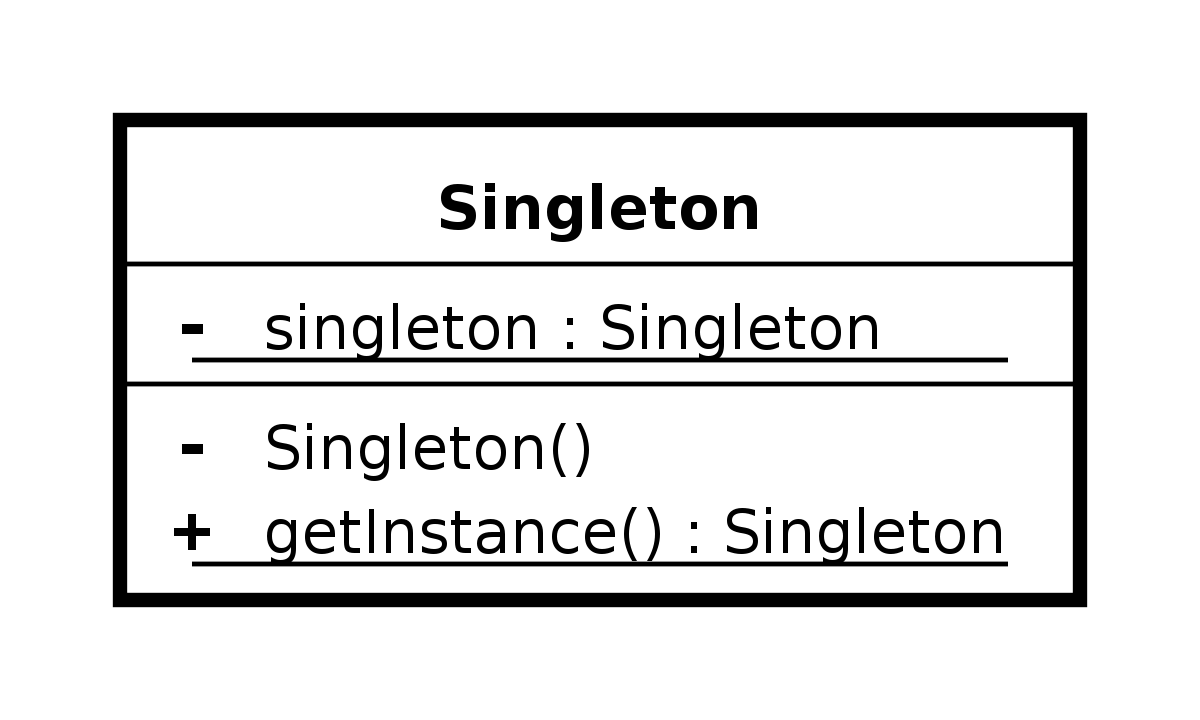
# Patrón Singleton



En AngularJS, el patrón singleton es muy utilizado ya que todos los servicios que se utilizan para programar en este lenguaje son singleton, es decir, solo hay una instancia de cada uno de ellos en memoria en tiempo de ejecución.

### Código fuente AngularJS

function getService(serviceName) {  
 if (cache.hasOwnProperty(serviceName)) {  
 if (cache[serviceName] === INSTANTIATING) {  
 throw $injectorMinErr('cdep', 'Circular dependency found: {0}', path.join(' <- '));  
 }  
 return cache[serviceName];  
 } else {  
 try {  
 path.unshift(serviceName);  
 cache[serviceName] = INSTANTIATING;  
 return cache[serviceName] = factory(serviceName);  
 } catch (err) {  
 if (cache[serviceName] === INSTANTIATING) {  
 delete cache[serviceName];  
 }  
 throw err;  
 } finally {  
 path.shift();  
 }  
 }  
}

Haciendo la analogía con el diagrama del patrón singleton tenemos que:

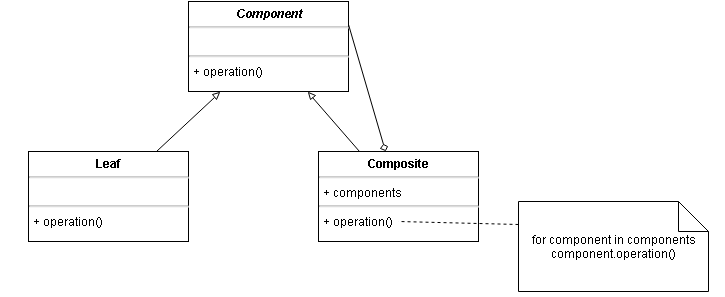
Clase **Singleton:** Es el cache o memoria de la aplicación, manejando por el archivo “angular.js”. Es el código fuente del framework.

**-Singleton: Singleton:** Se refiere a la instancia de cada uno de los diferentes servicios que se pueden crear y utilizar. Estas instancias se encuentran en la memoria caché.

**-Singleton():** Es el constructor de la clase. En este caso, angularJS no utiliza este método.

**+GetInstance(): -Singleton:** En este caso se refiere al metodo “GetService”, el cual recibe como parámetro el nombre del servicio a instanciar. Si el mismo ya está instanciado lo devuelve. Si no lo está, lo guarda en memoria por primera vez y lo retorna al usuario.

## Patrón Composite



En angularJS, el patrón composite se usa claramente en la composición de las vistas. Dentro de cada página html, podemos tener una o más directivas. Una directiva puede reemplazar o insertar una porción de HTML que, a su vez, también puede tener una o más directivas dentro, las cuales también se mostrarán. Así hasta que ya no haya ninguna directiva para reemplazar.

Este patrón es muy importante para reutilizar los componentes.

**Página Principal**

<html>  
 <head>  
 </head>  
 <body>  
 <zippy title="zippy">  
 zippy!  
 </zippy>  
 </body>  
</html>

**Directiva Zippy**

**myModule.directive('zippy', function () {  
 return {  
 restrict: 'E',  
 template: '<div><div class="header"></div><div class="content" ng-transclude leaf></div></div>',  
 link: function (scope, el) {  
 el.find('.header').click(function () {  
 el.find('.content').toggle();  
 });  
 }  
 }  
});**

**myModule.directive(leaf, function () {  
 return {  
 template: '<div><div class="hola">Hola</div></div>',  
 link: function (scope, el) {  
 el.find('.header').click(function () {  
 el.find('.content').toggle();  
 });  
 }  
 }  
});**

Como se puede ver en este ejemplo de código, la etiqueta “zippy” del archivo raíz se reemplaza con el código HTML insertado dentro de la directiva con el mismo nombre. Esta, a su vez, posee otra directiva más llamada “leaf”, que nuevamente reemplazará el html correspondiente. Esto podría continuar hasta que ya no haya ninguna directiva para reemplazar.

Haciendo la analogía con el diagrama del patrón singleton tenemos que:

**Component:** Es el archivo html de raíz en el cual se insertarán las directivas primarias, directamente en el HTML.

**-Composite::** Es cualquiera de las directivas que se encuentren en el html y que angular procesa para reemplazar o insertar nuevo html. Para ser **composite** deberá, a su vez, tener otras directivas dentro suyo para procesar.

**-Leaf:** Son las directivas terminales, es decir, que no poseen otras directivas dentro de ellas.

**-**La función **operation()** puede considerarse que es aquella para mostrar el html correspondiente.