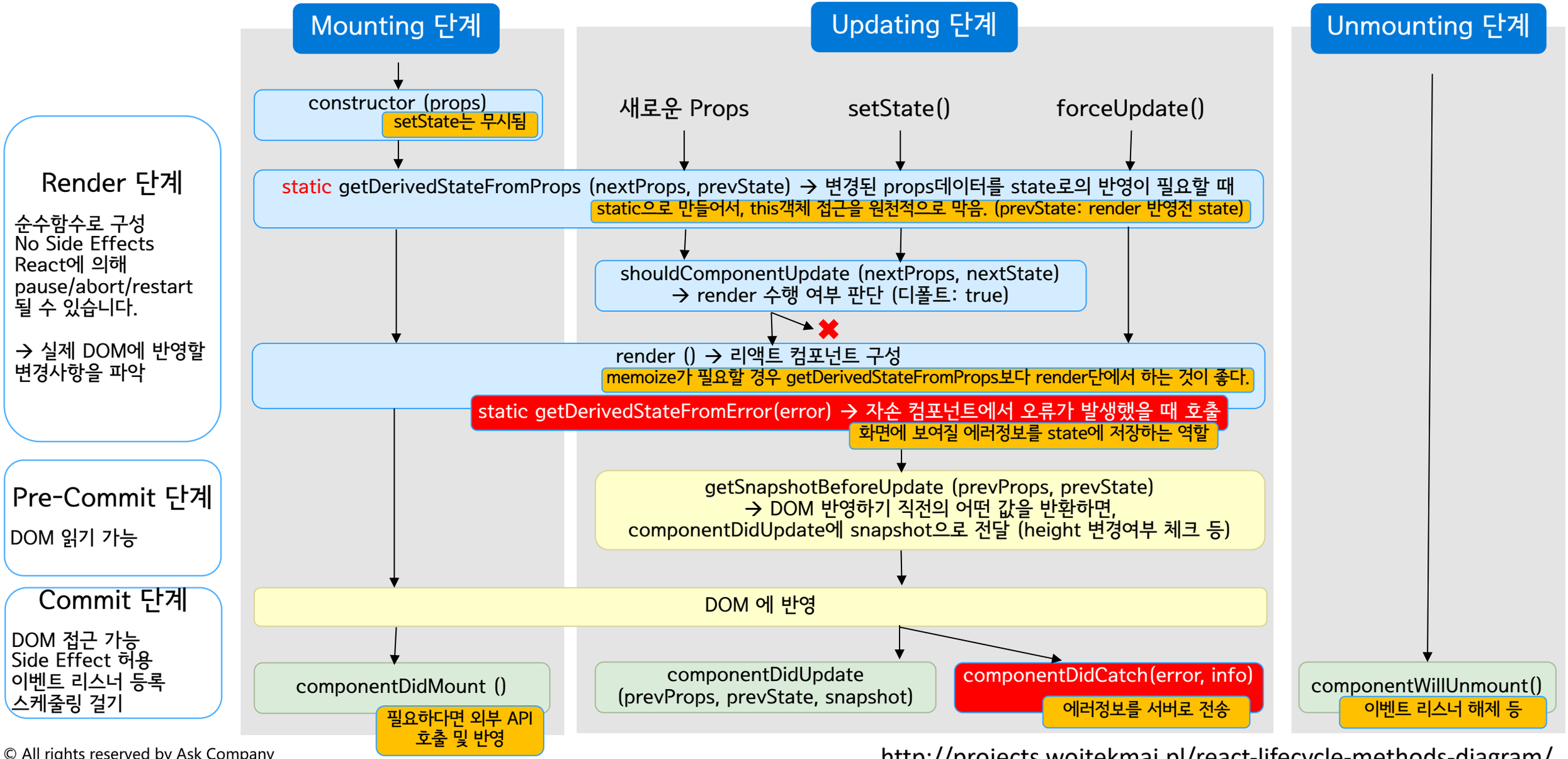
Ask Company

리액트와 함께 장고 시작하기 / 리액트

클래스 컴포넌트, 생명주기

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

클래스 컴포넌트, 생명 주기



http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

constructor (props)

