LIKE LION

5강

클라이언트 사이드 자바스크립트

WHATEVER YOU WANT, MAKE IT REAL.

강사 정길용

LIKE LION

{

DOM

Event

BOM

Ajax 프로그래밍

Web APIs

}

LIKE LION

웹 브라우저에서 실행되는 자바스크립트 환경

- ECMAScript: 자바스크립트 언어에 대한 표준
  - https://ecma-international.org/publications-and-standards/standards/ecma-262

- DOM(Document Object Model): 웹페이지 제어를 위한 표준
  - https://dom.spec.whatwg.org
  - window.document 등
  - Event

- BOM(Browser Object Model): 웹페이지 외부의 브라우저 기능 제어를 위한 표준
  - HTML 표준: https://html.spec.whatwg.org
  - window.navigator: 브라우저와 운영체제에 대한 정보 제공
  - window.location: 현재 페이지의 URL에 대한 제어(읽기, 수정)
  - window.history: 브라우저의 과거 페이지 이동 정보에 대한 제어(읽기, 수정)
  - alert, setTimeout 등

- Web APIs: 브라우저가 제공하는 웹 기능을 위한 표준
  - https://spec.whatwg.org
  - XMLHttpRequest: 서버와 통신에 사용되는 객체(Ajax)
  - Web Storage, Notifications API, WebSocket 등

참고: https://ko.javascript.info/browser-environment

LIKE LION

▶ DOM(Document Object Model)

- 1998년 10월 W3C에서 DOM 레벨 1 발표
- 2000년 11월 DOM 레벨 2 발표
- 2004년 04월 DOM 레벨 3 발표
- 2015년 11월 DOM 레벨 4 발표
- 2019년 05월 W3C에서 WHATWG으로 이관
    - https://dom.spec.whatwg.org

- HTML, XML 등의 문서를 제어하기 위한 방법을 정의

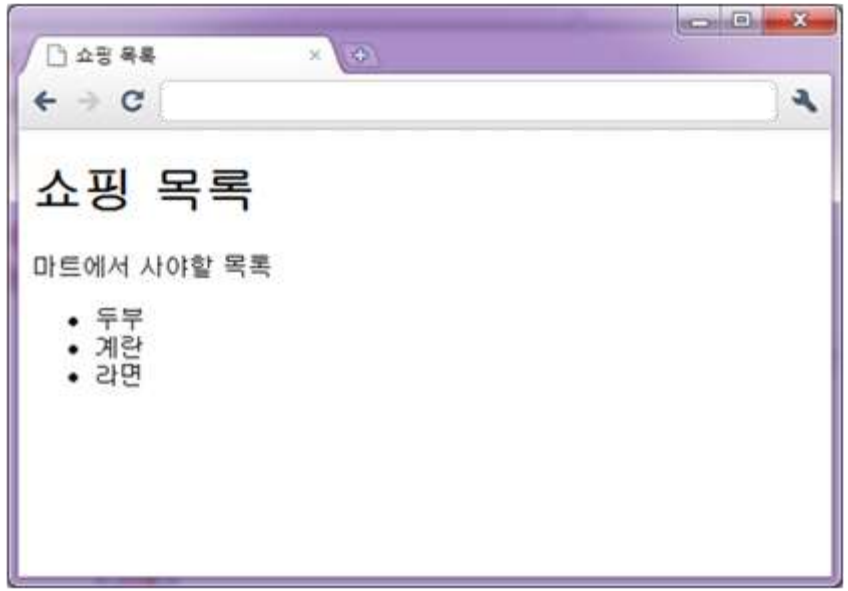
- 텍스트 기반의 HTML, XML 문서를 일정한 규칙에 의해 객체로 만들고 이를 이용하여 문서를 제어(특정 요소를 추출, 삽입, 삭제, 이동 등)

참고: https://ko.javascript.info/browser-environment

LIKE LION

ex05-01.html

```
<html>
<head>
<title>쇼핑목록</title>
</head>
<body>
  <h1>쇼핑 목록</h1>
  <p>마트에서 사야할 목록</p>
  <ul id="purchases" class="list">
    <li>두부</li>
    <li>계란</li>
    <li>라면</li>
  </ul>
</body>
</html>
```



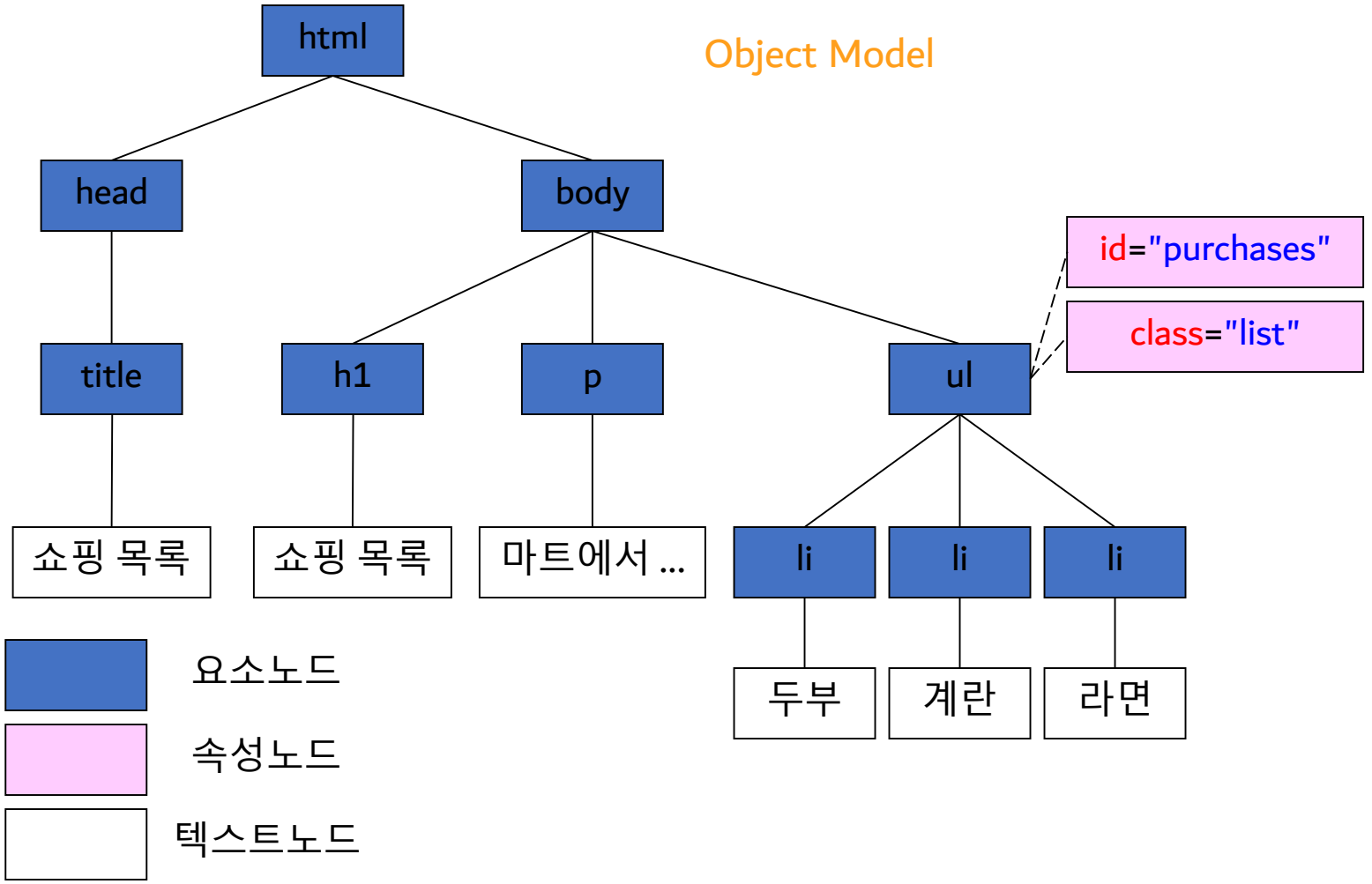
참고: https://ko.javascript.info/dom-nodes

