

MP0488.

**Desarrollo de interfaces de usuario
UF3. Componentes**

3.4. Desarrollo de software basado en componentes

Índice

☰	Objetivos	3
☰	Desarrollo basado en componentes	4
☰	Etapas DSBC	6
☰	Reutilización de software	9
☰	#7: ventanas modales	14
☰	POO en JS	22
☰	Resumen	24

Objetivos

En esta lección perseguimos los siguientes objetivos:

- 1 Conocer las ventajas y desventajas del desarrollo de componentes.
 - 2 Conocer las etapas DSBC.
 - 3 Aprender qué es la reutilización de software, así como sus dificultades, ventajas y desventajas.
 - 4 Comprender los procesos y tipos de la reutilización de software.
 - 5 Conocer y manejar los niveles de reutilización.
-

¡Ánimo y adelante!

Desarrollo basado en componentes

Vamos a conocer los modelos basados en componentes.

Como tal, un **modelo basado en componentes** es un proceso que concede particular importancia al diseño y la construcción de sistemas basados en ordenadores que utilizan componentes de software reutilizable, dando paso a la integración de software como centro de enfoque de las nuevas tecnologías.

Superficialmente, se **parece bastante** a la ingeniería de software orientada a objetos.

El proceso comienza cuando un equipo establece requisitos para el sistema que se construirá a través de la investigación de requisitos, estableciendo un diseño arquitectónico general, sin entrar en más detalles.

Para todo este proceso, podemos plantearnos varias preguntas:

- ¿Hay componentes comerciales de línea -conocidos por las siglas CDL- disponibles para implementar los requisitos?
- ¿Hay disponibles componentes reutilizables desarrollados internamente para implementar los requisitos?
- Las interfaces para los componentes disponibles, ¿son compatibles dentro de la arquitectura del sistema que se construirá?

- i** Si los requisitos no se pueden implementar con CDL o desarrollo propio, se aplicarán métodos de software para la construcción de nuevos componentes.

Pero con el uso de modelos de desarrollo basado en componentes, podremos obtener algunos beneficios:

- 1 Alcanzar mayor nivel de reutilización del software.
- 2 Simplificar las pruebas ejecutadas en cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- 3 Simplificar el mantenimiento del sistema.
- 4 Alcanzar mayor calidad, ya que un componente puede ser construido y luego mejorado continuamente por expertos a lo largo del ciclo de vida del mismo.

VENTAJAS

- Al realizarse un exhaustivo análisis de riesgo, se estudian todos los riesgos potenciales, y se seleccionan una o varias alternativas, propuestas para reducir o eliminar los riesgos.
- Une los mejores elementos de los restantes modelos.
- Reduce riesgos del proyecto.
- Incorpora objetivos de calidad.
- Integra el desarrollo con el mantenimiento.

DESVENTAJAS

- Genera mucho tiempo en el desarrollo del sistema.
- Requiere experiencia en la identificación de riesgos.
- Genera mucho trabajo adicional.
- Cuando un sistema falla, se pierde tiempo y coste dentro de la empresa.
- Exige una cierta habilidad por parte de los analistas.

Etapas DSBC

A continuación, vamos a conocer las ventajas y desventajas, así como las etapas del desarrollo basado en componentes: cualificación, adaptación, composición y actualización.

Aproximadamente el **60%** del diseño y del código de aplicaciones es reutilizable y un **75%** de las funciones son comunes a más de un programa.

Solo el **15%** del código encontrado en muchos sistemas es único y novedoso para una aplicación específica.

VENTAJAS

- Funcionalidad mejorada.
- Reducción de costes y tiempos.
- Reutilización del software.
- Simplificación de pruebas y mantenimiento de los sistemas.
- Mayor calidad en el desarrollo de nuevos componentes.
- Ciclos de desarrollo más cortos.
- Reutilización del software.

DESVENTAJAS

- Se genera un exceso en los tiempos y mucho trabajo adicional.
- Se genera confiabilidad de los componentes.
- Es posible que el código de los componentes no esté disponible para la reprogramación de componentes por parte de los usuarios.
- Se generan ciertas restricciones en el uso de componentes.

El **CBD (Component Based Development)**, desarrollo basado en componentes, es una arquitectura extensible para soportar un proceso de ciclo completo.