초기 속성값으로부터 상탯값을 만들어낼 때 구현

초기 props에 대한 대응일 뿐, 변경되는 props에 대해서는 반영 X

생성자 내에서의 setState 호출은 무시됩니다. (mount이후에만 유효하기에)

외부 API 호출/반영이 필요하다면

마운트가 끝난 후인 componentDidMount()에서 구현하기를 추천 → 대개 setState 호출이 필요하기 때문

함수형 컴포넌트에서는 useEffect 훅을 활용

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      messageLength: this.props.message.length
    };
  }
```

```
class App extends React.Component {
```

Stage 3 단계의 클래스 필드 문법 적용

```
  state = {
    messageLength: this.props.message.length
  };
```

static getDerivedStateFromProps (nextProps, prevState)

정적 메서드로서, this 객체 접근을 원천적으로 봉쇄.

속성값 변화에 따라 외부 API 호출이 필요하다면

- this 객체 접근이 불허되기에 불가 → componentDidUpdate에서 구현

```
class App extends React.Component {
  componentDidUpdate(prevProps) {
    const { postId } = this.props;
    if ( prevProps.postId !== postId ) {
      // postId가 변경되었습니다.
    }
  }
```

일일이 상탯값마다
변경여부를 체크해야 … ☹

함수형 컴포넌트에서는
useEffect 훅으로 보다 쉬운 체크

속성값을 계산하여 상탯값에 반영이 필요할 때

- 불필요한 계산량을 줄이기 위해, 메모이제이션이 필요
- → render에서 lodash/memoize 팩키지를 활용하는 것을 추천 (구현이 단순해짐)

```
import memoize from 'lodash/memoize';

class App extends React.Component {
  getFavoritePostList = memoize(postList => (
    postList.filter(post => post.isFavorite)
  ));

  render() {
    const { postList } = this.props;
    const favoritePostList = this.getFavoritePostList(postList);
    생략...
  }
```

함수형 컴포넌트에서는
useMemo 훅을 사용

render()

함수형 컴포넌트에서는 함수 그자체

화면에 보여질 내용을 반환

반환 가능 타입 : 리액트 컴포넌트, Array(key 속성 필요), 문자열/숫자, null/bool 등

속성값과 상탯값만으로 반환값을 결정

순수함수로 구현, setState 호출하지 않기

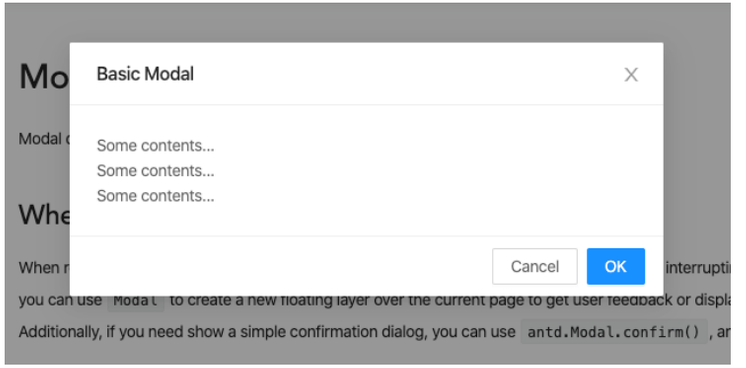
No side effects : 서버와의 통신 등

조건부 렌더링 → null/false을 반환

