Class 21 - Discussion of Event propagation and related concepts of bubbling and capturing.

Agenda

What is Event Propagation?
Concept of Bubbling
Concept of capturing
Machine coding question
Star Rating Component
Counter Component

Event Propagation

1. In web development, events in the DOM can move from the outermost elements towards the target element (capturing) and then back out towards the outer elements (bubbling), enabling interactions at multiple levels of the DOM tree.
2. Event propagation refers to the process of how events are dispatched and processed in the Document Object Model (DOM) hierarchy of elements in web development.
3. There are two main phases of event propagation: capturing phase and bubbling phase.

4. During these phases, events can propagate through the DOM tree from the root to the target element (capturing) or from the target element back up to the root (bubbling).

Bubbling

1. Event bubbling is one of the phases of event propagation in the DOM. When an event occurs on a DOM element, it first triggers the event handlers on the target element itself, then it bubbles up to its parent elements in the DOM hierarchy.
2. This allows for the creation of more general event listeners that can be applied to parent elements to handle events from multiple child elements.

Capturing

1. Event capturing is the other phase of event propagation in the DOM. During the capturing phase, events start from the root element and propagate down to the target element.

This phase occurs before the bubbling phase and can be useful when you want to capture events on parent elements before they reach their child elements.

title: Event Bubbling

description: Discussion of Event propagation and related concepts of bubbling and capturing.

1. let's use an example to understand event bubbling and capturing with three nested elements: #grandparent, #parent, and #child. Suppose you have the following HTML structure:
2. code

```
<body>
   <div id="grandparent">
     <div id="parent">
       <div id="child"></div>
     </div>
   </div>
 </body>
```

3. Add some style

```
<style>
       #grandparent {
           background-color: red;
           height: 100px;
           width: 100px;
       }
       #parent {
           background-color: green;
           height: 75px;
           width: 75px;
       }
       #child {
           background-color: blue;
           height: 50px;
           width: 50px;
       }
   </style>
```

4. Now, let's say we attach a click event listener to each of these elements and observe how events work
5. code

```
const grandparent = document.querySelector('#grandparent');
const parent = document.querySelector('#parent');
const child = document.querySelector('#child');

grandparent.addEventListener('click', function() {
 console.log('Grandparent clicked');
});

parent.addEventListener('click', function() {
 console.log('Parent clicked');
});

child.addEventListener('click', function() {
 console.log('Child clicked');
});
```

6. Observe the output

Concept of Event Bubbling

1. Event bubbling is a type of event propagation in the DOM where an event not only triggers the event handlers on the target element (the element on which the event occurred) but also triggers event handlers on its ancestor elements in the hierarchy, bubbling up from the target element to the root of the document.
2. In this example, if you click on the child div, here's what happens due to event bubbling:

a. The click event is first dispatched to the child div, and any event listeners attached to the child are invoked. In this case, "Child clicked" is logged to the console.
b. The event then bubbles up to the parent div, triggering any event listeners attached to the parent. Consequently, "Parent clicked" is logged.
c. The event continues to bubble up to the grandparent div, and "Grandparent clicked" is logged as the event listeners on the grandparent are also triggered.
d. If there were more ancestor elements (up to the document object), the event would continue to bubble up, potentially triggering their event listeners as well.
e. Output
   Child clicked
   Parent clicked
   Grandparent clicked
f. This example demonstrates the sequence of event propagation during the bubbling phase in the context of nested elements.

title: Event Capturing

description: Discussion of Event propagation and related concepts of bubbling and capturing.

1. Update the listener to see the capture phase in action

```
// grandparent.addEventListener('click', function() {
//   console.log('Grandparent clicked');
// });

// parent.addEventListener('click', function() {
//   console.log('Parent clicked');
// });

// child.addEventListener('click', function() {
//   console.log('Child clicked');
// });

grandparent.addEventListener('click', function() {
   console.log('Grandparent clicked (Capturing)');
 }, true);
  parent.addEventListener('click', function() {
   console.log('Parent clicked (Capturing)');
 }, true);
  child.addEventListener('click', function() {
   console.log('Child clicked (Capturing)');
 }, true);
```

2. Let's break down the code and see how capturing works:
3. Adding Event Listeners
    a. The code then adds a click event listener to each of these elements. The addEventListener method is used to achieve this, and it takes three arguments:
    b. The type of event to listen for ("click" in this case).

4. The function to call when the event occurs. Here, it logs a message indicating which element was clicked and notes that it's capturing the event.
5. A boolean value that specifies whether to use capturing. Setting this to true opts into the capturing phase.