El **desarrollo CBD** ejercerá influencia sobre todas las dimensiones y aspectos de la composición de aplicaciones, incluyendo todos los tipos de clientes, servidores de aplicación y bases de datos.

El predecesor al CBD ha sido el desarrollo orientado a objetos (conocido por **POO**), donde los objetos fueron el primer elemento en ofrecer altos niveles de reutilización en el sector. Aunque la falta de sofisticación de los IDEs y la ausencia de una infraestructura común para permitir la interoperabilidad de los objetos ha estado impidiendo su adopción generalizada.

Ahora, en el desarrollo CBD se redefinen objetos dentro del contexto de una infraestructura estandarizada para interoperabilidad, *frameworks* (marcos) para la construcción y ensamblaje de aplicaciones, y componentes pre-construidos que se adscriben a infraestructuras y *frameworks* de componentes.

Etapas del desarrollo CBD

Componentes monolíticos

Describen el estado en que se encuentran los componentes. Sin embargo, ninguno de los estándares de componentes y frameworks actuales han resuelto los problemas de portabilidad e interoperabilidad que permitirían que los componentes fueran distribuidos o persistieran en múltiples frameworks de vendedores. *No existen componentes estructurales estandarizados y, en consecuencia, se están introduciendo frameworks con un alto contenido de componentes estructurales propietarios.*

Componentes distribuidos

Se cambia de un modelo de construcción física al diseño lógico, donde los componentes monolíticos ceden paso a componentes distribuidos, al permitir los estándares de componentes de mayor madurez, y los nuevos componentes estructurales la interoperabilidad y portabilidad de componentes. Estos frameworks se concentran más en el modelo de desarrollo y soportan más aspectos del ciclo de vida de los componentes.

Persistencia de componentes

Tiene lugar cuando prácticamente toda la infraestructura asociada al proceso CBD se convierte en un producto de uso corriente o *commodity*, lo que hace que aumente el número de proveedores de servicios con muchos tipos de componentes que pueden ser adaptados fácilmente de acuerdo con reglas comerciales. Es decir, aumenta la interoperabilidad entre vendedores, sin requerir relaciones tecnológicas bilaterales específicas.

Las cuestiones relativas a herramientas (cobertura del ciclo de vida, paradigma de desarrollo, metáfora de utilización y disponibilidad de componentes pre-construidos) se convierten en factores clave de diferenciación competitiva.

Reutilización de software

La reutilización de software se basa en crear nuevo software basándose en uno ya existente, en lugar de rediseñar todo desde un principio.

Para ello, se emplean elementos creados en desarrollos anteriores, mejorando la calidad y reduciendo el precio de los desarrollos.

La **reutilización de software** aparece como una alternativa para desarrollar aplicaciones y sistemas de una manera más *eficiente, productiva y rápida*.

Elementos que intervienen en la reutilización

- 1 Especificaciones de requerimientos previamente concebidas.
- 2 Diseños previamente definidos (estructuras de datos, algoritmos, etc.).
- 3 Código probado y depurado con anterioridad.
- 4 Planes y casos de prueba previamente utilizados.
- 5 Personal cualificado (aprovechamiento de la experiencia de los ingenieros de un proyecto a otro).
- 6 Paquetes de software de propósito general.

VENTAJAS

- Reducir el tiempo de desarrollo y, por supuesto, los costes, incluyendo la inversión inicial.
- Incrementar la productividad, fiabilidad y eficiencia, lo que no requiere cambios sustanciales en la organización.
- No tener que reinventar las soluciones.
- Facilitar la compartición de productos del ciclo de vida, con un proceso de implantación incremental.
- Consistencia y familiaridad, ya que es soportado por multitud de herramientas y no requiere procesos desmesurados de formación.

INCONVENIENTES

- En muchas empresas no existe plan de reutilización (no se considera prioritario).
- Escasa formación y mucha resistencia del personal, además de un pobre soporte metodológico.
- Uso de métodos que no promueven la reutilización (estructurados).
- Gastos de soportar la reutilización de software.

Los aspectos para la **reutilización** de software existente se basan, principalmente, en utilizar componentes ya desarrollados que cumplen los requisitos del proyecto actual.

