

PROGRAMACIÓN II



SEMANA 12 - 1
EXCEPCIONES

Eric Gustavo Coronel Castillo
ecoronel@uch.edu.pe
gcoronelc.blogspot.com

Excepciones

Uno de los mayores problemas en la programación es el tratamiento de errores, que pueden ser generados por:

- Fallas o limitaciones del hardware (por ejemplo errores de lectura de archivos)
- Fallas en el software (casos en los cuales no se cumple una determinada condición).

Para facilitar el tratamiento de errores en PHP se ha creado el concepto de Excepciones, el cual se refiere a una situación de error en la ejecución de un programa, cada vez que ocurre una excepción (un error) el programa debe de tratarla, normalmente mostrando un mensaje de error y ejecutando alguna rutina de tratamiento de errores.

Índice

1	TIPOS DE ERRORES	4
1.1	ERRORES DE SINTAXIS.....	4
1.2	ERRORES DE LÓGICA.....	5
1.3	ERRORES DE EJECUCIÓN.....	7
2	¿QUÉ ES UNA EXCEPCIÓN?	9
3	GESTIÓN DE EXCEPCIONES	11
3.1	PALABRAS CLAVES	11
3.2	ESQUEMA GENERAL.....	11
3.3	GENERAR EXCEPCIONES.....	12
3.4	PROPAGACIÓN DE EXCEPCIONES.....	12
4	CREACIÓN DE UN LOG DE ERRORES	13
5	PROYECTO EJEMPLO	15
5.1	REQUERIMIENTO DE SOFTWARE	15
5.2	ABSTRACCIÓN	15
5.3	ESTRUCTURA DEL PROYECTO EN NETBEANS	18
5.4	CODIFICACIÓN.....	18
5.4.1	Capa: View	18
5.4.2	Capa: Controller	23
5.4.3	Capa: Model.....	25
5.4.4	Capa: DAO.....	26
5.5	EJECUCIÓN DEL PROYECTO	27
5.5.1	Proceso de Logueo	27
5.5.2	Interfaz de Usuario Principal	28
5.5.3	Consultar el Estado de una Cuenta.....	28
5.5.4	Consultar Movimiento de una Cuenta.....	29

1 Tipos de Errores

1.1 Errores de Sintaxis

Los errores en la sintaxis son causados cuando el compilador de PHP no puede reconocer una instrucción. Esto causa que el compilador devuelva un mensaje de error, usualmente con una línea de código de referencia.

También se conoce a los errores de sintaxis como errores en tiempo de compilación.

