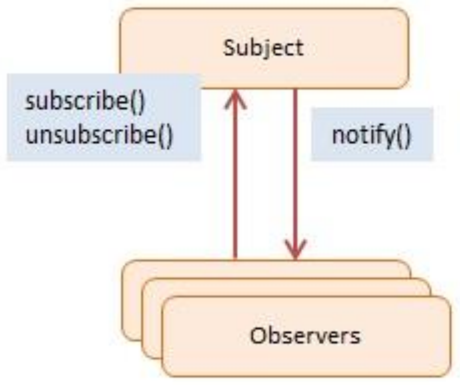
Summary

The Observer pattern offers a subscription model in which objects subscribe to an event and get notified when the event occurs. This pattern is the cornerstone of event driven programming, including JavaScript. The Observer pattern facilitates good object-oriented design and promotes loose coupling.

When building web apps you end up writing many event handlers. Event handlers are functions that will be notified when a certain event fires. These notifications optionally receive an event argument with details about the event (for example the x and y position of the mouse at a click event).

The event and event-handler paradigm in JavaScript is the manifestation of the Observer design pattern. Another name for the Observer pattern is Pub/Sub, short for Publication/Subscription.

Diagram



Participants

The objects participating in this pattern are:

- **Subject** -- In sample code: **Click**
  - maintains list of observers. Any number of Observer objects may observe a Subject
  - implements an interface that lets observer objects subscribe or unsubscribe
  - sends a notification to its observers when its state changes
- **Observers** -- In sample code: **clickHandler**
  - has a function signature that can be invoked when Subject changes (i.e. event occurs)

Sample code in JavaScript

The Click object represents the Subject. The clickHandler function is the subscribing Observer. This handler subscribes, unsubscribes, and then subscribes itself while events are firing. It gets notified only of events #1 and #3.

Notice that the fire method accepts two arguments. The first one has details about the event and the second one is the context, that is, the this value for when the eventhandlers are called. If no context is provided this will be bound to the global object (window).

The log function is a helper which collects and displays results.

```javascript
function Click() {
    this.handlers = [];  // observers
}

Click.prototype = {

    subscribe: function(fn) {
        this.handlers.push(fn);
    },

    unsubscribe: function(fn) {
        this.handlers = this.handlers.filter(
            function(item) {
                if (item !== fn) {
                    return item;
                }
            }
        );
    },

    fire: function(o, thisObj) {
        var scope = thisObj || window;
        this.handlers.forEach(function(item) {
            item.call(scope, o);
        });
    }
}

// log helper

var log = (function() {
    var log = "";

    return {
        add: function(msg) { log += msg + "\n"; },
        show: function() { alert(log); log = ""; }
    }
})();

function run() {

    var clickHandler = function(item) {
        log.add("fired: " + item);
    };

    var click = new Click();

    click.subscribe(clickHandler);
    click.fire('event #1');
    click.unsubscribe(clickHandler);
    click.fire('event #2');
    click.subscribe(clickHandler);
    click.fire('event #3');

    log.show();
}
```

Run

JavaScript Optimized Code

The example given is not optimized for JavaScript. Significant improvements can be obtained by applying advanced JavaScript techniques resulting in more effective, robust, and maintainable apps.

To learn how, check our comprehensive **JavaScript + jQuery Design Pattern Framework**. This unique package includes optimized JavaScript for all GoF patterns using more advanced features, such as namespaces, prototypes, modules, function objects, closures, anonymous functions, and ohters.