LIKE LION

Document

```
<html>
<head>
<title>쇼핑목록</title>
</head>
<body>
  <h1>쇼핑 목록</h1>
  <p>마트에서 사야할 목록</p>
  <ul id="purchases" class="list">
    <li>두부</li>
    <li>계란</li>
    <li>라면</li>
  </ul>
</body>
</html>
```

Object Model



참고: https://ko.javascript.info/dom-nodes

LIKE LION

▶ 노드(Node)

- DOM 트리구조는 모든 구성원이 각각의 객체로 인식되며 이러한 객체 하나 하나를 노드라고 함

▶ 노드의 종류(주로 사용되는 노드)

- 문서노드(document node)
- 요소노드(element node)
- 속성노드(attribute node)
- 텍스트노드(text node)
- ……

종류	설명	nodeName	nodeType	nodeValue
문서 노드	문서	#document	9	null
요소 노드	태그	태그의 이름	1	null
속성 노드	요소의 속성	속성의 이름	2	속성의 값
텍스트 노드	요소의 내용	#text	3	문자열 값

참고: https://ko.javascript.info/dom-nodes

LIKE LION

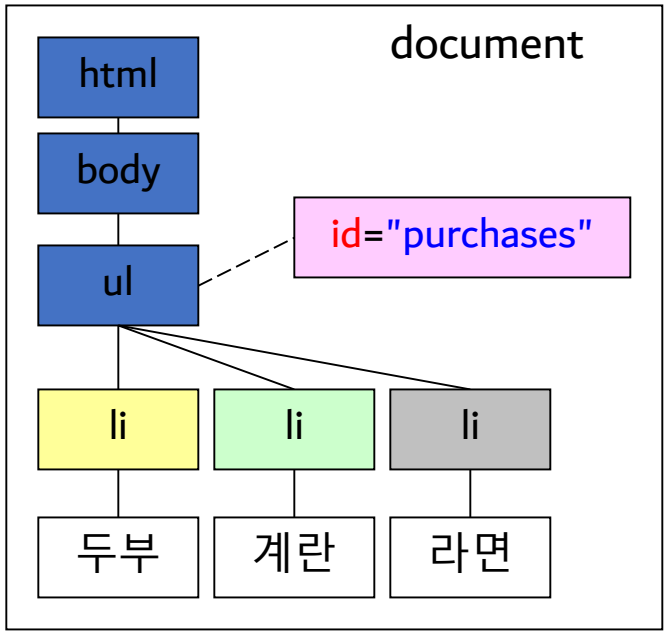
ex05-01.html

▶ 태그의 id를 이용하여 노드 찾기

- document.getElementById(id)
  - id 속성값에 해당하는 노드객체를 반환

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

ul



참고: https://ko.javascript.info/searching-elements-dom

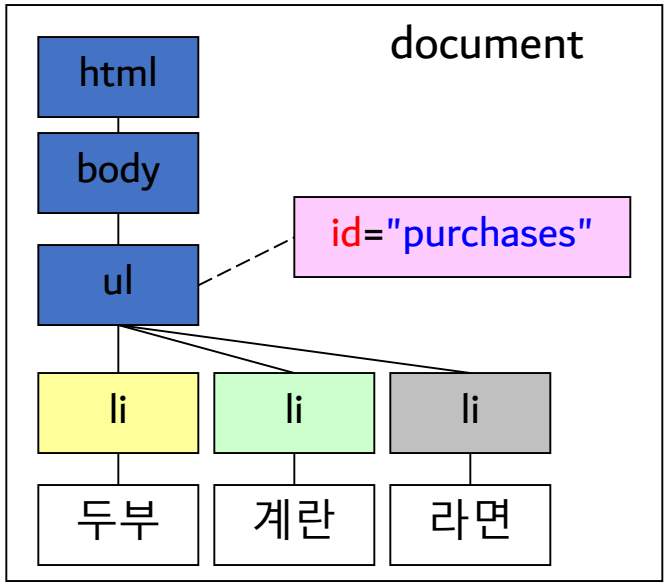
LIKE LION

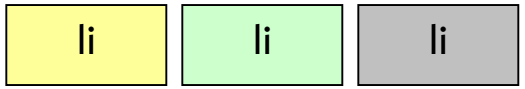
ex05-01.html

▶ 태그명을 이용하여 노드 찾기

- 요소노드.getElementsByTagName(tagName)
  - 지정한 요소노드의 하위 모든 요소를 대상으로 태그명(tagName)에 해당하는 요소노드를 배열로 반환(tagName에 "*"을 지정하면 모든 요소를 배열로 반환)

```
const liList = purchases.getElementsByTagName("li");
```