How Capturing Works

1. Step by Step Through a Click on child
2. If the child element is clicked:
    a. The capturing phase is initiated from the document down towards the child. First, the grandparent's click event listener is triggered, logging "Grandparent clicked (Capturing)".
    b. The event continues to propagate to the parent, triggering its event listener and logging "Parent clicked (Capturing)".
    c. Finally, the event reaches the child, its event listener is triggered, and it logs "Child clicked (Capturing)".
3. Since all the event listeners were set up for capturing and there are no listeners for the bubbling phase in this code, the event propagation ends here.
4. **Output-**

```
Grandparent clicked (Capturing)
Parent clicked (Capturing)
Child clicked (Capturing)
```

Quick Summary of both the Scenarios

1. Scenario 1: useCapture set to false (Bubbling)
    a. When you click on the #child element, the event will propagate in the bubbling phase. The order of event handling will be:
        i. from the innermost Element to the outermost Element , just like how a bubble flows from bottom to top
    b. Output
    #child clicked (Bubbling)
    #parent clicked (Bubbling)
    #grandparent clicked (Bubbling)
    The output in the console will be:

    Child clicked (Bubbling)
    Parent clicked (Bubbling)
    Grandparent clicked (Bubbling)
2. Scenario 2: useCapture set to true (Capturing)
    a. When you click on the #child element, the event will propagate in the capturing phase. The order of event handling will be:
    b. order
    #grandparent clicked (Capturing)
    #parent clicked (Capturing)
    #child clicked (Capturing)

The output in the console will be:

Grandparent clicked (Capturing)
Parent clicked (Capturing)
Child clicked (Capturing)

3. In both scenarios, the event propagation follows the sequence based on the capturing or bubbling phase, as determined by the useCapture parameter.

title: Event Propagation cycle

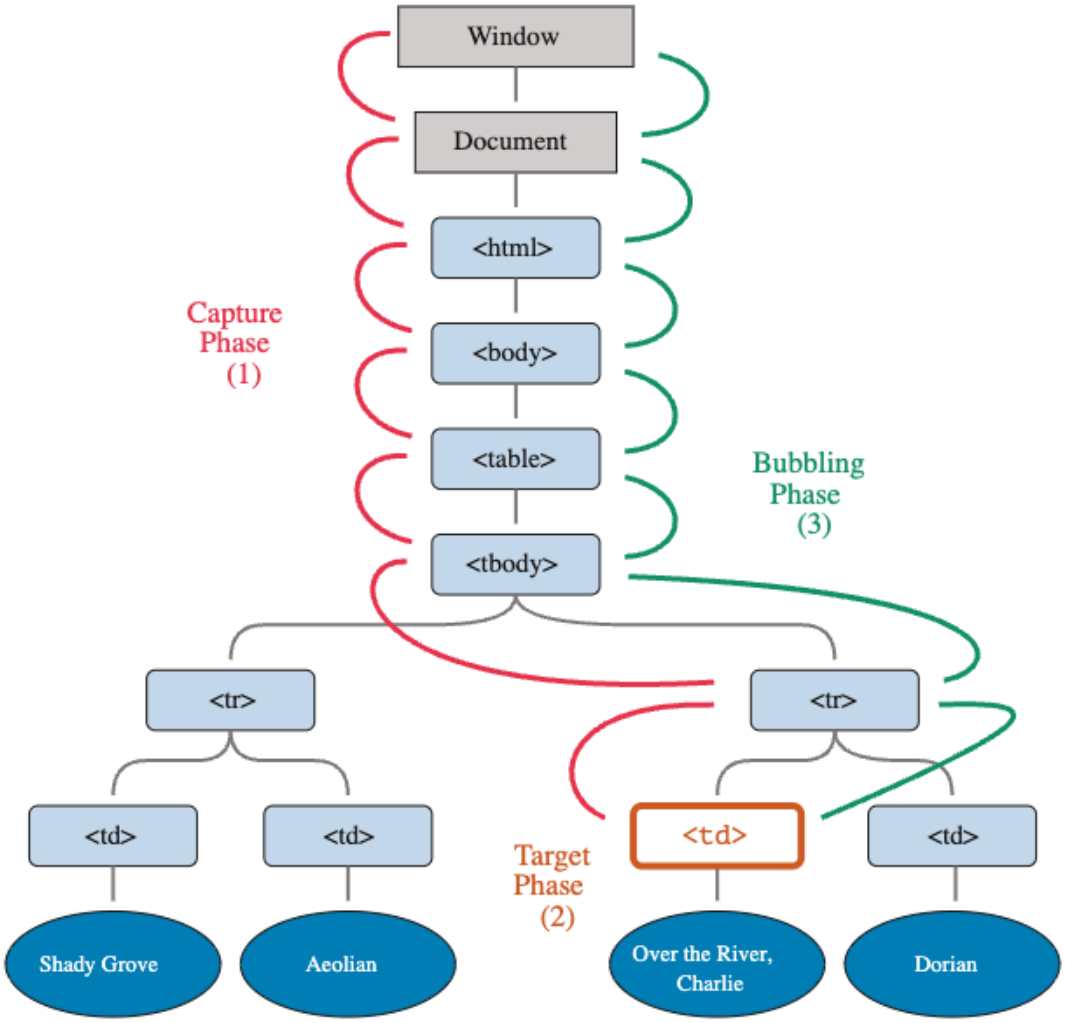
1. The event propagation cycle refers to the sequence of phases through which an event travels within the Document Object Model (DOM) hierarchy. There are two main phases in the event propagation cycle and the target Element as we have discussed
2. the capturing phase and the bubbling phase. Here's an overview of the cycle:
    a. Capturing Phase:
        i. The event starts at the root of the DOM tree (typically the <html> element).
        ii. The event travels down the DOM tree through each ancestor element of the target element.\
        iii. During this phase, event handlers registered with the capturing phase (useCapture set to true) are triggered