COMPONENTES YA DESARROLLADOS

El software existente se puede adquirir de una tercera parte, o provenir de uno desarrollado internamente para un proyecto. Se denominan componentes **CCYD (componentes comercialmente ya desarrollados)**, listos para utilizarse en el proyecto actual y totalmente validados.

CON PRE-TEST COMPLETO

Especificaciones, diseños, código o datos de prueba existentes desarrollados para proyectos anteriores que son similares al software que se va a construir para el proyecto actual. Las modificaciones requeridas para estos componentes tendrán un riesgo relativamente bajo.

CON PRE-TEST PARCIAL

Especificaciones, diseños, código o datos de prueba existentes ya desarrollados para proyectos anteriores que se relacionan con el software del proyecto actual, pero que requerirán una modificación sustancial, limitando la experiencia sólo al área de aplicación representada por estos componentes.

COMPONENTES NUEVOS

Aquellos componentes de software que el equipo debe programar específicamente para las necesidades del proyecto actual.

Unidades de software a reutilizar

Pero, ¿sólo podemos reutilizar componentes? Veamos qué otras unidades se pueden reutilizar.

Reutilización de sistemas de aplicación

- Se incorpora sin ningún cambio en otros sistemas.
- Configuración de la aplicación para diferentes clientes.

Reutilización de componentes

- Varía en tamaño desde subsistemas hasta objetos simples.

Reutilización de objetos y funciones

- Reutilización de componentes software que implementen una única función.

Una vez que comencemos a reutilizar los elementos que nos convengan en cada proyecto, podremos tener los siguientes tipos de reutilización:

Oportunista

El ingeniero de software reutiliza piezas que él sabe que se ajustan al problema.

Sistemática

El esfuerzo es a nivel organizacional y planificado de antemano, ya que todo componente ha de ser ideado para ser reutilizado con facilidad.

Bottom-Up

Se desarrollan pequeños componentes para una determinada aplicación y se incorporan a un repositorio.

Top-Down

Se determinan las piezas necesarias que encajan unas con otras, desarrollándolas poco a poco. Esto conlleva una alta inversión económica y de tiempo.

¿Basta con reutilizar código fuente?

Hasta hace poco tiempo, la respuesta a esta pregunta resultaba ser un rotundo **NO**, ya que a todos se nos había enseñado a escribir código directamente a partir de un documento que describía vagamente las necesidades de usuario. Hoy por hoy, sin embargo, reutilizar código fuente es realmente problemático debido a:

- Reutilizar código en antiguas plataformas de desarrollo (COBOL, FORTRAN, Lisp, C...) estaría desactualizado en estos tiempos.
- Quienes apostaron por reutilización basada en componentes COM, CORBA [JACO] se encuentran hoy día con que la tecnología actual nos orienta hacia los **Web Services**.

Almacén de modelos de software

El repositorio deberá permitir almacenar y gestionar modelos creados con herramientas CASE, sin olvidar funcionalidades como gestión de versiones y de accesos, gestión de configuración...

Localizador de modelos por su similitud

El repositorio de modelos debe ser lo suficientemente extenso para requerir técnicas de búsqueda, por lo que se hace vital disponer de un sistema avanzado de búsqueda que permita discriminar atributos, incluir la semántica...

Localizador y extractor automático de patrones

Los repositorios en los que se van introduciendo nuevos modelos al sistema, y donde se vayan creando automáticamente patrones de elementos interrelacionados que más se repiten en determinados modelos para formar parte de la familia de patrones de la organización, reutilizables por cualquiera de sus diseñadores.

Métricas

Como todos los procesos, el de la reutilización debe ser medido de forma apropiada, sobre todo para poder medir el retorno de inversión (ROI), aspecto vital para conseguir el apoyo de la alta dirección en los procesos de reutilización de software.

