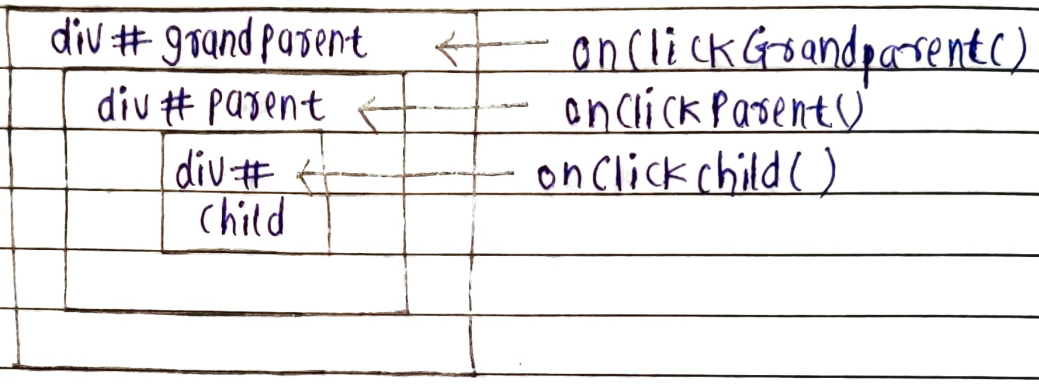
↑ bubbling ↓ capturing

* Event Bubbling and Capturing / Trickling



third argument as false

In case of event bubbling, if you clicked on div#child then
onClickchild() method will be called first
then it moves up to the hierarchy and goes directly till the end of dom
after onclickchild() method
onclickparent() method will be called and then
onclick GrandParent() will be called.

And opposite to this is event capturing
it captures down the dom tree
so when you click on div#child then
onclick Grandparent() method will be called first
then onclickParent() method will be called
and then onclickchild() will be called.

third argument as true

Some scenarios :-

Page No.
Date

① in case of bubbling: when you click on div#parent Here, first onClickParent() method/handler will be called and then onClickGrandparent() method will be called.

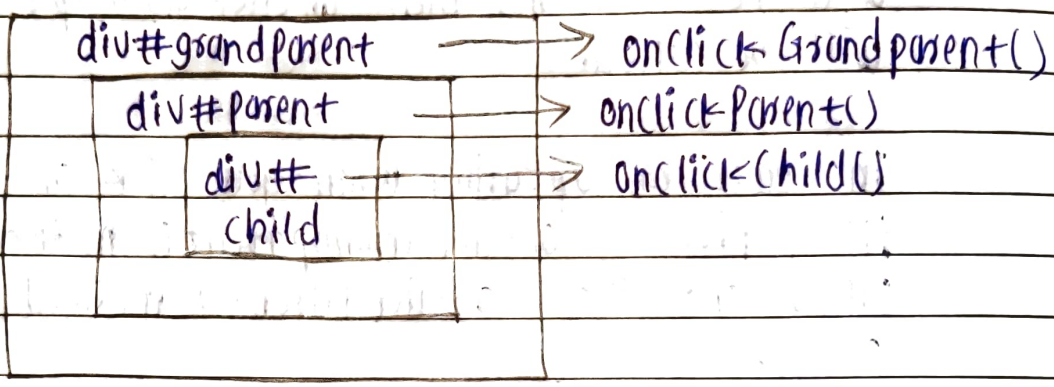
in case of capturing when you click on div#parent Here, first onClickGrandparent() method will be called and then onClickParent() method will be called.

IMP Event capturing is also called as Event Trickling (trickling down and bubbling up)

[Event Bubbling will be used by default
(if you don't pass any third argument)
(i.e. third argument will be false).

[by default first capturing happens then bubbling happens
(here case is when you are passing third argument as both
means for some true and for some false

example while we are passing third argument :-



JS code

```
document.querySelector("#grandparent")
  .addEventListener('click', () => {
    console.log("Grandparent clicked");
  }, true); // capturing

document.querySelector("#parent")
  .addEventListener('click', () => {
    console.log("Parent clicked");
  }, false); // bubbling

document.querySelector("#child")
  .addEventListener('click', () => {
    console.log("Child clicked");
  }, false); // bubbling
```

↓

① when you clicked on #child
grandparent clicked
child clicked
parent clicked

② when you clicked on #parent
Grandparent clicked
parent clicked

③ when you clicked on #grandparent
Grandparent clicked

Scenario :-
#grandparent → third argument as false (bubbling)
#parent → true (capturing)
#child → false (bubbling)

① when user clicked on #child
Parent clicked
child clicked
Grandparent clicked

② when user clicked on #parent
Parent clicked
Grandparent clicked

③ when user clicked on #grandparent
Grandparent clicked

Scenario :- #grandparent → true
#parent → true
#child → false

clicked on	#child	#parent	#grandparent
	Grandparent clicked	Grandparent	Grandparent
	parent clicked	parent	
	child clicked		

If you want to stop event bubbling, this can be achieved by the use of event.stopPropagation() method. If you want to stop the event flow from Event target to top element in DOM, event.stopPropagation() method stops the event to travel to bottom to top.

