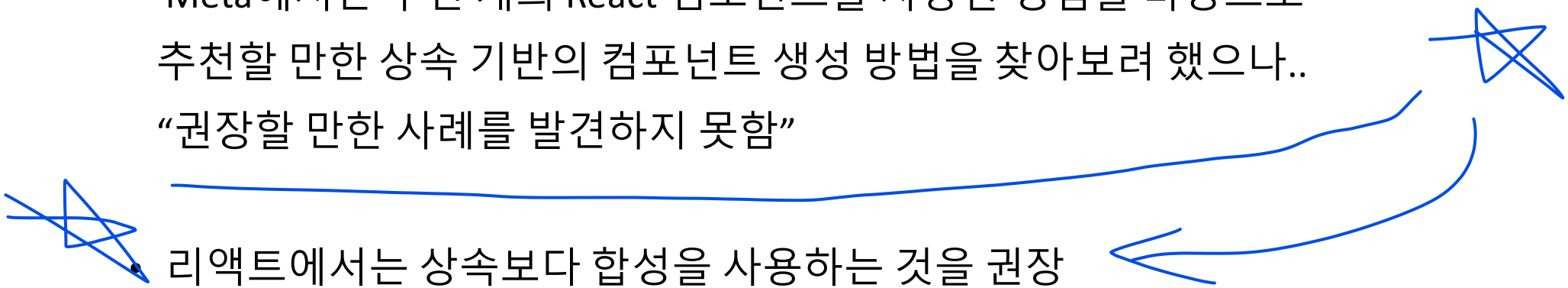
```jsx
function Dialog(props) {
  return (
    <FancyBorder color="blue">
      <h1>
        {props.title}
      </h1>
      <p>
        {props.message}
      </p>
      {props.children}
    </FancyBorder>
  )
}
```

```jsx
function SignUpDialog(props) {
  const [nickname, setNickname] = useState('');

  const handleChange = (event) => {
    setNickname(event.target.value);
  }

  const handleSignUp = () => {
    alert(`어서 오세요, ${nickname}님!`);
  }

  return (
    <Dialog
      title="화성 탐사 프로그램"
      message="닉네임을 입력해 주세요.">
      <input
        value={nickname}
        onChange={handleChange} />
      <button onClick={handleSignUp}>
        가입하기
      </button>
    </Dialog>
  );
}
```