#7: ventanas modales

#7.1: modal.html

#7.2: modal_2.html

#7.1: modal.html

Observa la codificación completa para que muestre una ventana modal.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Ventana modal con HTML y Javascript</title>
    <link rel="stylesheet" href="style.css">
    <script src="script.js">
      function mostrarVentana(){
        // Accedemos al contenedor
        var ventana = document.getElementById('miVentana');

        // Definimos su posición vertical. La ponemos fija para simplificar el código
        ventana.style.marginTop = "100px";
        // Definimos su posición horizontal
        ventana.style.marginLeft = ((document.body.clientWidth-350) / 2) + "px";
        // Y lo hacemos visible
        ventana.style.display = 'block'; }

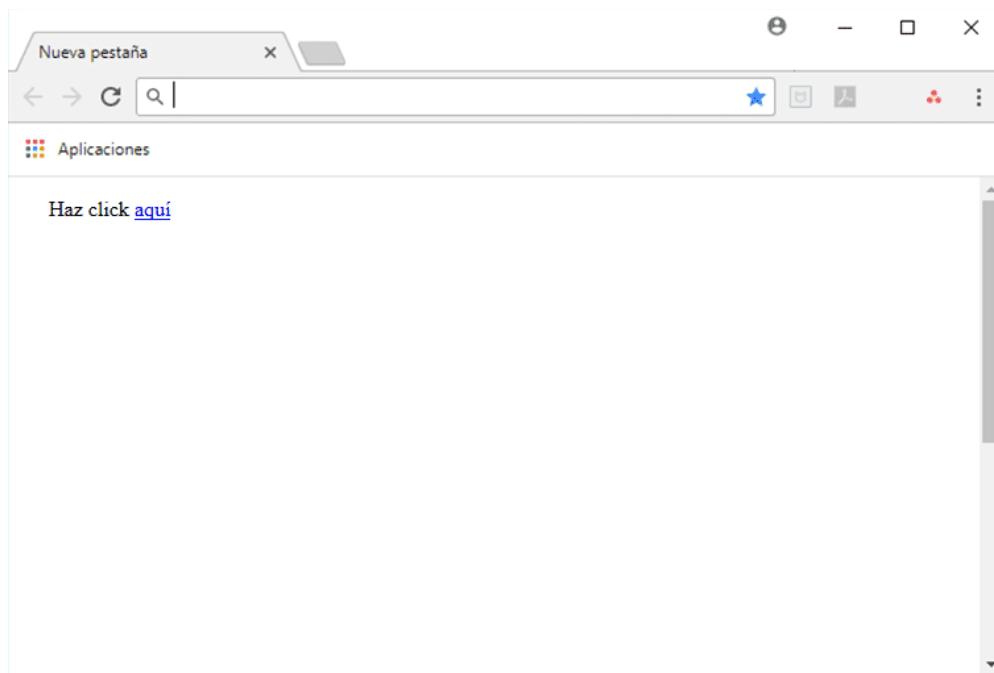
      function ocultarVentana(){
```

```
//Accedemos al contenedor
var ventana = document.getElementById('miVentana');
//Y lo hacemos visible
ventana.style.display = 'none'; // Y lo hacemos invisible}
</script>
</head>
<body>
    Haz click <a href="javascript:mostrarVentana();">aquí</a>
    <div id="miVentana" style="position: fixed; width: 350px; height: 190px; top: 0; left: 0; font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 12px; font-weight: normal; border: #333333 3px solid; background-color: #FAFAFA; color: #000000; display:none;">
        <div style="font-weight: bold; text-align: left; color: #FFFFFF; padding: 5px; background-color:#006394">Título de la ventana</div>
        <p style="padding: 5px; text-align: justify; line-height: normal">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum a est. Suspendisse vehicula, nisl vitae molestie pulvinar.</p>
        <div style="padding: 10px; background-color: #F0F0F0; text-align: center; margin-top: 44px;"><input id="btnAceptar" onclick="ocultarVentana();" name="btnAceptar" size="20" type="button" value="Aceptar" /></div></div>
</body>
</html>
```

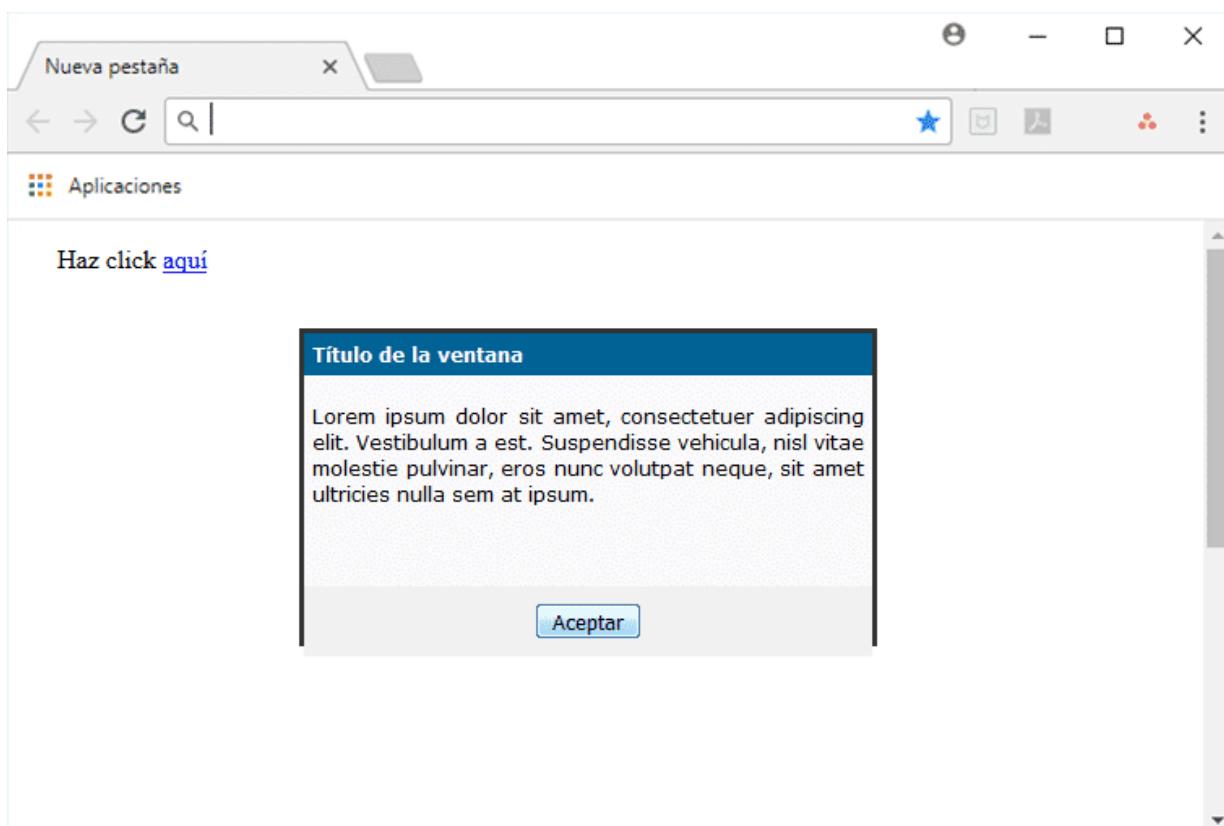
```
<head>
<title>Ventana modal con HTML y Javascript</title>
<script type="text/javascript">
function mostrarVentana()