on each ancestor element in the order they appear in the hierarchy from the root to the target.

iv. The event reaches the target element.

b. Target Phase:
    i. The event reaches the target element for which the event was triggered.
    ii. Event handlers registered on the target element are executed.

c. Bubbling Phase:
    i. After the target phase, the event travels back up the DOM tree in reverse order.
    ii. Event handlers registered with the bubbling phase (useCapture set to false) are triggered on each ancestor element in the reverse order from the target to the root.
    iii. The event eventually reaches the root of the DOM tree.



3.

title: Stop Propagation

1. Now Sometimes you would want to stop the propagation of the event , Suppose you only want the event to propagate upto a certain element and not after that

2. let's modify the example to demonstrate how to stop event propagation after the click event on the #child element using the stopPropagation method.
3. Comment the capture phase

```
grandparent.addEventListener("click", function () {
 console.log("Grandparent clicked");
});

parent.addEventListener("click", function () {
 console.log("Parent clicked");
});

child.addEventListener("click", function (e) {
 console.log("Child clicked");
 e.stopPropagation(); // Stop propagation after clicking the
child element
});
```

4. In this example, when you click on the #child element, the event propagation will be stopped after the event handler for the #child element executes.
5. As a result, the event will not continue to bubble up to the parent and grandparent elements. Only the message "Child clicked (Bubbling)" will be logged to the console.
6. If you remove the line e.stopPropagation();, you'll see the standard bubbling behavior where the event continues to propagate, and you'll see all three messages in the console: "Child clicked (Bubbling)", "Parent clicked (Bubbling)", and "Grandparent clicked (Bubbling)".

7. Remember that stopping propagation can impact the expected behavior of event handling, so it should be used with caution and only when necessary.
8. You can play around now with setting true false for the useCapturing , stopping propagations and observing the behaviours

title: Machine Coding Questions

1. Machine coding questions are a type of interview question commonly used in the hiring process for software engineers.
2. These questions are practical, hands-on coding tasks that require candidates to design and implement a piece of software solution to a specific problem, usually within a limited time frame
3. We will be solving two Machine coding Problems Today , They are mentioned below with theri problem statements
    a. Star Rating Component:
        i. Design a star rating component that allows users to rate something using stars.
        ii. The component should display a visual representation of the rating using filled and empty stars. Users can click on the stars to select a rating.
    b. Counter Component:

i. Create a counter component that displays a number and has buttons to increment and decrement the number.
ii. The user should be able to click the buttons to increase or decrease the displayed number.