```
render() {
  const { message } = this.props;
  return message.length > 0 && <p>{message}</p>;
}
```

컴포넌트를 다른 특정 DOM요소에 렌더링이 필요할 때

return ReactDOM.createPortal(, domNode); → modal 처리 시에 유용



대개 Modal은 UI에 따라 컴포넌트 바깥에 렌더링이 되어야 합니다.

componentDidMount()

함수형 컴포넌트에서는 componentDidMount/
componentWillUnmount를 useEffect 훅을 통해 지원

이벤트 리스너 등록이 필요할 때

window.addEventListener ...

componentWillUnmount에서 필히 해제하기 : window.removeEventListener → 빼먹는 경우가 많아요. ☹

componentDidUpdate에서도 동일한 로직을 적용하는 경우가 많음.

함수형 컴포넌트에서는 둘을 하나로 구현 (useEffect 훅)

DOM을 읽을 수 있습니다. → ex) 픽셀 단위의 가로 크기 조회 (크기에 따른 배경색 지정)

직접적으로 setState가 필요하다면, Render단계에서 수행해주세요.

```
class App extends React.Component {
  state = { boxWidth: 0 };
  divRef = React.createRef();

  componentDidMount() {
    const rect = this.divRef.current.getBoundingClientRect();
    this.setState({ boxWidth: rect.width })
  }

  render() {
    return <div ref={this.divRef}>boxWidth: {this.state.boxWidth}</div>;
  }
}
```

마운트 당시의 width를 state에 반영
resize에 대응할려면, resize 이벤트 처리 필요합니다.

shouldComponentUpdate (nextProps, nextState)

render 수행 여부를 결정 (bool 반환) → 디폴트 true

렌더링 성능 최적화가 필요할 때 구현

getSnapshotBeforeUpdate (prevProps, prevState) => snapshot

render가 수행되기 이전의 DOM의 상탯값을 가져와서,
componentDidUpdate에서 비교할 목적

componentDidUpdate (prevProps, prevState, snapshot)

```
divRef = React.createRef();

getSnapshotBeforeUpdate(prevProps, prevState) {
  const rect = this.divRef.current.getBoundingClientRect();
  return rect.height;  // snapshot
}

componentDidUpdate(prevProps, prevState, snapshot) {
  if ( snapshot != null ) {
    const rect = this.divRef.current.getBoundingClientRect();
    if ( rect.height !== snapshot ) {
      console.log("height가 변경되었습니다.");
    }
  }
}
```

componentDidUpdate (prevProps, prevState, snapshot)

getSnapshotBeforeUpdate에서 남긴 snapshot을 활용할 수 있음.

componentDidMount와 비슷한 로직을 구현하는 경우가 많음.

ex) 외부 API 호출

```
componentDidMount() {
  const { post } = this.props;
  this.setCommentList(post);
}

componentDidUpdate(prevProps) {
  const { post } = this.props;
  if ( prevProps.post !== post )
    this.setCommentList(post);
}

setCommentList(post) {
  requestCommentList(post)
    .then(commentList => this.setState({ commentList }));
}
```

참고) 함수형 컴포넌트에서의 useEffect 훅에서는 하나의 루틴에서 구현하게 됩니다.

componentWillUnmount

componentDidMount와 쌍으로

이벤트 리스너를 등록했었다면, 이벤트 리스너를 필히 해제.

타이머를 걸어뒀다면, 타이머 필히 해제

이외에도 다양한 리소스 해제가 필요할 경우 구현

```
componentDidMount() {
  this.timerHandle = setInterval(() => {
    console.log("1초 단위 호출")
  }, 1000);
}

componentWillUnmount() {
  clearInterval(this.timerHandle);
}
```

클래스형 컴포넌트 추천 작성 순서

1. propTypes 타입 정의
2. state 초기화
3. render를 제외한 생명주기 메서드
4. 생명주기 메서드를 제외한 나머지 메서드
5. render 메서드
6. 컴퍼넌트 외부에서 정의하는 변수와 함수

새롭게 시작하는 리액트 프로젝트에서는
함수형 컴포넌트를 쓰세요.

하지만, 클래스형 컴포넌트에 대한 이해는
꼭 필요합니다.

출처: 실전 리액트 프로그래밍 (프로그래밍 인사이트), p166

Life is short.
You need Python and Django.

I will be your pacemaker.