{
    var ventana = document.getElementById('miVentana'); // Accedemos al contenedor
    ventana.style.marginTop = "100px"; // Definimos su posición vertical. La ponemos fija para simplificar el código
    ventana.style.marginLeft = ((document.body.clientWidth-350) / 2) + "px"; // Definimos su posición horizontal
    ventana.style.display = 'block'; // Y lo hacemos visible
}

function ocultarVentana()
{
    var ventana = document.getElementById('miVentana'); // Accedemos al contenedor
    ventana.style.display = 'none'; // Y lo hacemos invisible
}
</script>
</head>
<body>
Haz click <a href="javascript:mostrarVentana();">aquí</a>
<div id="miVentana" style="position: fixed; width: 350px; height: 190px; top: 0; left: 0; font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 12px; font-weight: normal; border: #333333 3px solid; background-color: #FAFAFA; color: #000000; display:none;">
    <div style="font-weight: bold; text-align: left; color: #FFFFFF; padding: 5px; background-color:#006394">Título de la ventana</div>
    <p style="padding: 5px; text-align: justify; line-height: normal">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum a est. Suspendisse vehicula, nisl vitae molestie pulvinar, eros nunc volutpat neque, sit amet ultricies nulla sem at ipsum.</p>
    <div style="padding: 10px; background-color: #F0F0F0; text-align: center; margin-top: 44px;"><input id="btnAceptar" onclick="ocultarVentana();" name="btnAceptar" size="20" type="button" value="Aceptar" /></div>
</div>
</body>
</html>
```

Codificación desarrollada en SublimeText.



Resultado mostrado en nuestro navegador (en este caso, Google Chrome).



Resultado en forma de ventana modal mostrada en nuestro navegador (en este caso, Google Chrome).

#7.2: modal_2.html

Ahora, veamos un poco de código CSS para que *acomodemos* nuestra ventana modal como más nos guste, es decir, variando cualquiera de las propiedades de que dispone.

En este caso, podríamos decir que existe una **reutilización** de código de otras ventanas modales, por lo que se modifica únicamente parte del código.

