리액트 중간 정리: 3-5

Chapter 3: JSX

JSX란?

- JavsScript를 확장한 문법, JavaScript의 모든 기능이 포함되어 있음
- React에서 JSX 사용이 필수가 아니지만, 간결한 코드로 UI 개발에 시각적인 도움을 줌
- JSX는 React 엘리먼트(element)를 생성

```
const element = <h1>Hello, {name}</h1>;
```

- JSX의 중괄호 안에는 유효한 모든 JavaScript 표현식을 넣을 수 있음

  ex. 2 + 2, user.firstName, formatName(user) 등

- JSX도 표현식
  - JSX를 if 구문 및 for loop 안에 사용하고, 변수에 할당하고, 인자로 받거나 함수로 반환할 수 있음

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

- JSX 속성 정의
  - 어트리뷰트에 따옴표를 이용해 문자열로 정의하거나, 중괄호를 이용해 JavaScript 표현식 삽입

```
const element = <a href="https://www.reactjs.org"> link </a>;
const element = <img src={user.avatarUrl}></img>;
```

  - JSX는 JavaScript에 가깝기 때문에, React DOM은 camelCase 프로퍼티 명명 규칙 사용

        - class → className, tabindex → tabIndex
- JSX로 자식 정의
    - 태그가 비어 있다면 xml처럼 />를 이용해서 닫아줌

```
const element = <img src={user.avatarUrl} />;
```

    - 자식을 포함할 수 있음

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

- JSX는 주입 공격(Injection Attaks) 방어
    - 기본적으로 React DOM은 JSX에 삽입된 모든 값을 렌더링하기 전에 이스케이프 처리
    - 애플리케이션에서 명시적으로 작성되지 않은 내용은 주입되지 않음
    - 모든 항목은 렌더링 되기 전에 문자열로 변환됨
    - 이런 특성으로 인해 XSS (cross-site-scripting) 공격을 방지할 수 있음
- JSX는 객체를 표현
    - Babel은 JSX를 React.createElement() 호출로 컴파일

> Babel - JavaScript 컴파일러(transpiler)
>
> - 최신 문법을 이전 버전 브라우저에서 사용 (하위 호환성을 위해)
> - typescript를 javascript로 컴파일 (** typescript - 정적 typing, es2015 기반 객체지향적 문법)
> - polyfill - 지원하지 않는 브라우저 제외

```
const element = (
  <h1 className="greeting">
```

```
    Hello, world!
  </h1>
);
```

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

Chapter 4: 엘리먼트 렌더링

엘리먼트

- 엘리먼트는 React 앱의 가장 작은 단위
- 엘리먼트는 화면에 표시할 내용을 기술
- 브라우저 DOM 엘리먼트와 달리 React 엘리먼트는 일반 객체이며(plain object) 쉽게 생성할 수 있음
- React DOM은 React 엘리먼트와 일치하도록 DOM을 업데이트 함
- 엘리먼트는 컴포넌트의 구성 요소

DOM에 엘리먼트 렌더링하기

- 루트(root) DOM 노드

```
<div id="root"></div>
```

  - 이 안에 들어가는 모든 엘리먼트를 React DOM에서 관리
  - React로 구현된 애플리케이션은 하나의 Root DOM 노드가 있음
- React 엘리먼트 렌더링
  - DOM 엘리먼트를 ReactDOM.createRoot()에 전달
  - React 엘리먼트를 root.render()에 전달

```
const root = ReactDOM.createRoot(
  document.getElementById('root')
);
const element = <h1>Hello, world</h1>;
root.render(element);
```

엘리먼트 업데이트

- React 엘리먼트는 불변객체 - 엘리먼트를 생성한 이후에는 해당 엘리먼트의 자식이나 속성을 변경할 수 없음
- 영화에서 하나의 프레임과 같이 특정 시점의 UI를 보여줌
- UI를 업데이트 하기 위해 새로운 엘리먼트를 생성하고 이를 root.render()에 전달
- React DOM은 해당 엘리먼트와 그 자식 엘리먼트를 이전의 엘리먼트와 비교하고, 필요한 경우에만 DOM을 업데이트 함 (시간 출력 예제)

Chapter 5: Component와 props

리액트 컴포넌트

- 리액트는 컴포넌트 기반 구조를 가짐
  - React 앱에서는 버튼, 폼, 다이얼로그, 화면 등의 모든 것들이 컴포넌트로 표현됨
  - 작은 컴포넌트들이 모여서 하나의 컴포넌트를 구성하고 이러한 컴포넌트들이 모여서 페이지를 구성
  - 컴포넌트를 통해 UI를 재사용 가능한 개별적인 여러 조각으로 나눌 수 있음
- 개념적으로 컴포넌트는 JavaScript 함수와 유사
  - props라는 임의의 입력을 받아서, 화면에 어떻게 표시되는지를 기술하는 React 엘리먼트를 반환함
- 컴포넌트의 종류
  - 함수 컴포넌트
    - JavaScript 함수 형태의 컴포넌트

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const Welcome = (props) => {
  return <h1>Hello, {props.name}</h1>;
}
```

- 클래스 컴포넌트
  - ES6 class를 사용하여 컴포넌트 정의

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

  - React 관점에서 두 가지 유형의 컴포넌트는 동일함
  - 컴포넌트의 이름은 항상 대문자로 시작
    - React는 소문자로 시작하는 컴포넌트를 DOM 태그로 처리

- 컴포넌트 렌더링
  - 사용자 정의 컴포넌트

```
const element = <Welcome name="Sara" />;
```

    - React가 사용자 정의 컴포넌트로 작성한 엘리먼트를 발견하면 JSX 어트리뷰트와 자식을 해당 컴포넌트에 단일 객체로 전달함 - 이 객체가 props
  - 렌더링 과정

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
const element = <Welcome name="Sara" />;
root.render(element);
```

1. <Welcome name="Sara" /> 엘리먼트로 root.render()를 호출

2. React는 {name: 'Sara'}를 props로 하여 Welcome 컴포넌트를 호출
3. Welcome 컴포넌트는 결과적으로 <h1>Hello, Sara</h1> 엘리먼트를 반환
4. React DOM은 <h1>Hello, Sara</h1> 엘리먼트와 일치하도록 DOM을 효율적으로 업데이트

- 컴포넌트 합성
  - 컴포넌트는 자신의 출력에 다른 컴포넌트를 참조할 수 있음
  - 여러 개의 컴포넌트를 합쳐서 하나의 컴포넌트를 만드는 것을 컴포넌트 합성이라 함
- 컴포넌트 추출
  - 큰 컴포넌트에서 일부를 추출해서 새로운 컴포넌트를 만드는 과정 (Comment 컴포넌트에서 Avatar, UserInfo 추출 예제)
  - 기능 단위로 구분하는 것이 좋고, 나중에 곧바로 재사용이 가능한 형태로 추출
  - UI 일부가 여러 번 사용되거나(Button, Panel, Avatar), UI 일부가 자체적으로 복잡한 경우(App, Comment) 별도의 컴포넌트로 만들어 두면 더 큰 앱에서 작업할 때 효율적으로 사용할 수 있음

Props

- Props 개념
  - 리액트 컴포넌트의 속성
  - 컴퍼넌트에 전달할 다양한 정보를 담고 있는 자바스크립트 객체
- Props 특징
  - 읽기 전용
    - 함수 컴포넌트나 클래스 컴포넌트 모두 컴포넌트의 자체 props를 수정해서는 안 됨
  - 모든 React 컴포넌트는 자신의 props를 다룰 때 반드시 순수 함수처럼 동작해야 함