참고: https://ko.javascript.info/searching-elements-dom

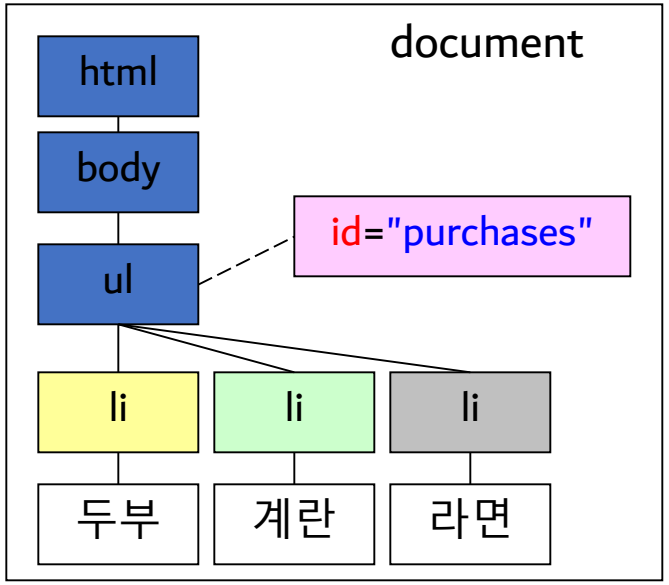
LIKE LION

ex05-01.html

▶ 트리구조를 이용하여 노드 찾기

- 부모/자식 노드 찾기

요소노드의 속성	설명
childNodes	자식 노드가 배열 형태로 저장
firstChild	첫번째 자식 노드(요소, 텍스트, 주석)
firstElementChild	첫번째 자식 요소 노드
lastChild	마지막 자식 노드(요소, 텍스트, 주석)
lastElementChild	마지막 자식 요소 노드
parentNode	부모 노드




```
const purchases = document.getElementById('purchases');
const firstItem = purchases.firstElementChild;
const lastItem = purchases.lastElementChild;
const liList = purchases.childNodes;
```

ul

li

li

li li li

참고: https://ko.javascript.info/dom-navigation

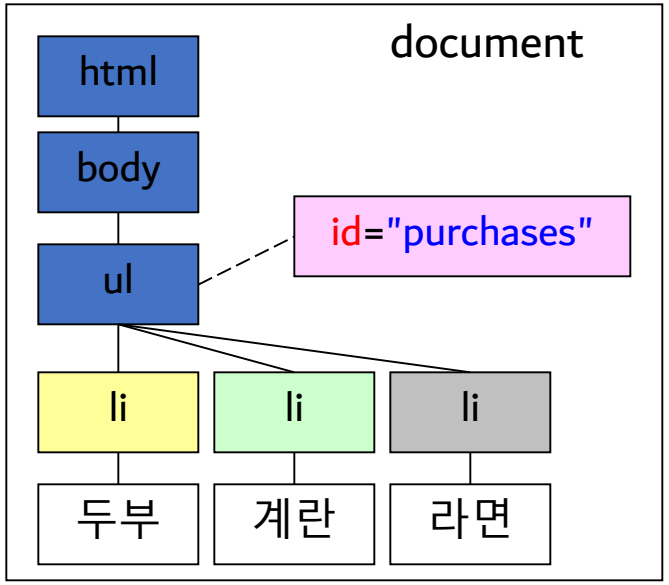
LIKE LION

ex05-01.html

▶ 트리구조를 이용하여 노드 찾기

- 형제 노드 찾기

속성	설명
previousSibling	바로 앞의 형제 노드(요소, 텍스트, 주석)
previousElementSibling	바로 앞의 형제 요소 노드
nextSibling	바로 뒤의 형제 노드(요소, 텍스트, 주석)
nextElementSibling	바로 뒤의 형제 요소 노드




```
const purchases = document.getElementById("purchases");
const secondItem = purchases.childNodes[3];
const firstItem = secondItem.previousElementSibling;
const lastItem = secondItem.nextElementSibling;
```

ul

li

li

li

참고: https://ko.javascript.info/dom-navigation

LIKE LION

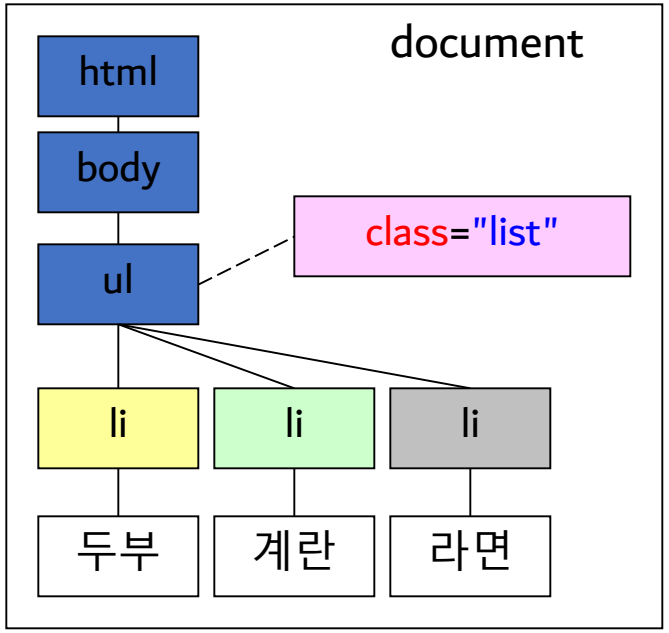
ex05-01.html

▶ class 속성으로 노드 찾기

- document.getElementsByClassName(className)
  - class 속성값이 className인 요소 노드의 목록을 반환

```
const purchases = document.getElementsByClassName("list")[0];
```

ul



참고: https://ko.javascript.info/searching-elements-dom

LIKE LION

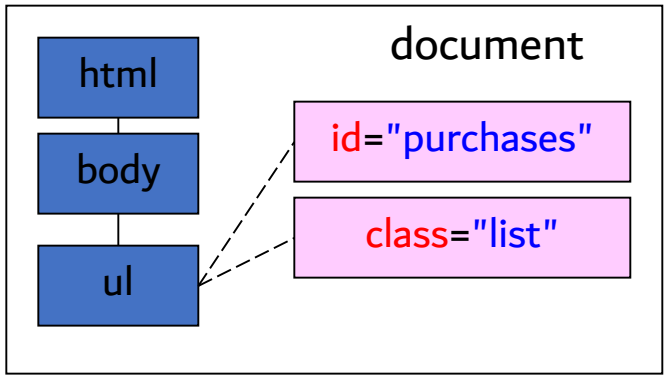
ex05-01.html

▶ CSS 셀렉터 이용

- Selector: CSS에서 사용하는 노드 선택 구문
  - https://www.w3.org/TR/css3-selectors/
- document.querySelector(selector)
  - 지정한 selector 구문에 매칭되는 노드 목록 중 첫번째 노드를 반환
- document.querySelectorAll(selector)
  - 지정한 selector 구문에 매칭되는 노드 목록을 반환

```
var purchases = document.querySelector(".list");
var purchases = document.querySelector("#purchases");
var purchases = document.querySelectorAll("ul")[0];
```

ul



참고: https://ko.javascript.info/searching-elements-dom

LIKE LION

ex05-03.html 

elem.innerHTML

- elem의 내부 HTML 코드의 값을 조회하거나 수정
- elem 자신은 제외

```
const shoppingList =
document.querySelector('#purchases');
console.log(shoppingList.innerHTML);
```

```
'₩n    <li>두부</li>₩n    <li>계란</li>₩n    <li>
라면</li>₩n  '
```

elem.outerHTML

- elem의 내부 HTML 코드의 값을 조회하거나 수정
- elem 자신을 포함

```
const shoppingList =
document.querySelector('#purchases');
console.log(shoppingList.outerHTML);
```

```
'<ul id="purchases" class="list">₩n    <li>두부
</li>₩n    <li>계란</li>₩n    <li>라면</li>₩n
</ul>'
```

```
<ul id="purchases" class="list">
  <li>두부</li>
  <li>계란</li>
  <li>라면</li>
</ul>
```