```
/* Modal (background) */
.modal {
    display: none; /* oculto por defecto */
    position: fixed;
    z-index: 1;
    padding-top: 100px;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    overflow: auto; /* Activación de scroll si es necesario */
    background-color: rgb(0,0,0);
    background-color: rgba(0,0,0,0.4);
}

/* Modal Content */
.modal-content {
    position: relative;
    background-color: #fefefe;
    margin: auto; /*para centrar horizontalmente: margin: 0 auto;*/
    padding: 0;
    border: 1px solid #888;
    width: 80%;
    box-shadow: 0 4px 8px 0 rgba(123,30,0,0.8),0 14px 18px 0
    rgba(0,0,0,0.7);
    -webkit-animation-name: animatetop;
    -webkit-animation-duration: 0.4s;
    animation-name: animatetop;
    animation-duration: 0.4s
}

/* Animación */
@-webkit-keyframes animatetop { from {top:-300px; opacity:0} to {top:0;
```

```
    opacity:1}
}

@keyframes animatetop {from {top:-300px; opacity:0} to {top:0; opacity:1}}

/* Botón cerrar */
.close {color: white; float: right; font-size: 28px; font-weight: bold;}
.close:hover,.close:focus {color: #000; text-decoration: none; cursor: pointer;}

.modal-header {padding: 2px 16px; background-color: #5cb85c; color: white;}
.modal-body {padding: 2px 16px;}
.modal-footer {padding: 2px 16px; background-color: #5cb85c; color: white;}
```

CSS para nuestra nueva ventana modal.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Ventana modal nueva HTML + JS</title>
    <link rel="stylesheet" href="style.css">
    <script src="script.js">
      function mostrarVentana(){
        // Accedemos al contenedor
        var ventana = document.getElementById('miVentana');

        // Definimos su posición vertical. La ponemos fija para
        // simplificar el código
        ventana.style.marginTop = "100px";
        // Definimos su posición horizontal
        ventana.style.marginLeft = ((document.body.clientWidth-350)
        / 2) + "px";
        // Y lo hacemos visible
        ventana.style.display = 'block'; }

      function ocultarVentana(){
        //Accedemos al contenedor
        var ventana = document.getElementById('miVentana');
        //Y lo hacemos visible
        ventana.style.display = 'none'; // Y lo hacemos invisible}
    </script>
```

```
</head>
<body>
    <h2>Ventana modal con header + footer</h2>
    <button id="myBtn">Abrir Modal</button>

    <div id="myModal" class="modal">
        <div class="modal-content">
            <div class="modal-header">
                <span class="close">x</span>
                <h2>Modal Header</h2>
            </div>
            <div class="modal-body">
                <p>Texto para el cuerpo de la ventana Modal</p>
                <p>Texto variado...</p>
            </div>
            <div class="modal-footer"><h3>Modal Footer</h3></div>
        </div></div>

<script>
// Adquirimos la ventana modal
var modal = document.getElementById('myModal');

// Adquirimos el botón que activa la ventana modal
var btn = document.getElementById("myBtn");

// Adquirimos el <span> que cierra la ventana modal
var span = document.getElementsByClassName("close")[0];

// Cuando el usuario pincha en el botón, se abre la ventana modal
//elemento.método (una función)
btn.onclick = function() {modal.style.display = "block";}

// Cuando el usuario pincha en el botón x(span), se cierra la ventana modal
span.onclick = function() {modal.style.display = "none";}

// Si el usuario pincha fuera de la ventana modal, cierra la misma
window.onclick = function(event) {if (event.target == modal) {modal.style.display = "none";}}
</script>
</body>
</html>
```

