Below is a **refined, internally accurate, exam-oriented explanation of Event Capturing**, written in the **same structured, deep, and clean style** as your Event Bubbling notes.
This version focuses on **how it works internally, execution order, control methods, and comparisons**, making it ideal for **college exams, viva, and concept mastery**.

🎯 Event Capturing in JavaScript

(Concept · Internal Flow · Control · Examples)

1️⃣ What is Event Capturing?

Event capturing is the **first phase** of the JavaScript event propagation mechanism.

In this phase, the event **travels from the root of the DOM tree down to the target element**, allowing ancestor elements to intercept the event **before it reaches the actual target**.

📘 Simple Definition

> Event capturing means the event is handled from the **outermost ancestor → down to the target element**.

2️⃣ Event Propagation Model (Complete Picture 🧠)

Every DOM event goes through **three phases**:

Phase	Direction	Description
Capturing Phase	Top → Down	Event travels from `document` to target
Target Phase	At target	Event reaches the actual element
Bubbling Phase	Bottom → Up	Event bubbles back to ancestors

📌 **Capturing always happens before bubbling**

3️⃣ Why Event Capturing Exists (Internal Reason)

Event capturing exists to:

- Allow **early interception** of events
- Enable **global-level validation or filtering**
- Provide **fine-grained control** over event flow
- Support advanced UI frameworks and libraries

Although bubbling is more commonly used, capturing is essential in **security, access control, and global monitoring** scenarios.

4️⃣ Direction of Event Capturing (VERY IMPORTANT)

```css
document
  ↓
html
  ↓
body
  ↓
parent element
  ↓
target element
```

✔️ Event flows **downward**

✔️ Opposite of bubbling

✔️ Executes **before target & bubbling handlers**

5️⃣ How to Enable Event Capturing

By default, event listeners listen during **bubbling phase**.

To enable capturing, pass `true` (or `{ capture: true }`) as the **third argument** of `addEventListener()`.

✅ Syntax

```js
element.addEventListener("click", handler, true);
```

OR (modern style)

```js
element.addEventListener("click", handler, { capture: true });
```

6️⃣ Basic Example: Event Capturing Order

🧠 Scenario

Clicking the **button**, but capturing listeners execute **from parent to child**.

✅ Code Example

```html
```

```
<div id="container">
  <button id="btn">Click Me</button>
</div>

<p id="output"></p>

<script>
const output = document.getElementById("output");

document.getElementById("container")
.addEventListener("click", () => {
  output.innerHTML += "Container (capture)<br>";
}, true);

document.getElementById("btn")
.addEventListener("click", () => {
  output.innerHTML += "Button (capture)<br>";
}, true);
</script>
```

🔍 Output (Click Button)

```scss
Container (capture)
Button (capture)
```

✔️ Confirms **top → down execution**

7️⃣ Capturing vs Bubbling (Side-by-Side 🔥)

Feature	Capturing	Bubbling
Phase Order	1st	3rd
Direction	Root → Target	Target → Root
Default	❌ No	✅ Yes
Listener Flag	`true`	`false`
Use Case	Intercept early	Delegation

8️⃣ Using `preventDefault()` in Capturing Phase

🧠 Key Point

`event.preventDefault()` **stops the browser's default action**,
but **does NOT stop propagation**.

✅ Example

```html
<a href="https://example.com" id="link">Click Me</a>

<script>
document.getElementById("link")
.addEventListener("click", function(event) {
  alert("Capturing phase");
  event.preventDefault();
}, true);
</script>
```

✔️ Navigation is prevented

✔️ Capturing handler executes first

9️⃣ Stopping Propagation in Capturing Phase 🚫

🧠 What Happens?

If `stopPropagation()` is called during capturing:

- Event **never reaches target**
- Bubbling phase **never occurs**

✅ Example

```html
<div id="parent">
  <button id="child">Click</button>
</div>

<p id="output"></p>

<script>
const output = document.getElementById("output");

document.getElementById("parent")
.addEventListener("click", function(event) {
  output.innerHTML += "Parent captured<br>";
  event.stopPropagation();
}, true);

document.getElementById("child")
.addEventListener("click", function() {
  output.innerHTML += "Child captured<br>";
}, true);
</script>
```

🔍 Output

```nginx
Parent captured
```

✔️ Event never reaches child

✔️ Bubbling completely blocked

🔟 event.target vs event.currentTarget (Capturing Context)

Property	Meaning
`event.target`	Element where event originated
`event.currentTarget`	Element handling event

Even during capturing:

```js
 console.log(event.target);       // Button
console.log(event.currentTarget); // Parent
```

1️⃣1️⃣ Real-World Use Cases of Capturing

✔️ Global input validation

✔️ Security restrictions

✔️ Modal overlays

✔️ Blocking clicks before reaching children

✔️ Framework-level event control

1️⃣2️⃣ Common Mistakes ❌

- Assuming capturing is default
- Forgetting third parameter in `addEventListener`

- Confusing bubbling output with capturing
- Using `preventDefault()` instead of `stopPropagation()`

1️⃣3️⃣ Exam / Viva Ready Answers 🎓

Q: What is event capturing?

👉 It is the phase where an event propagates from the root to the target.

Q: How to enable capturing?

👉 Pass `true` or `{ capture: true }` in `addEventListener`.

Q: Does capturing happen before bubbling?

👉 Yes, always.

1️⃣4️⃣ Key Takeaways ✅

✔️ Capturing = Top → Down

✔️ Happens before target & bubbling

✔️ Disabled by default

✔️ Powerful but less used

✔️ Essential for full event control

⏭️ What Should We Learn Next?

👉 **stopPropagation vs stopImmediatePropagation**

👉 **preventDefault vs stopPropagation**

👉 **Event Delegation + Bubbling + Capturing Combined**

Just type `next` and we'll continue 🚀