# Object-Oriented CSS

By Kevin P. Wojdak

## *Think Object*

I've seen it. You've seen it. And, anyone else who's done a lot of work with Cascading Style Sheets (CSS) has seen it. You're a website developer and you want to get some insight into how someone's page works so, you open their stylesheet file. You're hoping to see how something is formatted and, instead, you find a messy stylesheet…you see that each and every element from their HTML page has been styled as a unique item and all of the item characteristics are crammed in and duplicated across many similar styles. This type of stylesheet fails to utilize the power of CSS and it's similarities to object-oriented programming.

Designers may not realize it but CSS has many similarities to object-oriented programming languages. By understanding these similarities, you can start thinking of the elements of an interface as "objects" and therefore create a well-structured and efficient stylesheet based around these objects.

Just as you do when you design and plan out any website or interface project, you need to design and plan your stylesheet.

## *Relationship between CSS and Object-Oriented Programming*

In object-oriented programming (OOP), developers build generalized objects having common reusable functionality. They then create new objects by adding extra functionality to the previously defined common objects. The new objects inherit the attributes and functionality of the old objects and receive any new functionality added to them.

CSS works in a similar way. Proper CSS definition begins with general HTML element types such as <div> layers, <p>, or <img> tags. Each of these element types is a common object and they are the most generalized objects in a HTML page. Style attributes are defined for each object via CSS selector and Class definitions in the stylesheet. General functionality is defined for these objects and then they inherit specialized attributes through additional Class definitions attached to them.

Starting with a Type selector such as a paragraph tag ( <p> ), the author defines general style attributes for elements common to all paragraphs. Once those common attributes are defined, every paragraph object on the HTML pages receives those attributes and acts similarly.

After defining general style attributes, the author creates specialized groups of common objects by defining unique characteristics of groups of objects in selectors called Classes.

A Class definition is attached to any object that should have the same characteristics as any other object in a group.

For example, when building an application using forms that has input buttons and fields having the same look and feel from screen-to-screen, a Class is defined for each type of input. Input objects are represented by the <input> tag and are given a HTML "type" attribute defining them as buttons, text fields, radio buttons, or checkboxes. Each of these object types have general style attributes defined giving them a common look and feel.

To further specialize an object group, we might want to have two types of input field (<input>) objects such as enabled and disabled fields. The disabled input field may have a gray background color. The author defines these style attributes in a Class named "textDisabled" and assigns that Class to the <input> tag.

To illustrate, the HTML markup for a disabled input field is:

*<input type="text" class="textDisabled"></input>*

Whenever we place an <input> object assigned this Class on the screen, it will have a gray background. We define a second class for the enabled text field and use that wherever we are not showing a field as disabled. In this way, we keep control over the look and feel of each element or object through the Classes.

Just as a single change to an object Class in OOP ripples through all of the objects of that Class, a single change to a Class in a stylesheet cascades throughout the application wherever that Class is attached to an object. Within an OOP Class, a developer defines common attributes that apply to more than one element or object. These common attributes are then inherited by each object that has the Class defined as part of the object. The Class is reusable by many objects giving them all these same characteristics.

Finally, after defining attributes for objects and groups, we define "unique" style attributes for specific objects. These styles change a portion of one unique object within a screen. In CSS, the selector defined for unique objects is called an "Id" selector. In HTML markup, the author makes an object unique by giving it an "id" attribute and setting the value for that attribute to a unique name or number that no other object on the page shares. Then, in the CSS, style attributes unique to that one object are defined in the Id selector created for the object. The stylesheet Id selector matches the id attribute of your HTML object.

When the page is rendered on screen, the HTML objects get all of their style attributes from the CSS Types, Classes, and Id selectors defining their complete look and feel. With a good understanding of these component CSS selectors, the author very efficiently controls the presentation of the objects on the page.

Using the <input> object as an example, the technical hierarchy of the three CSS selectors in a style sheet for an object is as follows:

```
input { …Type of object style attributes }
        .textDisabled {  …Class (group) attributes  }
                #fld123  {  … unique field specific attributes (Id)  }
```

The <input> tag becomes a Type selector by specifying the object type of “input” within the stylesheet with no punctuation or notation before the selector name.

Class selectors are defined by placing a dot before the selector name and Id selectors are defined with an octotorphe (“#”) before the unique selector identifier. Follow normal CSS syntax to define the attributes of each.

Using the CSS style definitions above, the HTML for the disabled text <input> field with an id of “fld123” is:

*<input type=”text”  id=”fld123” class=”textDisabled”></input>*

As your comfort level with the relationship of CSS to object-oriented programming increases, you will start to think of the elements of a HTML page as “objects”. Then, by combining that understanding with a good architectural strategy for your stylesheets, your CSS takes on a logical structure and there is more control over your design when building the actual page.

## Plan Your CSS

Now that you understand the basic relationship of CSS to object-oriented programming, start applying an object-oriented approach to your stylesheet definition. Begin by looking at your design, breaking it down into objects and object groups, and then map out the basics.

For example, if your design is for a shopping cart, you probably have buttons to process different actions. Look at all of these buttons together and find the similarities and differences. Then, build your stylesheet from the ground up defining the basic attributes of the buttons in the “input” Type selector. After that, group similar presentation attributes into Class selectors. Finally, as you build the actual pages, any unique attributes of an individual object are defined in an Id selector just for that object.

By defining stylesheets using a structured and controlled method, just as in object-oriented programming, you create reusable Type and Class selectors. Reusing styles reduces the size of your stylesheets and makes it easier to manage and maintain them.

*Kevin P. Wojdak is a freelance web and IT professional in the Chicago area. His skills and experience have allowed him to pioneer new techniques and tricks in Rich Interface Development. He enjoys transforming other people’s creative inspiration into technical reality. Visit his personal website, Woj’s Twisted World (http://www.wojzworld.com) to see his resume or to find other articles and development techniques.*