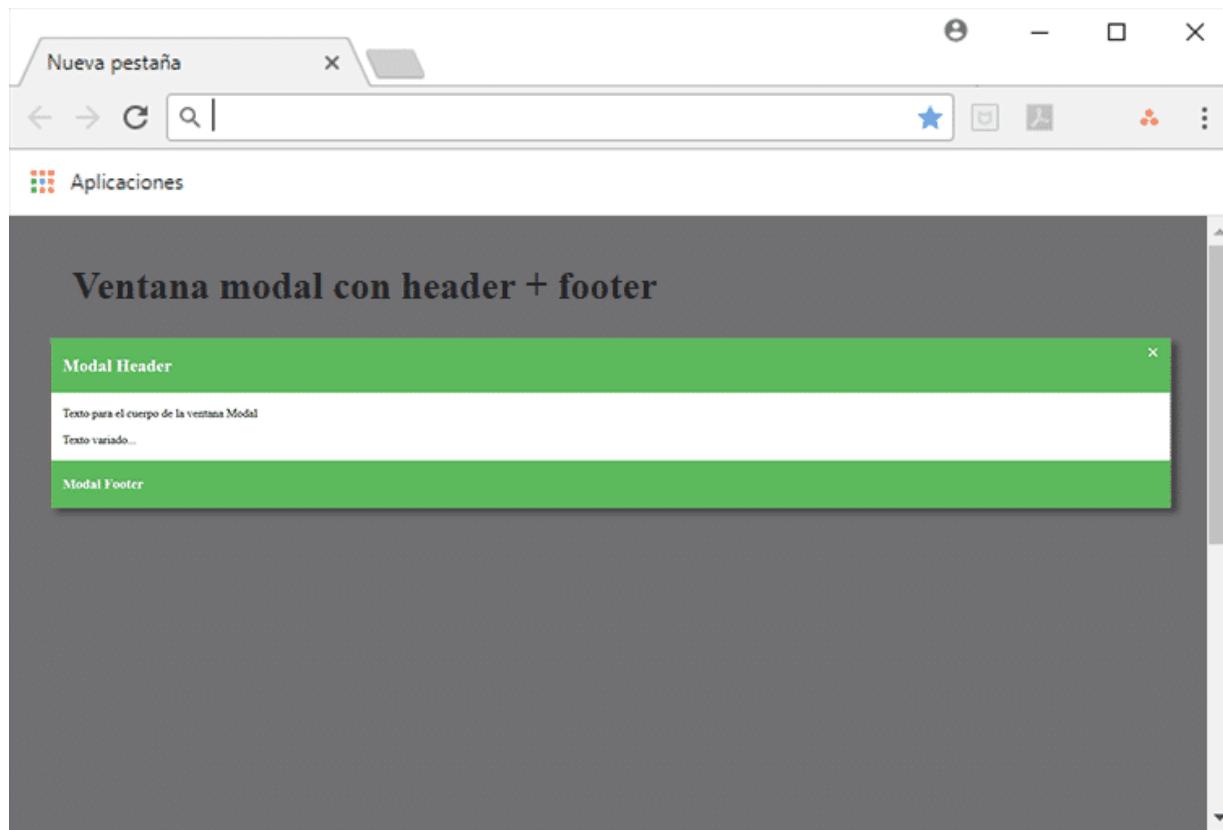
Codificación completa para que muestre una ventana modal.

```
22     <p>Texto variado...</p>
23   </div><!-- modal-body -->
24   <div class="modal-footer">
25     <h3>Modal Footer</h3>
26   </div><!-- modal-footer -->
27 </div>
28 </div>
29
30 <script>
31 // Adquirimos la ventana modal
32 var modal = document.getElementById('myModal');
33
34 // Adquirimos el botón que activa la ventana modal
35 var btn = document.getElementById("myBtn");
36
37 // Adquirimos el <span> que cierra la ventana modal
38 var span = document.getElementsByClassName("close")[0];
39
40 // Cuando el usuario pincha en el botón, se abre la ventana modal
41 //elemento.método (una función)
42 btn.onclick = function() {modal.style.display = "block";}
43
44 // Cuando el usuario pincha en el botón x(span), se cierra la ventana modal
45 span.onclick = function() {modal.style.display = "none";}
46
47 // Si el usuario pincha fuera de la ventana modal, cierra la misma
48 window.onclick = function(event) {
49   if (event.target == modal) {modal.style.display = "none";}
50 }
51 </script>
52
53 </body>
```

Codificación desarrollada en SublimeText.



