getElementsByClassName

`getElementsByClassName()` is a method which exists on HTML `Document`s and `Element`s to return an `HTMLCollection` of descendant elements within the `Document` / `Element` which has the specified class name(s).

Let's implement our own `Element.getElementsByClassName()` that is similar but slightly different:

- It is a pure function which takes in an element and a `classNames` string, a string containing one or more `getElementsByClassName(document.body, 'foo bar')`
- Similar to `Element.getElementsByClassName()`, only descendants of the element argument are searched, not the element itself.
- Return an array of `Element`s, instead of an `HTMLCollection` of `Element`s.

Do not use `document.querySelectorAll()` which will make the problem trivial otherwise. You will not be allowed to use it during real interviews.

Examples

```
const doc = new DOMParser().parseFromString(
  `<div class="foo bar baz">
    <span class="bar baz">Span</span>
    <p class="foo baz">Paragraph</p>
    <div class="foo bar"></div>
  </div>`,
  'text/html',
);

getElementsByClassName(doc.body, 'foo bar');
// [div.foo.bar.baz, div.foo.bar] <-- This is an array of elements.
```

Solution

The solution is pretty straightforward if you are familiar with the HTML DOM APIs. In particular, we need to know the following:

- `Element.classList` which returns a live `DOMTokenList` of class attributes of the element. This is

preferred over `className` because `className` is a string and needs to be manually parsed.

- `Element.children` which returns a live `HTMLCollection` of the child elements. We use this over `Node.childNodes` which returns a live `NodeList` of child `Node`s because `childNodes` will include non-element nodes like text and comment nodes, which are not relevant in this question.
  - However `HTMLCollection` does not have `.forEach`, so we have to iterate through it using traditional `for` loops.

For the class name to match, they have to be a subset of the `classList` of an element. The matching is also case-sensitive and duplicate class names (in both the input and on the elements) do not matter.

We can maintain an `elements` array to collect the matching elements while recursively traversing the root element. A depth-first traversal is performed.

Remember that the element argument itself is not included in the results.

JavaScript TypeScript

```
function isSubset(a, b) {
  return Array.from(a).every((value) => b.contains(value));
}

/**
 * @param {Element} element
 * @param {string} classNames
 * @return {Array<Element>}
 */
export default function getElementsByClassName(element, classNames) {
  const elements = [];
  const classNamesSet = new Set(classNames.trim().split(/\s+/));

  function traverse(el) {
    if (el == null) {
      return;
    }

    if (isSubset(classNamesSet, el.classList)) {
      elements.push(el);
    }

    for (const child of el.children) {
      traverse(child);
```

```
    }
  }

  for (const child of element.children) {
    traverse(child);
  }

  return elements;
}
```

Edge cases

- Element argument is not included in the results even if it matches the tag name.
- Duplicate class names in both the input and on the element's `class` are ignored.
- Whitespace in input and on the element's `class` are handled properly.

Techniques

- Recursion
- DOM APIs
  - How to get an `Element`'s class
  - How to traverse an `Element`'s children
- CSS: class matching algorithm