SECTION 13.2 상속에 대해 알아보기

- **Inheritance**
  - **객체지향 프로그래밍에서 부모 클래스의 속성, 함수 등을 물려받아 자식 클래스를 만드는 것**

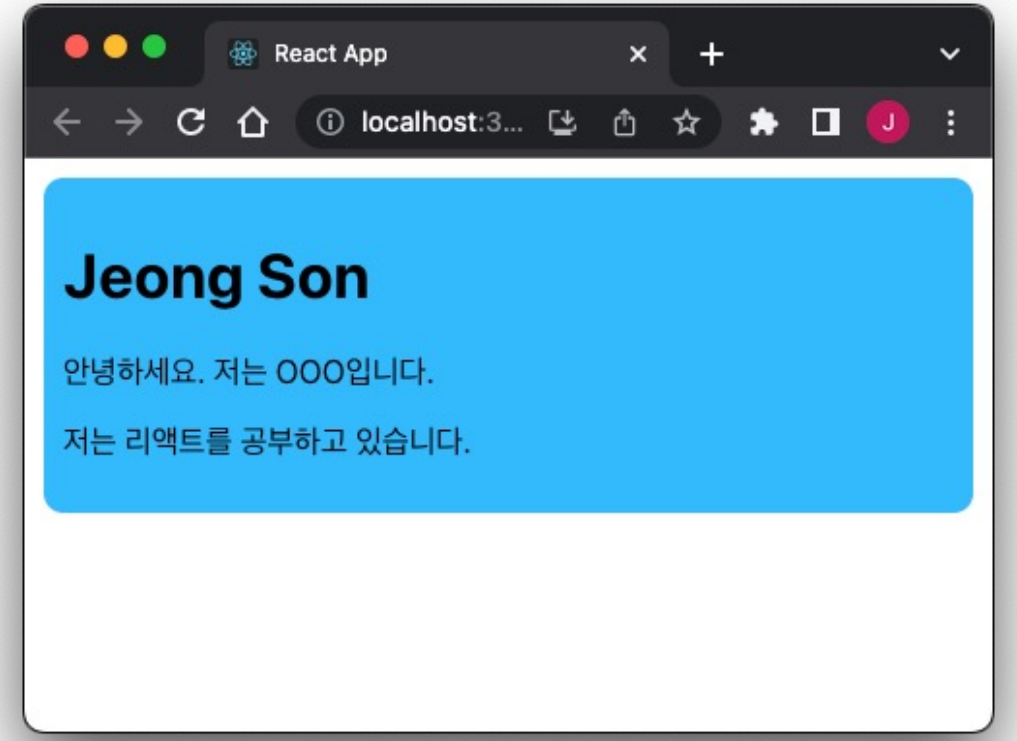
- Meta에서는 수천 개의 React 컴포넌트를 사용한 경험을 바탕으로
  추천할 만한 상속 기반의 컴포넌트 생성 방법을 찾아보려 했으나..
  "권장할 만한 사례를 발견하지 못함"



- 리액트에서는 상속보다 합성을 사용하는 것을 권장
- props와 합성은 명시적이고 안전한 방법으로 컴포넌트의 모양과 동작을 커스터마이징 하는데 필요한 모든 유연성을 제공

SECTION 13.3 Card 컴포넌트 만들기

- Containment와 Specialization을 사용한 Card Component



SECTION 13.3 Card 컴포넌트 만들기

- **Containment와 Specialization을 사용한 Card Component**

```jsx
function Card(props) {
  const { title, backgroundColor, children } = props;

  return (
    <div
      style={{
        margin: 10,
        padding: 10,
        borderRadius: 10,
        backgroundColor: backgroundColor || "white"
      }}
    >
      {title && <h1>{title}</h1>}
      {children}
    </div>
  )
}

export default Card;
```

```jsx
import Card from './Card';

function ProfileCard(props) {
  return (
    <Card title="Jeong Son" backgroundColor="#34bafc">
      <p>안녕하세요. 저는 000입니다.</p>
      <p>저는 리액트를 공부하고 있습니다.</p>
    </Card>
  )
}

export default ProfileCard;
```

[요약]

- 합성이란?
  - 여러 개의 컴포넌트를 합쳐서 새로운 컴포넌트를 만드는 것
  - 다양하고 복잡한 컴포넌트를 효율적으로 개발할 수 있음
- 합성 기법
  - Containment
    - 하위 컴포넌트를 포함하는 형태의 합성 방법
    - 리액트 컴포넌트의 특수한 props.children 속성을 사용
    - 여러 개의 children 집합이 필요한 경우 별도로 props를 각각 정의해서 사용
  - Specialization
    - 범용적인 개념을 구별되게 구체화하는 것
    - 범용적으로 쓸 수 있는 컴포넌트를 만들어 놓고 이를 구체화시켜서 컴포넌트를 사용하는 합성 방법
  - Containment와 Specialization을 함께 사용하기
    - props.children을 통해 하위 컴포넌트를 포함시키기(Containment)
    - 별도의 props를 선언하여 구체화시키기(Specialization)

[요약]

- 상속
  - 다른 컴포넌트로부터 상속받아서 새로운 컴포넌트를 만드는 것
  - 상속을 사용하여 컴포넌트를 만드는 것을 추천할 만한 사용 사례를 찾지 못함
  - 리액트에서는 상속이라는 방법을 사용하는 것보다는 합성을 사용하는 것이 더 좋음