참고: https://ko.javascript.info/basic-dom-node-properties

LIKE LION

ex05-03.html 

▶ elem.textContent

- elem의 내부 텍스트 노드의 값을 조회하거나 수정
- 소스코드의 값 그대로 조회

```
const secondLi =
document.querySelector('#purchases > li:nth-
child(2)');
console.log(secondLi.textContent); // 계란□
✓□
```

```
<ul id="purchases" class="list">
  <li>두부<span>✓□</span></li>
  <li>계란<span hidden>✓□</span></li>
  <li>라면<span>✓□</span></li>
</ul>
```

▶ elem.innerText

- elem의 내부 텍스트 노드의 값을 조회하거나 수정
- 브라우저에 의해서 실제 보이는 값으로 조회
- 화면에 보이지 않는 요소는 제외(hidden)

```
const secondLi =
document.querySelector('#purchases > li:nth-
child(2)');
console.log(secondLi.innerText); // 계란
```

- 두부 ✔
- 계란
- 라면 ✔

참고: https://ko.javascript.info/basic-dom-node-properties

LIKE LION

ex05-03.html 

▶ 노드 생성

- document 객체의 createXxx() 메소드를 이용

메소드	설명
createElement(nodeName)	지정한 태그명으로 요소노드 생성
createTextNode(nodeValue)	지정한 내용으로 텍스트노드 생성
createAttribute(attributeName)	지정한 이름으로 속성노드 생성

```
const newLiNode = document.createElement("li");

const newTextNode = document.createTextNode("우유");
```

li

우유

참고: https://ko.javascript.info/modifying-document

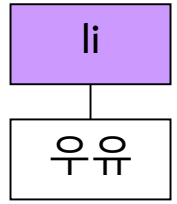
LIKE LION

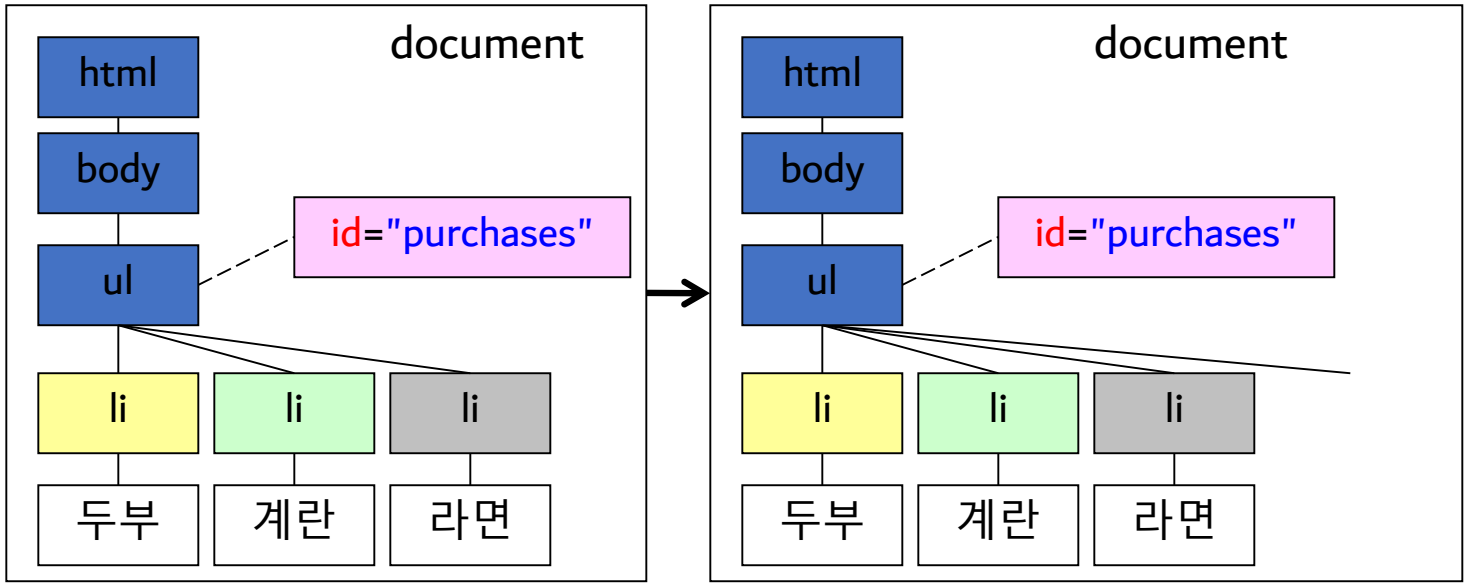
ex05-03.html

▶ 노드 추가

- 요소노드.appendChild(childNode)
  - 지정한 노드를(childNode) 요소노드의 마지막 자식노드로 추가

```
const newLiNode = document.createElement("li");
const newTextNode = document.createTextNode("우유");
newLiNode.appendChild(newTextNode);
purchases.appendChild(newLiNode);
```





