Class 22 - Event Delegation

Agenda

What is Event Delegation?
Relation with concept of Bubbling
Machine coding question
    Nested Comment structure

Event delegation

1. Event delegation is a powerful pattern in JavaScript, particularly useful when working with dynamically added elements or when optimizing performance for applications with many event listeners.
2. It leverages the fact that most events in JavaScript bubble up through the DOM, meaning that an event fired on a child node will propagate up to its parent nodes. (events like focus and blur do no bubble up)
3. By taking advantage of this behavior, you can set a single event listener on a parent element to manage all events that bubble up from its children, rather than setting an event listener on each child individually.
4. Imagine you're browsing a website like Amazon, and on the product listing page, there are multiple product cards, each

containing an "Add to Cart" button. Here's how event delegation becomes useful:

a. Event delegation involves attaching a single event listener to a common ancestor element (in this case, the container holding the product cards). Instead of placing event listeners on each "Add to Cart" button individually, you attach one listener to the container.
b. Event bubbling which refers to the natural propagation of an event through the DOM hierarchy. When an event occurs on a deeply nested element, it first triggers the event handler on that element and then "bubbles up" through its ancestors as you have already seen in the previous class
c. When a user clicks the "Add to Cart" button on a product card, the event starts at the button (target) and then bubbles up through its parent elements.
d. Since you've attached a single event listener to the container holding the product cards, the event bubbles up to the container.
e. The event listener captures the event at the container level and checks whether the clicked element (the button) matches certain criteria (e.g., having the class add-to-cart).
f. If the criteria are met, the listener knows that an "Add to Cart" action is intended and can extract information about the specific product from the event's context

5. Let's go through an example with code to demonstrate event delegation and event bubbling using different categories of products (headphones, laptops, mobiles) on a web page:
6. html

```html
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
   <title>Document</title>
 </head>
 <body>
   <div id="categories">
     <div class="category" id="headphones">
       <h2>Headphones</h2>
       <div class="product">Product A</div>
       <div class="product">Product B</div>
     </div>
     <div class="category" id="laptops">
       <h2>Laptops</h2>
       <div class="product">Product X</div>
       <div class="product">Product Y</div>
     </div>
     <div class="category" id="mobiles">
       <h2>Mobiles</h2>
       <div class="product">Product P</div>
       <div class="product">Product Q</div>
     </div>
   </div>
 </body>
</html>
```

7. JS

```
const categoriesContainer =
document.getElementById("categories");

categoriesContainer.addEventListener("click", (event) => {
 const clickedElement = event.target;

 // Check if the clicked element is a product
 if (clickedElement.classList.contains("product")) {
     const parent = clickedElement.parentElement;
     const category = parent.querySelector("h2").textContent;
     // const category = clickedElement
     //   .closest(".category")
     //   .querySelector("h2").textContent;
   const product = clickedElement.textContent;

   console.log(`Clicked on ${product} in the ${category}
category.`);
   // Handle the click action for the product here
 }
});
```

a. In this example:
   i. The categoriesContainer element is the common ancestor for all categories and products.
   ii. The event listener is attached to the categoriesContainer to capture clicks on any of its child elements.

iii. When a product is clicked, the event bubbles up through the category section, reaching the categoriesContainer.

iv. The listener checks if the clicked element has the class product. If it does, it extracts the category and product information and performs the necessary action.

v. This code efficiently handles clicks on products within any category, demonstrating the combined usage of event delegation and event bubbling.

vi. With this setup, regardless of the number of categories or products, you only need one event listener to handle all clicks, making your code more maintainable and efficient.

title: Event Delegation Example-2

Let's create an example where event delegation is used to change the background color of elements by clicking on them. In this example, we'll create a set of colored boxes, and clicking on any box will change its background color using event delegation.