Resultado mostrado en nuestro navegador (en este caso, Google Chrome, sin ejecutar aún la ventana modal).



Resultado en forma de ventana modal mostrada en nuestro navegador (en este caso, Google Chrome).



Cierre de la ventana modal.

POO en JS

Y ahora vamos a conocer la programación orientada a objetos con JavaScript.

Código espagueti: los *menos* entendidos en el lenguaje de programación JS lo utilizan como metáfora crítica al uso de este lenguaje. Con esta expresión se refieren, principalmente, a un código sin estructura, formado por línea tras línea de código en una sola fila.

En la programación por procedimientos, las funciones se usan para llevar a cabo tareas. Necesitamos funciones, pero también necesitamos un diseño con el que podamos trabajar.

La **programación orientada a objetos** enfoca la creación de esos **objetos**.

Entonces, ¿cómo podemos traducir todos los requisitos a objetos? Una técnica es crear una lista de sustantivos en base a nuestra descripción y después refinrar la lista para obtener solamente aquéllos que son relevantes para el problema.

De hecho, cuando creamos objetos necesitamos **clases**, o lo que es lo mismo, una *plantilla tipo* para crear objetos.



Por convención, el nombre de la clase se escribe en mayúsculas.

El constructor es una función especial que declara e *inicializa* los atributos de datos. Dentro de la función del constructor, los atributos se agregan usando la palabra clave *this*. Después, se listan los métodos de la clase sin separadores.

```
class NombreClase {  
    constructor(...argumentos){  
        this.atributo = argumento1;  
        this.atributo2 = argumento2;  
        ...}  
    metodo_uno(){...}  
    metodo_dos(){...}  
}
```

Pseudocódigo para poder crear una clase en JS

Veamos un **ejemplo** práctico con la creación de una clase típica: *libro*.

```
class Book {  
    constructor(titulo, autor, isbn){  
        this.titulo = titulo ;  
        this.autor = autor;  
        this.isbn = isbn;  
    }  
  
    getTitulo(){return this.titulo;}  
    setTitulo(newTitulo){this.titulo= newtitulo ;}  
  
    getAutor(){return this.autor;}  
    setAutor(newAutor){this.autor = newAutor;}  
  
    getIsbn(){return this.isbn;}  
    setIsbn(newIsbn){this.isbn = newIsbn;}  
}
```

La clase por sí sola es inútil para nosotros, a menos que hagamos algo con ella. Los objetos deben ser **instanciados** con el operador **new**. Los datos que se envían al objeto son los parámetros que definimos en nuestro constructor.

El objeto creado a partir de la clase se conoce como **instancia** y se crea de la siguiente forma:

```
variableNombre = new ClaseNombre(..argumentos);
```

Creación de la clase instancia.

Resumen

Has terminado la lección, repasemos los puntos más importantes que hemos tratado.

- Un **modelo basado en componentes** es un proceso que concede particular importancia al diseño y la construcción de sistemas basados en ordenadores que utilizan componentes de software reutilizable, dando paso a la integración de software como centro de enfoque de las nuevas tecnologías.
- Si los requisitos no se pueden implementar con *CDL* o *desarrollo propio*, se aplicarían métodos de software para la construcción de nuevos componentes.
- Desarrollo basado en componentes: aproximadamente el **60%** del diseño y del código de aplicaciones es reutilizable, y un **75%** de las funciones son comunes a más de un programa. Sólo el **15%** del código encontrado en muchos sistemas es único y novedoso.
- Etapas del desarrollo **CBD (Component Based Development)**: componentes monolíticos, componentes distribuidos y persistencia de componentes.
- La **reutilización de software** aparece como una alternativa para desarrollar aplicaciones y sistemas de un manera más *eficiente, productiva y rápida*.



PROEDUCA