참고: https://ko.javascript.info/modifying-document

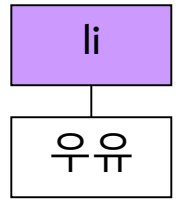
LIKE LION

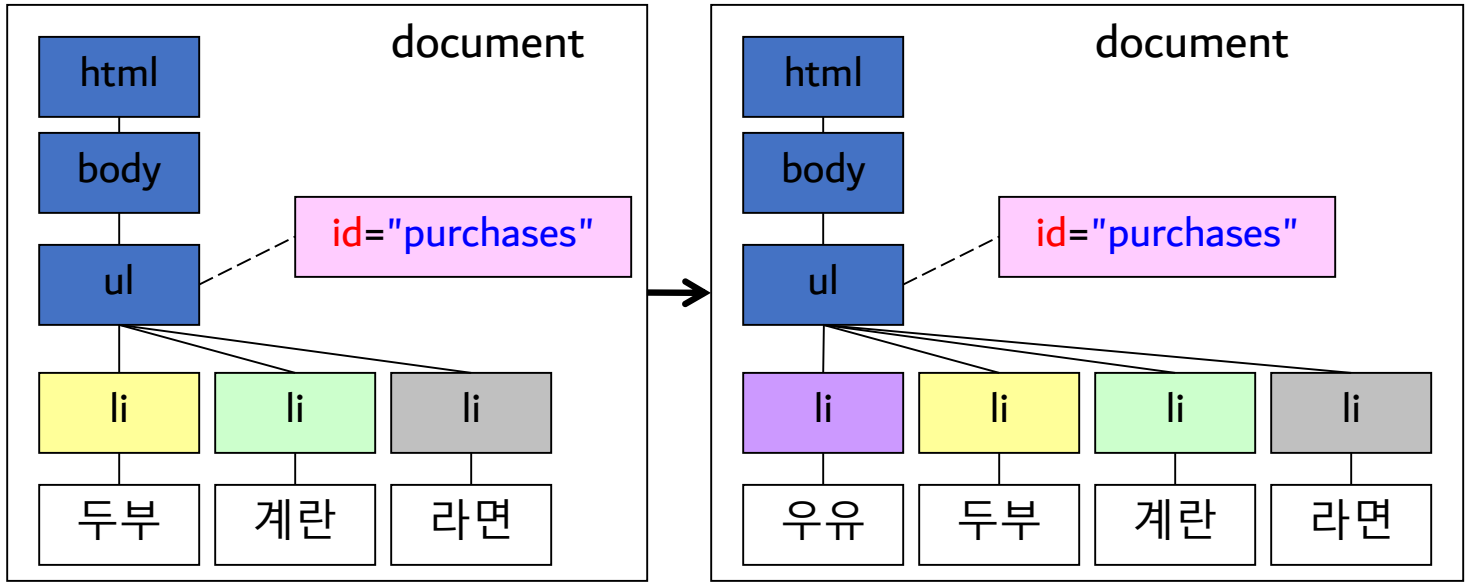
ex05-03.html

▶ 노드 삽입

- 요소노드.insertBefore(newNode, targetNode)
  - 지정한 노드를(newNode) targetNode 앞에 삽입

```
const purchases = document.getElementById("purchases");
purchases.insertBefore(newLiNode, purchases.firstChild);
```





참고: https://ko.javascript.info/modifying-document

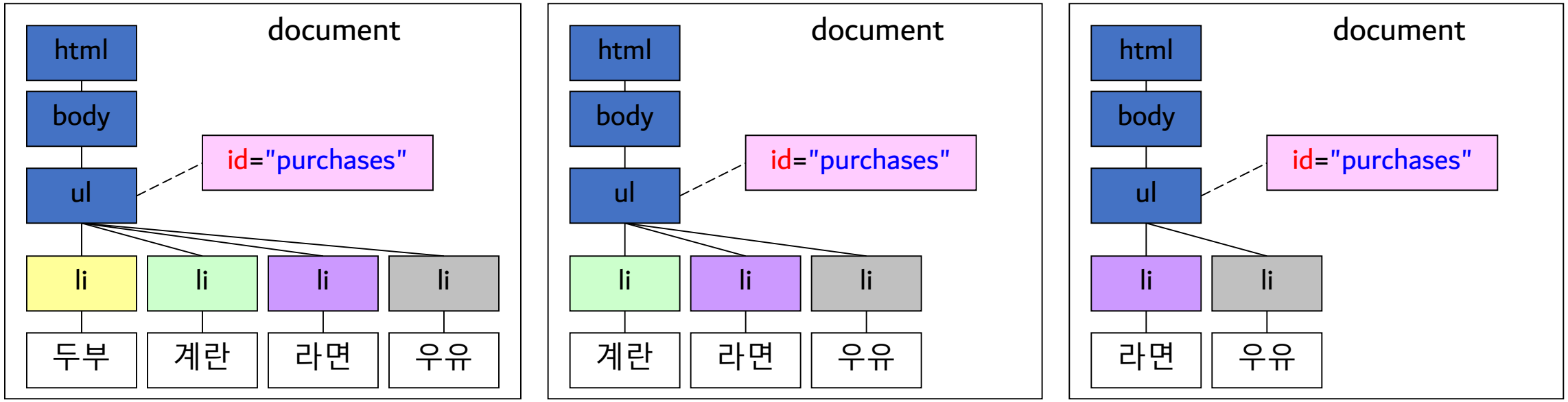
LIKE LION

ex05-03.html

▶ 노드 삭제

- 요소노드.removeChild(childNode)
  - 지정한 자식 노드를(childNode) 삭제한다.
- 요소노드.remove()
  - 자신을 삭제한다.

```
const purchases = document.getElementById("purchases");
purchases.removeChild(purchases.firstElementChild);
purchases.firstElementChild.remove();
```



