19. 📘 DOM 트리 구조를 따라 부모나 형제 요소를 구조적으로 찾기

📌 1. DOM은 가족처럼 구성된다 – 부모, 자식, 형제 관계란?

DOM(Document Object Model)은 트리(tree) 구조입니다. 즉, 어떤 노드를 기준으로 보면 위에는 부모가 있고, 아래에는 자식이 있으며, 옆에는 형제 노드들이 있습니다. 이 관계는 마치 가족처럼 이해할 수 있습니다.

```
<div id="card">
  <h2>제목</h2>
  <p>설명</p>
  <button>클릭</button>
</div>
```

위 구조에서 `<button>` 요소를 기준으로 주변을 탐색해보면 다음과 같습니다:

```
const btn = document.querySelector("button");

console.log(btn.parentNode);            // 부모 요소: <div id="card">
console.log(btn.previousElementSibling); // 바로 앞 형제 요소: <p>
console.log(btn.parentNode.children);    // 자식 요소들: [h2, p, button]
```

→ `.parentNode`는 부모로 올라가는 길

→ `.previousElementSibling`은 앞 형제

→ `.children`은 자식 목록 (단, 요소만!)

📌 2. `.children` vs `.childNodes` – 요소만? 모든 노드?

아래와 같은 리스트가 있다고 해볼게요:

```
<ul id="list">
  <li>Apple</li>
```

```
  <li>Banana</li>
</ul>
```

이제 각각의 자식들을 출력해봅니다:

```
const list = document.getElementById("list");

console.log(list.children);   // [li, li]
console.log(list.childNodes); // [#text, li, #text, li, #text]
```

`.children`은 **요소 노드만 반환하는 HTMLCollection**

`.childNodes`는 **텍스트 노드(#text)까지 포함하는 NodeList**

📚 시각화 예시:

```
ul
 ├── #text        ← 줄바꿈 문자
 ├── li "Apple"    ← 첫 번째 항목
 ├── #text        ← 줄바꿈 문자
 ├── li "Banana"   ← 두 번째 항목
 └── #text        ← 마지막 줄바꿈
```

따라서 반복 작업에는 `.children`이 더 안전하고 깔끔합니다.

📌 3. 형제 요소 탐색 – `.nextElementSibling`, `.previousElementSibling`

```
<ul>
  <li>첫째</li>
  <li class="here">둘째</li>
  <li>셋째</li>
</ul>
```

```
const current = document.querySelector(".here");
```

```
console.log(current.previousElementSibling.textContent); // "첫째"
console.log(current.nextElementSibling.textContent);     // "셋째"
```

→ 현재 요소를 기준으로 좌우의 형제 요소를 쉽게 탐색할 수 있습니다. 이때도 요소(Node가 아닌 Element)에만 적용된다는 점이 중요합니다.

📌 4. 기준점 탐색 vs 전체 탐색 – 무엇이 다를까?

방식	기준	장점	단점
전체 탐색 (e.g. querySelector)	document	간단하고 직관적	문서 전체를 순회하므로 느릴 수 있음
기준점 탐색 (e.g. parentNode)	특정 요소	빠르고 정확	기준 요소가 명확해야 함

예를 들어 버튼을 클릭했을 때, 해당 카드의 이미지만 바꾸고 싶다면 다음처럼 탐색 범위를 줄이는 게 좋습니다:

```
button.addEventListener("click", (e) => {
  const card = e.target.parentNode;
  const image = card.querySelector("img");
  image.src = "new.png";
});
```

→ 이렇게 하면 해당 버튼이 속한 카드에서만 이미지를 바꿀 수 있습니다. 이 방식이 바로 **스코프 내로잉(scope narrowing)**입니다.

📌 5. 성능적 장점 – DOM을 덜 순회하자

- 전체 탐색은 요소가 많을수록 느려짐
- 기준 요소에서 탐색하면 최소한의 DOM만 확인
- 이벤트 버블링과 조합하면 클릭한 요소만 조작 가능
- 컴포넌트 기반 개발에서는 더 필수적인 방식

React, Vue 같은 프레임워크도 내부적으로는 이런 “기준 중심의 DOM 탐색”을 핵심으로 삼고 있습니다. 컴포넌트 내부에서만 DOM을 다루게 하면 성능도 좋고, 유지보수도 쉬워지니까요.

✅ 핵심 요약

- `parentNode`, `children`, `nextElementSibling` 등을 사용하면 현재 요소 기준으로 DOM을 탐색할 수 있다.
- `.children`은 요소만, `.childNodes`는 모든 노드를 포함한다.
- 기준점 탐색은 성능 면에서 효율적이며, 복잡한 UI에서 더욱 유용하다.
- 컴포넌트 기반 개발, 이벤트 처리, 효율적인 DOM 조작을 위해 **기준 탐색 능력은 필수 역량**이다.