Consider same previous example :-

① scenario
all third argument false (bubbling)
but in #parent (only in parent)

```
.addEventListener('click', (event) => {
    console.log("Parent clicked");
    event.stopPropagation();
}, false);
```

when user clicked on #child
child clicked
Parent clicked

when clicked on #parent
Parent clicked

when clicked on #grandparent
Grandparent clicked

Page No.
Date

② scenario
all third argument false (bubbling)
only in #child (event.stopPropagation)

User clicked on #child
child clicked

User clicked on #parent
parent clicked
Grandparent clicked

User clicked on #grandparent
Grandparent clicked

③ scenario
all third argument true (capturing)
only in #child (event.stopPropagation)

User clicked on #child
Grandparent clicked
Parent clicked
child clicked

User clicked on #parent
Grandparent clicked
parent clicked

User clicked on #grandparent
Grandparent clicked

④ scenario
all third argument true (capturing)
only in #grandparent (event.stopPropagation)

User clicked on #child
Grandparent clicked

User clicked on #parent
Grandparent clicked

User clicked on #grandparent
Grandparent clicked

* Events
Events are responsible for interaction of JS with HTML web pages. The general definition of event is any act can occurs by someone. Events can be subscribed by listeners that occurs only when particular event can be fired.
ex. click event

* Event Flow
event flow is the order in which event is received on web page. If you click on an element like div which is nested to other elements, before the click is performed on target element. It must trigger the click event each of its parent element first starting at top with global window object.
By default, every element of HTML is child of window object.

Page No.
Date

* Event Delegation

event bubbling allows us to take advantage of event delegation - this concept relies on the fact that if you want some code to run when you select any one of a large number of child elements, you can set the event listener on their parent and have events that happen on them bubble up to their parent rather than having to set the event listener on every child individually. Remember, bubbling involves checking the element the event is fired on for an event handler first, then moving up to the element's parent.

A good example is a series of list items - if you want each one to pop up a message when selected you can set the "click" event listener on parent <ul>, and events will bubble from list items to the

(01)

Example :-

```
<div>
  <ul id="category">
    <li id="laptops">laptops</li>
    <li id="cameras">cameras</li>
    <li id="shoes">shoes</li>
  </ul>
</div>
```

- laptops
- cameras
- shoes

```
document.querySelector("#category")
    .addEventListener('click', (e) => {
        if(e.target.value
        if(e.target.tagName == "LI") {
            window.location.href = "/" + e.target.id;
        }
});
```

Here when user click on li element #laptops on dom then user will redirect to localhost:8080/laptops, similarily when user clicks on li element #cameras then user will redirect to localhost:8080/cameras and similarily when user clicks on li element #shoes then user will redirect to the localhost:8080/shoes

(02)

example

```
<div id="form">
    <input type="text" id="name" data-uppercase>
    <input type="text" id="pan">
    <input type="text" id="aadhar" data-uppercase>
</div>
```

```
document.querySelector("#form")
    .addEventListener('keyup', (e) => {
        if(e.target.dataset.uppercase != undefined) {
            e.target.value = e.target.value.toUpperCase();
        }
});
```

Here whenever user is going to type something in input field which has id #name, whatever user is typing, it will be in uppercase, we have provided this functionality.
similarly, whenever user will type in input field which has id #aadhar, whatever user is typing it will be uppercase (we can see effect on dom)

This functionality will not be for input field which has id #pan because it does not have attribute data-uppercase

[we are providing this functionality considering data-uppercase attribute
i.e.
those input field has attribute data-uppercase to that input field, this functionality will be applied]

Page No.
Date

* Benefits of Event Delegation :-

① less memory usage, better performance
② less time required to set up event handlers on page. (we need to write less code)
③ The "document" object is available immediately. As long as the element is rendered, it can start functioning correctly without delay. you don't need to wait for "DOMContentLoaded" or "load" events.
④ event delegation technique utilizes the event bubbling to handle event at a higher level in the DOM than the event on which the event originated.

* Disadvantages of event delegation (cons) :-

① all the events are not bubbled up.
(e.g. blur, focus, scrolling, etc)
(keep this in mind)

② to let event delegation work, you need to not use stopPropagation() and let event bubbled up.

[use event delegation]