참고: https://ko.javascript.info/modifying-document

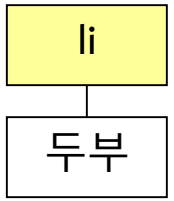
LIKE LION

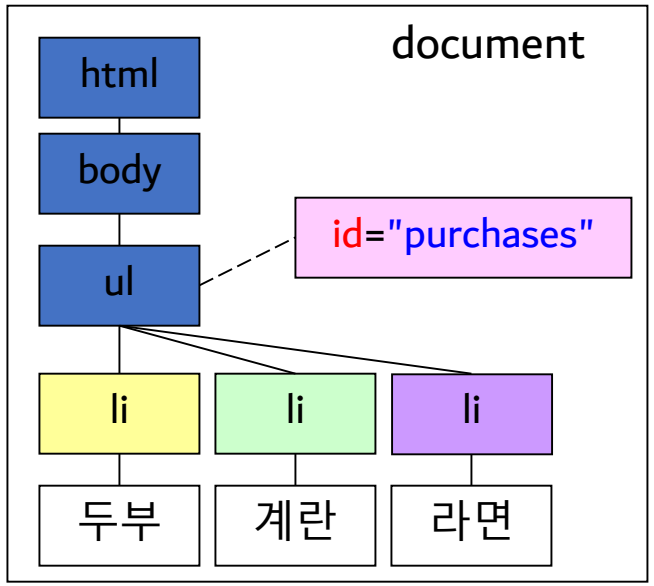
ex05-03.html

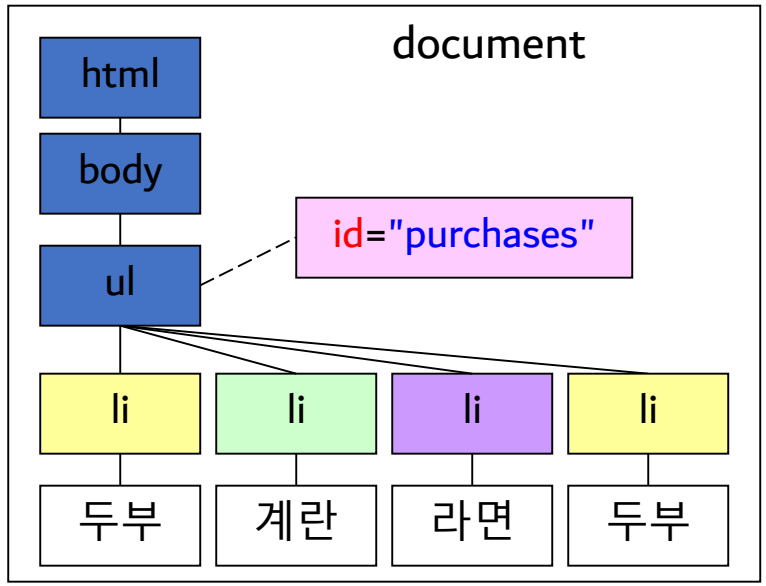
▶ 노드 복사

- 노드.cloneNode(haveChild)
  - 지정한 노드를 복사한다. haveChild가 true이면 하위 모든 노드를 같이 복사하고 false(기본값)이면 지정한 노드만 복사한다.

```
const purchases = document.getElementById("purchases");
const cloneLi = purchases.firstChild.cloneNode(true);
purchases.appendChild(cloneLi);
```







참고: https://ko.javascript.info/modifying-document

LIKE LION

ex05-06.html 

HTML 표준 속성

- HTML 표준 속성은 DOM 객체의 속성으로 저장됨
  - elem.href -> "hello.html"
  - elem.src -> "hello.png"
  - elem.type -> "text"
  - elem.name -> "userName"

```
<a href="hello.html">클릭해봐</a>
<img src="hello.png" format="png">
<input type="text" name="userName">
```

HTML 표준이 아닌 속성

- HTML 표준이 아닌 속성은 DOM 객체의 속성으로 저장되지 않음
  - elem.format -> undefined
- DOM 객체의 속성대신 elem.getAttribute(attrName) 메서드 이용
  - elem.getAttribute('format') -> png

custom 속성

- 개발자가 임의로 부여한 HTML 속성
- data-age, data-user-name 처럼 "data-" 접두어로 시작
- elem.dataset.age, elem.dataset.userName 처럼 dataset 객체의 속성으로 접근 가능
- 속성명을 -로 연결했을 경우 카멜케이스로 변환된 속성명 사용

참고: https://ko.javascript.info/dom-attributes-and-properties

LIKE LION

ex05-07.html 

▶ 요소에 스타일 적용

- 요소의 style 속성에 직접 스타일 지정
  - `<span style="font-size: 40px;">😆</span>`

- 스타일을 명시한 클래스 작성 후 class 속성으로 적용

```
.pad100 {
  padding: 100px;
}
.size30 {
  font-size: 30px;
}
```

```
<ul class="pad100 size30">
  <li>두부</li>
  <li>계란</li>
</ul>
```

- 일반적으로 스타일 정의는 style 속성보다는 class를 사용하는 방식이 선호됨
  - 재사용성: 여러 요소에 적용 가능
  - 성능 향상: 브라우저의 캐싱
  - 유지보수: CSS 파일에서 일괄적인 스타일 관리

- 동적 스타일이 필요한 경우 제한적으로 style 사용

참고: https://ko.javascript.info/styles-and-classes

LIKE LION

ex05-07.html 

자바스크립트로 style 속성 접근

- 요소노드의 style 속성에 CSS 스타일 정보가 객체로 저장됨
- 스타일 속성에 접근할 경우 style.스타일속성명 형태로 사용
  - elem.style.top = "10px"
  - elem.style.left = "20px"

- font-size 같이 -로 연결한 스타일 속성은 카멜케이스로 변환된 속성명 사용
  - elem.style.fontSize = "10px"
  - elem.style.backgroundColor = "yellow"

- 요소노드의 style 속성은 객체이고 읽기 전용 전체 스타일을 한번에 바꾸기 위해서 문자열을 할당할 수 없음
  - elem.style = "font-size: 10px; background-color: yellow;" (x)

- 전체 스타일을 한번에 바꾸기 위해서 cssText 속성 사용
  - elem.style.cssText = "font-size: 10px; background-color: yellow;" 를 사용

참고: https://ko.javascript.info/styles-and-classes

LIKE LION

ex05-08.html 

▶ 자바스크립트로 class 접근

- elem.className 속성에 class 값이 저장되어 있음(class는 예약어라서 className을 대신 사용)
- class 값 전체를 바꿀때는 elem.className = "pad100 size30" 처럼 직접 값을 명시

▶ class 속성을 하나씩 접근

- elem.classList
  - class 속성의 목록을 가지고 있는 유사 배열 객체
  - add("hello"): hello 클래스 추가
  - remove("hello"): hello 클래스 제거
  - toggle("hello"): hello 클래스가 있으면 제거하고 없으면 추가
  - contains("hello"): hello 클래스의 존재 여부 반환

```css
.pad100 {
  padding: 100px;
}
.size30 {
  font-size: 30px;
}
```

```html
<ul class="pad100 size30">
  <li>두부</li>
  <li>계란</li>
</ul>
```

▶ class가 적용된 style 확인(최종 계산된 스타일)

- getComputedStyle(element, [pseudo])
  - 외부 css 파일, 내부 <style>, 인라인 스타일 등 모든 스타일 요소가 반영된 최종 계산된 스타일 반환
  - elem.style과 유사한 스타일 정보가 담긴 객체를 반환하지만 모든 속성은 읽기 전용
  - element: 스타일 값을 읽을 요소노드
  - pseudo: ::before 같은 pseudo-element의 스타일이 필요할 때

참고: https://ko.javascript.info/styles-and-classes

이벤트란?