Creating a counter

1. Firstly we will be discussing the problem statement of Creating a counter that displays a number and has buttons to increment and decrement the number. The user should be able to click the buttons to increase or decrease the displayed number.
2. html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width,
initial-scale=1.0">
 <link rel="stylesheet" href="styles.css">
 <title>Counter App</title>
</head>
<body>
 <div class="counter">
   <button class="btn" id="decrement">-</button>
   <span id="count">0</span>
   <button class="btn" id="increment">+</button>
   <button class="btn" id="reset">Reset</button>
 </div>
 <script src="script.js"></script>
```

```
</body>
</html>
```

2. css

```
<style>
    body {
      font-family: Arial, sans-serif;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
    }

    .counter {
      display: flex;
      align-items: center;
    }

    .btn {
      padding: 10px 15px;
      font-size: 18px;
      background-color: #3498db;
      color: #fff;
      border: none;
      cursor: pointer;
      transition: background-color 0.3s ease;
      margin:1rem;

    }

    .btn:hover {
      background-color: #2980b9;
    }
```

```
</style>
```

3. JS

```
const decrementButton = document.getElementById("decrement");
const incrementButton = document.getElementById("increment");
const resetButton = document.getElementById("reset");
const countDisplay = document.getElementById("count");

let count = 0;

decrementButton.addEventListener("click", () => {
 if (count > 0) {
   count--;
   countDisplay.textContent = count;
 }
});

incrementButton.addEventListener("click", () => {
 count++;
 countDisplay.textContent = count;
});

resetButton.addEventListener("click", () => {
 count = 0;
 countDisplay.textContent = count;
});
```

Star Rating Component

1. Now we will be discussing the problem statement of Designing a star rating component that allows users to rate something using stars.
2. The component should display a visual representation of the rating using filled and empty stars. Users can click on the stars to select a rating.
3. Html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link rel="stylesheet" href="styles.css">
 <title>Star Rating Component</title>
</head>
<body>
 <div class="rating">
   <!-- Display five stars, each with a data-value attribute representing its rating
value -->
   <span id="stars">
      <span class="star" data-value="1">&#9733;</span>
      <span class="star" data-value="2">&#9733;</span>
      <span class="star" data-value="3">&#9733;</span>
      <span class="star" data-value="4">&#9733;</span>
      <span class="star" data-value="5">&#9733;</span>
    </span>
    <!-- Display the current rating -->
    <p>Rating: <span id="rating">0</span>/5</p>
 </div>
 <script src="script.js"></script>
</body>
</html>
```

script.js

```
// Get all star elements
const stars = document.querySelectorAll(".star");

// Get the rating display element
const ratingDisplay = document.getElementById("rating");

// Add a click event listener to each star
stars.forEach((star) => {
 star.addEventListener("click", () => {
   // Get the value from the data-value attribute
   const value = parseInt(star.getAttribute("data-value"));
   // Update the rating display and stars based on the clicked value
   updateRating(value);
 });
});

// Function to update the rating display and filled stars
function updateRating(value) {
 stars.forEach((star) => {
   // Get the value from the data-value attribute
   const starValue = parseInt(star.getAttribute("data-value"));
   // Toggle the 'filled' class based on whether the star's value is
less than or equal to
   // the selected value
   star.classList.toggle("filled", starValue <= value);
 });
 // Update the rating display text content
 ratingDisplay.textContent = value;
}
```

Css

```
<style>
```

```
    body {
      font-family: Arial, sans-serif;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
    }

    .rating {
      text-align: center;
    }

    .star {
      font-size: 24px;
      cursor: pointer;
      transition: color 0.3s ease;
    }

    .star.filled {
      color: gold;
    }
  </style>
```

Can we optimise and add only one event listener rather than adding listeners on each star ?

- Use event delegation - to be discussed tomoro

- All events will bubble up to the parent

```
// Get all star elements
// const stars = document.querySelectorAll(".star");
const parentStar = document.querySelector("#stars");
const allStars = parentStar.querySelectorAll(".star");
```

```
// Get the rating display element
const ratingDisplay = document.getElementById("rating");

/**
* evaluate the two approach and let me know which one is better
*/

// 1. Add a click event listener to each star

// Add a click event listener to each star
// stars.forEach((star) => {
//   star.addEventListener("click", () => {
//     // Get the value from the data-value attribute
//     const value = parseInt(star.getAttribute("data-value"));
//     // Update the rating display and stars based on the clicked
value
//     updateRating(value);
//   });
// });

// 2. Add a single click event listener to the parent star element
parentStar.addEventListener("click", (e) => {
 // Get the value from the data-value attribute
 const target = e.target;
 const value = parseInt(target.getAttribute("data-value"));
 // Update the rating display and stars based on the clicked value
 updateRating(value);
});

// Function to update the rating display and filled stars
function updateRating(value) {
 allStars.forEach((star) => {
   // Get the value from the data-value attribute
   const starValue = parseInt(star.getAttribute("data-value"));
```

```
   // Toggle the 'filled' class based on whether the star's value is
less than or equal to
   // the selected value
   star.classList.toggle("filled", starValue <= value);
 });
 // Update the rating display text content
 ratingDisplay.textContent = value;
}
```