Html

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Document</title>
   <style>
     .color-box {
       width: 100px;
       height: 100px;
       display: inline-block;
     }
     </style>
 </head>
 <body>
   <div id="colorPalette">
     <div class="color-box" style="background-color:
rgb(255,0,0)"></div>
     <div class="color-box" style="background-color:
rgb(0,255,0)"></div>
     <div class="color-box" style="background-color:
rgb(0,0,255)"></div>
   </div>
   <script src="./code.js"></script>
 </body>
</html>
```

JS

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

colorPalette.addEventListener("click", (event) => {
 const clickedElement = event.target;

 // Check if the clicked element is a color box
 if (clickedElement.classList.contains("color-box")) {
   let color = clickedElement.style.backgroundColor;
```

```
    console.log(color)
    // color = color.replace('rgb', 'rgba').replace(')', ', 0.2)');
    document.body.style.backgroundColor = color;
  }
});
```

In this example:

1. The colorPalette element is the common ancestor for all color boxes.
2. The event listener is attached to the colorPalette to capture clicks on any of its child elements.
3. When a color box is clicked, the event bubbles up to the colorPalette.
4. The listener checks if the clicked element has the class color-box. If it does, it extracts the background color of the clicked color box.
5. The background color of the body element is then set to the extracted color, effectively changing the page's background color.
6. This demonstrates how event delegation can be used to efficiently manage interactions across a set of elements, in this case, color boxes. By using event delegation, you handle all color box clicks with a single event listener, making the code cleaner and more maintainable.

Nested Comment System

build a nested comment system where users can leave comments on a post. Each comment can have replies, creating a nested structure. Users should be able to reply to comments

CSS

```css
/* Enhanced style.css */
body {
   font-family: "Arial", sans-serif;
   background-color: #f9f9f9;
   color: #333;
   line-height: 1.6;
   padding: 20px;
 }
  #commentForm textarea {
   width: 100%;
   max-width: 600px;
   height: 100px;
   margin-bottom: 10px;
   padding: 10px;
   box-sizing: border-box;
   border-radius: 5px;
   border: 1px solid #ccc;
   resize: vertical;
 }
  #commentForm button {
   display: inline-block;
   padding: 10px 20px;
   background-color: #005a9c;
   color: #ffffff;
   border: none;
   border-radius: 5px;
   cursor: pointer;
   transition: background-color 0.3s;
 }
  #commentForm button:hover {
   background-color: #003d73;
 }
  .repliesContainer, .reply {
   background-color: #ffffff;
   margin: 10px 0;
   padding: 20px;
   border-radius: 8px;
   box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
   transition: box-shadow 0.3s;
 }
  .repliesContainer:hover, .reply:hover {
   box-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);
 }
  .repliesContainer .replyBtn, .comment .toggleReplies {
   margin-top: 10px;
   background-color: #007bff;
   color: white;
   border: none;
   border-radius: 5px;
   padding: 5px 10px;
   cursor: pointer;
   transition: background-color 0.3s;
 }
  .repliesContainer .replyBtn:hover, .comment .toggleReplies:hover {
   background-color: #0056b3;
 }
  .repliesContainer {
   margin-top: 20px;
 }
  .replyInput {
   width: 100%;
   margin-top: 10px;
   padding: 10px;
   box-sizing: border-box;
   border-radius: 5px;
   border: 1px solid #ccc;
   resize: vertical;
   display: none; /* Initially hide the reply input */
 }
  .repliesContainer:hover .replyInput {
   display: block; /* Show reply input on comment hover */
 }
  .collapsed {
   display: none;
 }
  .repliesContainer {
   padding-left: 1rem;
   border-left: 3px solid #eee;
   margin-left:10%;
 }
```

Based on css let us create our static html

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
   <title>Nested Comments System</title>
   <link rel="stylesheet" href="style.css" />
 </head>
 <body>
   <div id="commentForm">
     <textarea id="commentInput" placeholder="Write a
comment..."></textarea>
     <button id="submitComment">Comment</button>
   </div>

   <div id="commentsContainer">
     <!-- static html below for reference -->
     <div class="repliesContainer">
       <p>some comment</p>
       <button class="replyBtn">Reply</button>
       <!-- <div class="repliesContainer collapsed"></div> -->
       <textarea class="replyInput" placeholder="Write a
reply..."></textarea>
       <div class="repliesContainer">
         <p>some reply</p>
         <button class="replyBtn">Reply</button>
         <!-- <div class="repliesContainer collapsed"></div> -->
         <textarea
           class="replyInput"
           placeholder="Write a reply..."
         ></textarea>
       </div>
     </div>
   </div>
 </body>
 <script src="script.js"></script>
</html>
```

1. Inside the <body> tag, there are two main divisions (

elements).
   a. The first div with the ID commentForm contains a form for submitting comments. This form includes a <textarea> for entering a comment with a placeholder text "Write a comment…" and a <button> to submit the comment.
   b. The second div with the ID commentsContainer is where the comments will be displayed.

JS

1. DOMContentLoaded event - Wait for the DOM to be fully loaded: document.addEventListener("DOMContentLoaded", () => {...}); ensures that the code inside it only runs after the HTML document has been fully loaded. This is important to make sure that the elements we want to manipulate are available in the document.
2. code