- 브라우저에서 어떤 일이 일어 났음을 알려주는 신호
- 클릭, 키보드 입력, 마우스 이동, 스크롤 등의 작업
- 주로 요소 노드에서 발생

이벤트 핸들러

- 특정 이벤트가 발생했을 때 실행되는 함수
- 이벤트가 발생하는 대상에 이벤트와 이벤트 핸들러를 등록해서 처리

대표적인 이벤트 종류

- 마우스 이벤트
  - click, dblclick, mousemove, mouseover/mouseout, mousedown/mouseup, contextmenu
- 키보드 이벤트
  - keydown/keyup
- 폼 이벤트
  - focus/blur, input, change, submit
- 스크롤 이벤트
  - scroll
- 문서 로딩 이벤트
  - load, DOMContentLoaded, beforeunload/unload

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

ex05-04.html 

▶ DOM 프로퍼티에 할당

- DOM Level 0 방식 (비표준)
- 요소 노드의 on<event> 속성에 이벤트 핸들러를 등록하면 <event>가 발생했을 때 등록한 핸들러가 호출됨

```
<button>눌러봐</button>
```

```
var btn = document.querySelector('button');
btn.onclick = function(){
  console.log('버튼 클릭');
}
```

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

ex05-04.html 

▶ HTML 인라인 방식

- DOM Level 0 방식 (비표준)
- HTML 태그의 on<event> 속성에 <event>가 발생 했을 때 실행할 코드 지정
  - onclick, onmousemove, onkeydown 등
- 브라우저는 실행할 코드로 구성된 이벤트 핸들러를 만들어서 요소 노드의 on<event> 속성에 등록
- 아래의 두 코드는 동일한 효과

```
<button onclick="console.log('버튼 클릭');">눌러봐</button>
```

```
var btn = document.querySelector('button');
btn.onclick = function(){
  console.log('버튼 클릭');
}
```

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

ex05-04.html 

▶ DOM Level 0 방식의 문제점

- on<event> 속성의 값은 한개만 존재할 수 있기 때문에 이벤트 핸들러를 여러번 할당하면 기존 값이 덮어 씌워져서 이벤트 핸들러를 여러개 등록할 수 없음

```
var btn = document.querySelector('button');
btn.onclick = function(){
  console.log('버튼 클릭1');
}
btn.onclick = function(){
  console.log('버튼 클릭2');
}
```

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

ex05-04.html 

▶ elem.addEventListener(event, handler, [useCapture])

- DOM Level 2 방식 (표준)
- elem 요소노드에 event 발생시 실행할 handler 함수를 등록한다.

- event: 이벤트 이름 (click, mousemove, keydown 등)
- handler: 핸들러 함수
- useCapture: 캡처링 단계의 이벤트 캐치 여부. 기본은 false이고 버블링 단계의 이벤트를 캐치함

```
var btn = document.querySelector('button');
btn.addEventListener('click', function(){
  console.log('버튼 클릭');
});
btn.addEventListener('click', function(){
  console.log('버튼 클릭');
});
```

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

ex05-04.html 

elem.removeEventListener(event, handler, [useCapture])

- elem 요소노드에 event 발생시 실행할 handler 함수를 제거한다.
- 핸들러를 등록할 때 지정했던 매개변수와 동일한 인자값의 핸들러가 삭제됨

```
var btn = document.querySelector('button');
btn.addEventListener('click', function(){
  console.log('버튼 클릭');
});
btn.removeEventListener('click', function(){ // 제거 안됨
  console.log('버튼 클릭');
});
```

```
var btn = document.querySelector('button');
function handleClick(){
  console.log('버튼 클릭');
}
btn.addEventListener('click', handleClick, true);
btn.removeEventListener('click', handleClick, true); // 제거됨
```

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

ex05-07.html 

▶ Event 객체

- 발생한 이벤트의 상세 정보를 담고 있는 객체
- click 이벤트 였다면 마우스의 어떤 버튼이 눌렸는지, keydown 이벤트 였다면 어떤 키가 눌렸는지 같은 이벤트 상세 정보를 확인하고 싶을때 사용
- 이벤트 핸들러 함수의 첫번째 인자값으로 전달됨

```
var btn = document.querySelector('button');
document.addEventListener('mousemove', function(event){
  console.log('마우스 좌표', event.clientX, event.clientY);
});
```

```
마우스 좌표 103 104
마우스 좌표 112 93
마우스 좌표 123 81
```

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

ex05-09.html

▶ Event 객체의 주요 속성과 메서드

- type: 발생한 이벤트 명
- target: 이벤트가 발생한 요소노드
- currentTarget: 이벤트를 처리중인 요소노드
  - <button>의 부모 에 click 이벤트를 등록하고 button을 누르면 내부가 눌렸으므로 이벤트 핸들러가 호출됨. 이때 target은 <button>이 되고 currentTarget은 가 됨

```
<div>
  <button>눌러봐</button>
</div>
```

```
var div = document.querySelector('div');
div.addEventListener('click', function(event){
  console.log(event.target, event.currentTarget);
});
```

- preventDefault(): 다음과 같은 브라우저의 기본 동작을 취소 할때 호출
  - <a> 태그를 누르면 href 주소로 페이지 이동
  - <button type="submit"> 버튼을 누르면 서버로 데이터 전송
- 그밖에 이벤트 종류별로 사용 가능한 속성 제공

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

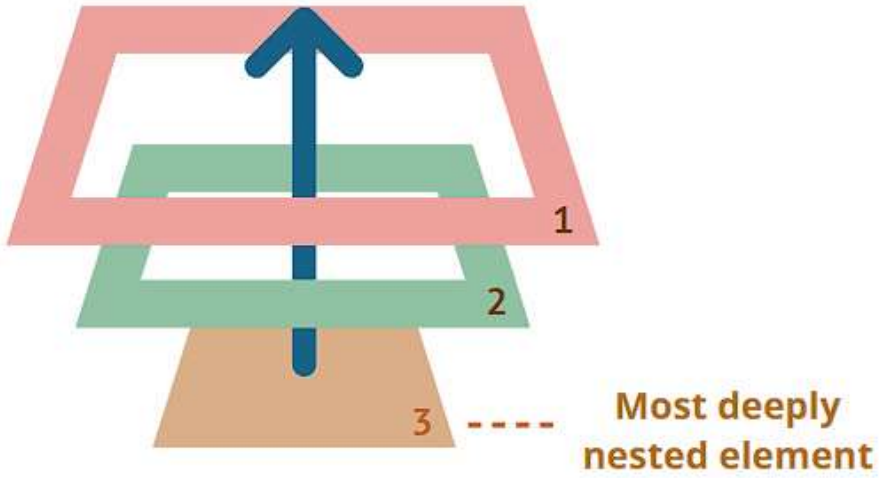
ex05-10.html

▶ 버블링(bubbling)

- 특정 요소에 이벤트가 발생하면 해당 요소의 이벤트 핸들러가 먼저 실행 된 후 부모 요소의 이벤트 핸들러가 연달아 실행됨
  - button의 onclick 핸들러 실행
  - div의 onclick 핸들러 실행
  - body의 onclick 핸들러 실행
  - document 객체를 만날 때까지 각 요소의 onclick 핸들러 실행

- event.stopPropagation() 호출시 이벤트 전파 중단
  - 대부분의 경우 버블링을 중단 시킬 일은 없음

```
<form onclick="alert('form')">FORM
  <div onclick="alert('div')">DIV
    <p onclick="alert('p')">P</p>
  </div>
</form>
```



