Event Bubbling

```
<!DOCTYPE html>
<html>
<head>
    <title>DOM Manipulation</title>
    <style>
        body * {
            margin: 10px;
            border: 2px solid red
        }

        #outerDiv
        {
            background-color: yellow;
            height: 500px;
        }

        #innerDiv
        {
            background-color: yellowgreen;
            height: 400px;
        }

        h3
        {
            background-color: green;
            height: 300px;
            font-size: 150px;
            text-align: center;
        }
    </style>
</head>
<body>
    <div id="outerDiv">
        <div id="innerDiv">
            <h3>Inner Heading</h3>
        </div>
    </div>
    <script src="dom.js"> </script>
</body>
</html>
```

```
document.querySelector('h3').addEventListener('click', () =>
console.log('Heading 3 Clicked'));

document.getElementById('innerDiv').addEventListener('click', () =>
console.log('Inner Div Clicked'));

document.getElementsByTagName('div')[0].addEventListener('click', () =>
console.log('Outer div Clicked'));

document.body.addEventListener('click', () => console.log('Body Clicked')
);
```

Outputs:

1. h3 is clicked
   Heading 3 Clicked
   Inner Div Clicked
   Outer div Clicked
   Body Clicked

2. #innerDiv is clicked
   Inner Div Clicked
   Outer div Clicked
   Body Clicked

3. #outerDiv is clicked
   Outer div Clicked
   Body Clicked

Event bubbling is the default behavior in which an event is first triggered on the target element that originally caused the event (e.g., a button click) and then "bubbles up" through its ancestor elements. This means that the event handlers of the ancestor elements are also invoked in sequence, from the innermost element to the outermost element.

Event Capturing or Trickling

Event capturing is the opposite phase of event propagation. In this phase, the event starts at the root of the DOM tree (usually the window object) and travels through the DOM tree down to the target element. This is less common and is mainly used in advanced scenarios.

```
document.querySelector('h3').addEventListener('click', () =>
console.log('Heading 3 Clicked'), true);

document.getElementById('innerDiv').addEventListener('click', () =>
console.log('Inner Div Clicked'), true);

document.getElementsByTagName('div')[0].addEventListener('click', () =>
console.log('Outer div Clicked'), true);

document.body.addEventListener('click', () => console.log('Body Clicked'),
true);
```

Outputs:

1. h3 is clicked
   Body Clicked
   Outer div Clicked
   Inner Div Clicked
   Heading 3 Clicked

2. #innerDiv is clicked
   Body Clicked
   Outer div Clicked
   Inner Div Clicked

3. #outerDiv is clicked
   Body Clicked
   Outer div Clicked

```
document.querySelector('h3').addEventListener('click', () =>
console.log('Heading 3 Clicked'), true);

document.getElementById('innerDiv').addEventListener('click', () =>
console.log('Inner Div Clicked'), false);

document.getElementsByTagName('div')[0].addEventListener('click', () =>
console.log('Outer div Clicked'), true);

document.body.addEventListener('click', () => console.log('Body Clicked'),
true);
```

Output:

```
h3 is clicked

Capturing Phase
Body Clicked
Outer div Clicked
Heading 3 Clicked

Bubbling Phase
Inner Div Clicked
```

```
document.querySelector('h3').addEventListener('click', () =>
console.log('Heading 3 Clicked'), false);

document.getElementById('innerDiv').addEventListener('click', () =>
console.log('Inner Div Clicked'), false);

document.getElementsByTagName('div')[0].addEventListener('click', () =>
console.log('Outer div Clicked'), true);

document.body.addEventListener('click', () => console.log('Body Clicked'),
true);
```

Output:

```
h3 is clicked

Capturing Phase
Body Clicked
Outer div Clicked

Bubbling Phase
Heading 3 Clicked
Inner Div Clicked
```

Stop Propagation

```
document.querySelector('h3').addEventListener('click', () =>
console.log('Heading 3 Clicked'), false);

document.getElementById('innerDiv').addEventListener('click', (e) =>
{
  e.stopPropagation();
  console.log('Inner Div Clicked');
}, false);

document.getElementsByTagName('div')[0].addEventListener('click', () =>
console.log('Outer div Clicked'), false);

document.body.addEventListener('click', () => console.log('Body Clicked'),
false);
```

Outputs:

1. h3 is clicked
   Heading 3 Clicked
   Inner Div Clicked

2. #innerDiv is clicked
   Inner Div Clicked

3. #outerDiv is clicked
   Outer div Clicked
   Body Clicked

```
document.querySelector('h3').addEventListener('click', (e) =>
{
  e.stopPropagation();
  console.log('Heading 3 Clicked');
}, false);

document.getElementById('innerDiv').addEventListener('click', () =>
console.log('Inner Div Clicked'), false);

document.getElementsByTagName('div')[0].addEventListener('click', () =>
console.log('Outer div Clicked'), false);

document.body.addEventListener('click', () => console.log('Body Clicked'),
false);
```

Outputs:

1. h3 is clicked
   Heading 3 Clicked

2. #innerDiv is clicked
   Inner Div Clicked
   Outer div Clicked
   Body Clicked

3. #outerDiv is clicked
   Outer div Clicked
   Body Clicked

```
document.querySelector('h3').addEventListener('click', (e) =>
{
  e.stopPropagation();
  console.log('Heading 3 Clicked');
}, true);

document.getElementById('innerDiv').addEventListener('click', () =>
console.log('Inner Div Clicked'), true);

document.getElementsByTagName('div')[0].addEventListener('click', () =>
console.log('Outer div Clicked'), true);

document.body.addEventListener('click', () => console.log('Body Clicked'),
true);
```

Outputs:

1. h3 is clicked
   Body Clicked
   Outer div Clicked
   Inner Div Clicked
   Heading 3 Clicked

2. #innerDiv is clicked
   Body Clicked
   Outer div Clicked
   Inner Div Clicked

3. #outerDiv is clicked
   Body Clicked
   Outer div Clicked

```
document.querySelector('h3').addEventListener('click', () =>
console.log('Heading 3 Clicked'), true);

document.getElementById('innerDiv').addEventListener('click', () =>
console.log('Inner Div Clicked'), true);

document.getElementsByTagName('div')[0].addEventListener('click', (e) =>
{
  e.stopPropagation();
  console.log('Outer div Clicked')
}, true);

document.body.addEventListener('click', () => console.log('Body Clicked'),
true);
```

Outputs:

1. h3 is clicked
   Body Clicked
   Outer div Clicked

2. #innerDiv is clicked
   Body Clicked
   Outer div Clicked

3. #outerDiv is clicked
   Body Clicked
   Outer div Clicked

Event Delegation

```
<!DOCTYPE html>
<html>
<head>
    <title>DOM Manipulation</title>
    <style>
       div
       {
            height: 500px;
            border: 1px solid red;
            background-color: lightpink;
       }

       ul
       {
            list-style: none;
            display: flex;
            flex-direction: row;
            justify-content: space-around;
       }

       li
       {
            height: 200px;
            width: 200px;
            background-color: #fff;
            font-size: xx-large;
       }
    </style>
</head>
<body>
    <div>
        <ul id="category">
            <li id="laptop">Laptop</li>
            <li id="mobile">Mobile</li>
            <li id="camera">Camera</li>
        </ul>
    </div>
    <script src="dom.js"> </script>
</body>
</html>
```

```
const list = document.getElementById("category");
list.addEventListener("click", function(event) {
  if (event.target.tagName === "LI") {
    console.log("Clicked on list item:", event.target.textContent);
  }
});
```

Event delegation is a technique in JavaScript where you attach a single event listener to a parent element in the DOM, rather than attaching individual event listeners to multiple child elements. This allows you to handle events for multiple elements using a single event handler function. Event delegation is especially useful when you have a large number of elements that need to share the same behavior or when elements are dynamically added or removed from the DOM.

The basic idea of event delegation is to take advantage of event bubbling. When an event occurs on a child element, it "bubbles up" through its ancestor elements, including the parent element. By listening for the event on a parent element, you can catch the event as it bubbles up and then determine which specific child element triggered the event using the event.target property.

Benefits of Event Delegation:

1. Efficiency: With event delegation, you attach fewer event listeners, which can improve performance, especially when dealing with a large number of elements.

2. Dynamic Elements: If new elements are added to or removed from the parent element dynamically, event delegation automatically handles events for these elements without needing to add new listeners.

3. Simplicity: It simplifies event management and reduces the need to attach and remove listeners when elements change.

4. Memory Management: Since you have fewer event listeners, you reduce the risk of memory leaks.

5. Event delegation is a powerful technique that helps you write more efficient and maintainable code, especially in scenarios where you have many similar elements that require the same behavior.

Limitations:

1) All the events are not bubbled up, some events like blur, focus are not bubbled up.
2) If e.stopPropogation is used in child, then events are not bubbled up.