```
// script.js
document.addEventListener("DOMContentLoaded", function () {
 });
```

3. Select DOM elements: The code selects three important elements from the DOM and assigns them to variables for easy reference:
    a. submitBtn: The button used to submit a new comment.
    b. commentInput: The input field where users type their new comment.
    c. commentsContainer: The container that will hold all the comments and replies.
4. code

```
// script.js
document.addEventListener("DOMContentLoaded", function () {
 const submitBtn = document.getElementById("submitComment");
 const commentInput = document.getElementById("commentInput");
 const commentsContainer =
document.getElementById("commentsContainer");
});
```

5. Update what to be done on submit button
    a. Add a click event listener to the submit button: When the submit button (submitBtn) is clicked, it should trigger a function that:
        i. Trims the input from commentInput to remove any leading or trailing spaces.
        ii. Checks if the trimmed input is not empty.
        iii. If it's not empty, we will call the addComment function with the trimmed input as an argument to add a new comment.
        iv. Clears the commentInput field by setting its value to an empty string.
    b. Code

```
// script.js
document.addEventListener("DOMContentLoaded", function () {
 const submitBtn = document.getElementById("submitComment");
 const commentInput = document.getElementById("commentInput");
 const commentsContainer = document.getElementById("commentsContainer");

 submitBtn.addEventListener("click", function () {
   const commentText = commentInput.value.trim();
   if (commentText) {
     addComment(commentText);
     commentInput.value = ""; // Clear input field after
adding
   }
 });
});
```

6. Now we need reply button for every comment
    a. Reply - to reply to the comment
    b. Instead of adding event listeners to each reply, we will **use event delegation** by adding a single click event listener to the commentsContainer.
    c. This approach is more efficient, especially when dealing with a dynamic number of elements (comments and replies can be added dynamically). The event listener checks the class name of the clicked target to determine the action:
        i. If the clicked target has a class name that includes "replyBtn", it will find the parent comment element, get the reply input field, check if the reply text is not empty, and then call addReply function to add the reply.
7. code

```
// Using event delegation for reply and functionality
   commentsContainer.addEventListener("click", (e) => {
       if (e.target.className.includes("replyBtn")) {
           const parentComment = e.target.parentElement;
           const replyInput =
parentComment.querySelector(".replyInput");
           const replyText = replyInput.value.trim();
           if (replyText) {
               addReply(parentComment, replyText);
               replyInput.value = ""; // Clear input field
after replying
           }
       }   });
```

8. Adding dynamically created comment to the comment container

```
function addComment(text) {
       const commentElement = document.createElement("div");
       commentElement.className = "repliesContainer";
       commentElement.innerHTML = `
         <p>${text}</p>
         <button class="replyBtn">Reply</button>
         <textarea class="replyInput" placeholder="Write a
reply..."></textarea>
     `;
       commentsContainer.appendChild(commentElement);
   }
```

9. Adding reply
   a. The thing to note here is that when we are replying we want to create the same repliesContainer structure
   b. This is needed so that we get the same reply button, same text area and same styling
   c. We create the repliesContainer dynamically and append it to the parent comment container (which is actually another repliesContainer itself)
   d. code

```
function addReply(parentComment, text) {
       // const repliesContainer =
       //
parentComment.querySelector(".repliesContainer");
       const replyElement = document.createElement("div");
       replyElement.className = "repliesContainer";
       replyElement.innerHTML = `<p>${text}</p>`;
       const btn = document.createElement("button");
       btn.className = "replyBtn";
```

```
        btn.innerText = "Reply";
        replyElement.appendChild(btn);
        const replyInput =
document.createElement("textarea");
        replyInput.className = "replyInput";
        replyInput.placeholder = "Write a reply...";
        replyElement.appendChild(replyInput);
        parentComment.appendChild(replyElement);
        parentComment.classList.remove("collapsed");
    }
```