참고: https://ko.javascript.info/bubbling-and-capturing

LIKE LION

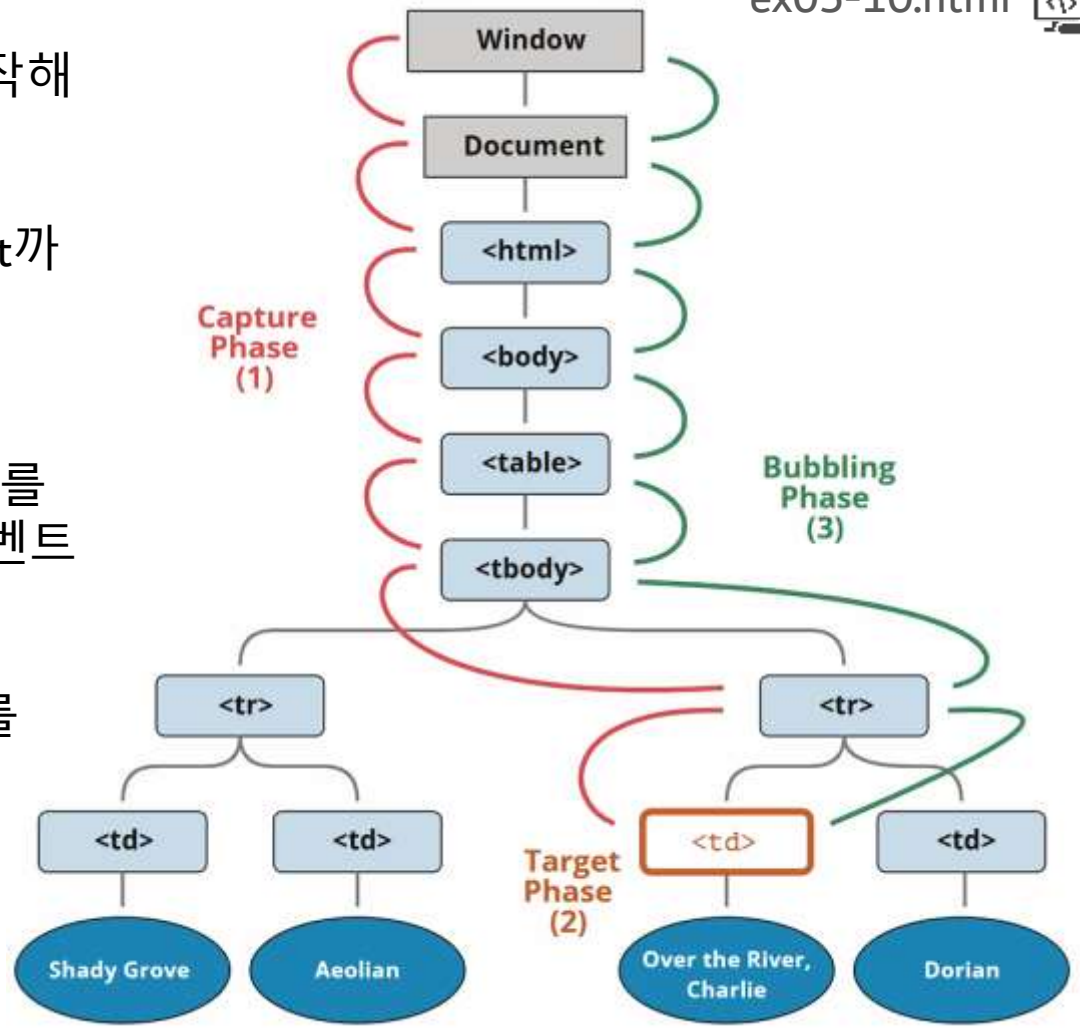
ex05-10.html

이벤트 전파 단계

- 캡처링(capturing) 단계 : 이벤트가 documen에서 시작해서 타겟 요소까지 하위 요소로 전파되는 단계
- 타깃 단계: 이벤트가 타겟 요소에 도달
- 버블링 단계:이벤트가 다시 타겟 요소에서 document까지 상위 요소로 전파되는 단계

캡처링

- addEventListener()의 세번째 매개변수인 useCapture를 생략하면 기본값은 false이고 이는 버블링 단계의 이벤트를 캐치
- 캡처링 단계의 이벤트를 캐치하기 위해서는 addEventListener()의 세번째 매개변수인 useCapture를 true로 지정해야 하지만 사용할 일은 거의 없음



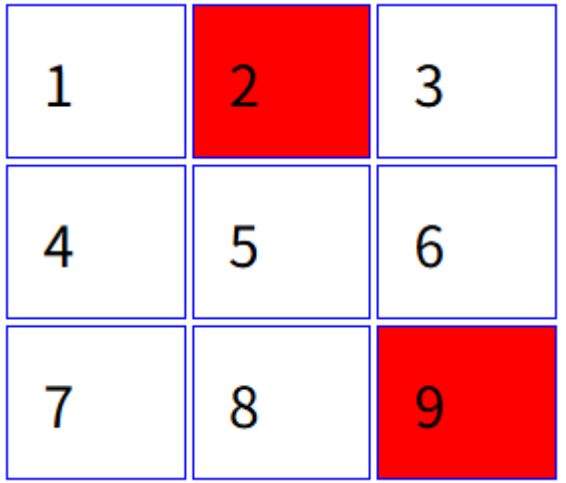
참고: https://ko.javascript.info/bubbling-and-capturing

LIKE LION

ex05-11.html

▶ 이벤트 위임이란?

- 이벤트 발생시 비슷한 처리를 해야하는 요소들이 여럿 있을 경우 각 요소에 하나씩 이벤트 핸들러를 할당하지 않고 공통의 부모 요소에 이벤트 핸들러를 하나만 할당해서 처리하는 방식
- 자식의 이벤트가 부모에게 전파되는 이벤트 버블링을 활용
- event.target 속성으로 실제 이벤트가 발생한 요소를 확인 가능
- 동적으로 추가된 자식 요소에 따로 이벤트를 추가할 필요 없음




```html
<table>
  <tr>
    <td>1</td><td>2</td><td>3</td>
  </tr>
  <tr>
    <td>4</td><td>5</td><td>6</td>
  </tr>
  <tr>
    <td>7</td><td>8</td><td>9</td>
  </tr>
</table>
```

```js
const table = document.querySelector('table');
table.addEventListener('click', function(event){
  if(event.target.tagName === 'TD'){
    event.target.style.backgroundColor = 'red';
  }
});
```

참고: https://ko.javascript.info/event-delegation