A call to addEventListener will override the previous one, or a call to removeEventListener will remove a listener, only when the handler functions specified for that event type are **strictly equal**.

Here's one idea: define your handler as a separate function in a closure, as follows:

obj = function() {

function handler() { /\* handle the click; "this" is the element \*/ }

return {

draw: function() {

this.elt.addEventListener('click', handler);

//draw a bunch of stuff

},

undraw: function() {

this.elt.removeEventListener('click', handler);

//undraw a bunch of stuff

}

};

}();

Now, since handler is always strictly equal to itself, the removeEventListener will successfully remove the handler. Or, a second addEventListener will not do anything (just leave the current handler in place).

However, handler has no access to the this of the object; it will be called with the event's target element as its this. In order to get the object's this into the event handler, you may be tempted to try

this.elt.addEventListener('click', handler.bind(this));

but this will fail to accomplish what you want, because the value of handler.bind(this) is different each time the method is invoked, so you will again end up with redundant event handlers, or removeEventListeners which do not work.

If you really want the object's this in the handler, and can't figure out how to retrieve it from the event, you could initialize a bound version of handler in some init function:

obj = {

handler: function(event) { /\* handle the click; "this" is the object \*/ },

draw: function() {

this.elt.addEventListener('click', this.handler);

//draw a bunch of stuff

},

undraw: function() {

this.elt.removeEventListener('click', this.handler);

//undraw a bunch of stuff

},

init: function() {

this.handler = this.handler.bind(this);

return this;

}

}.init();

Since this.handler is always identical to itself, this works as expected.

Using EventListener

A somewhat more elegant way to solve this problem is to pass to addEventListener, instead of a function, an object with the [EventListener](https://developer.mozilla.org/en-US/docs/Web/API/EventListener) interface, which is any object that implements the specially-named handleEvent method, which could be the 'this' object itself, so you can do:

obj = {

handleEvent: function(event) {

// "this" is the object

if (event.type === 'click') {

// do stuff

}

},

draw: function() {

this.elt.addEventListener('click', this);

},

undraw: function() {

this.elt.removeEventListener('click', this);

}

};

Note the this being passed to addEventListener. In other words, we are passing the object itself, in its incarnation as an instance of EventListener, by virtue of its having implemented handleEvent. handleEvent is a full-fledged method of the object and as such has full access to all its methods and properties, and because this is identical to itself, the adding, adding again, and removing behavior works as you want.

I don't see this approach used very much, but it can make event handling more streamlined, especially if you add some sugar around it, such as arranging for individual methods to handle each event type:

obj = {

handleEvent: function(event) {

return this[event.type](event); // dispatch to method with name of event

},

click: function(event) {

// handle click; "this" is obj

},

draw: function() {

this.elt.addEventListener('click', this);

},

undraw: function() {

this.elt.removeEventListener('click', this);

}

};