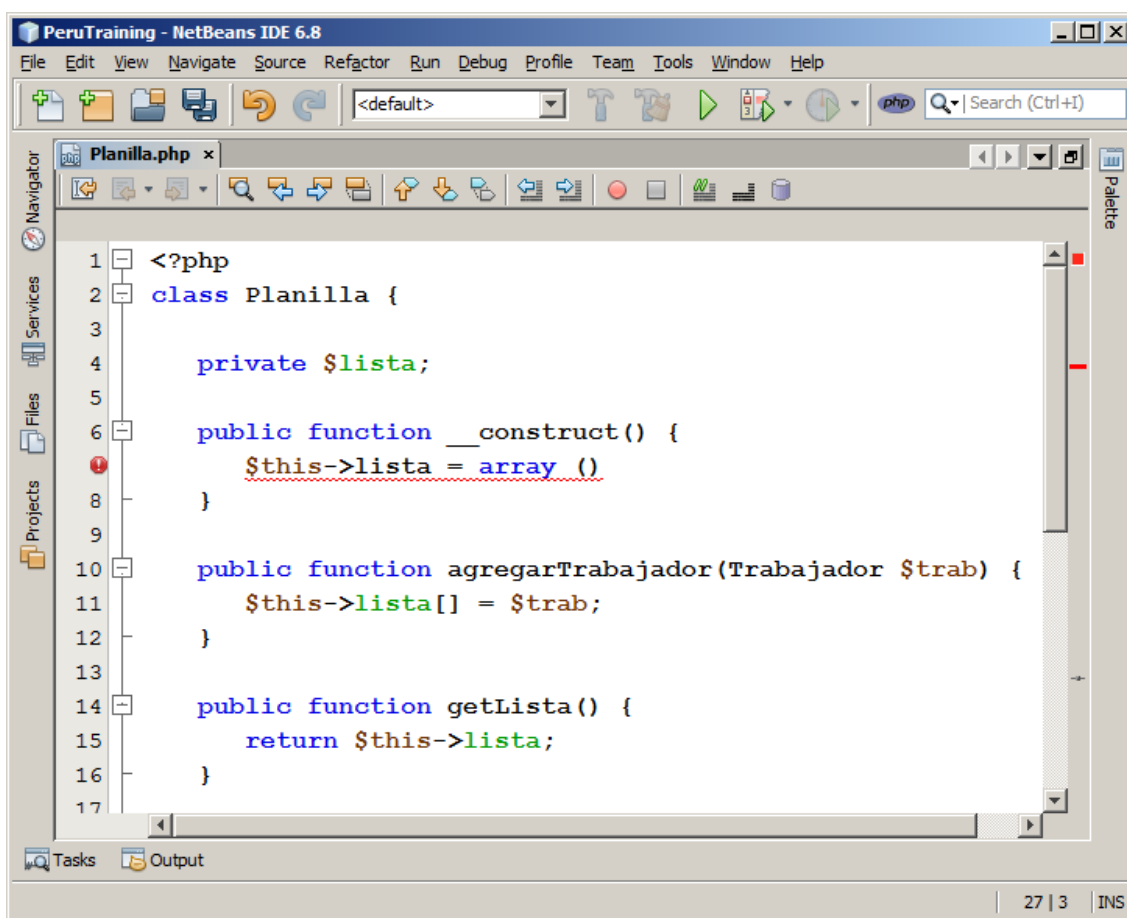


Figura 1 Error de sintaxis.

Cuando trabajamos con un IDE como NetBeans es fácil darnos cuenta de los errores de sintaxis, en la Figura 1 podemos notar que en la línea 7 hace falta un punto y coma.

Ejemplo 1

En el presente ejemplo veremos el mensaje que se muestra cuando existen errores de sintaxis:

Archivo: \Aprendizaje\ejemplo01.php

```
<?php  
  
$n1 = rand(1,20);  
$n2 = rand(1,20)  
$suma = $n1 + $n2;  
  
echo("$n1 + $n2 = $suma");  
  
?>
```

Cuando ejecutamos este programa ejemplo obtenemos el siguiente mensaje:

**Parse error: parse error in C:\PHP100\cap14\Aprendizaje
\ejemplo01.php on line 5**

Al revisar el código podemos constatar que al finalizar la línea 4 falta un punto y coma (;).

1.2 Errores de Lógica

Los errores de lógica son conocidos como BUGS. Estos son los errores que nos tomarán un tiempo hasta encontrarlos. Si damos a elegir, cualquier programador elegiría los errores en tiempo de compilación.

Para encontrar un error de lógica debemos identificar la clase donde podría estar el error, y si es posible el métodos o posibles métodos donde podría estar el error, luego de eso hacer una depuración (Debug) línea por línea (trace) en cada uno de los métodos.

Ejemplo 2

En el presente ejemplo veremos como un error de lógica puede alterar totalmente el resultado esperado.

Archivo: Aprendizaje\ejemplo02.php

```
<?php  
  
$n = rand(3,8);  
  
$f = 1;  
for( $k = 1 ; $k <= $n; $k++ );  
{  
    $f *= $n;  
}  
  
echo("Factorial de $n es $f");  
?>
```

El programa encuentra el factorial de un número aleatorio que se encuentra entre 3 y 8, no existe ningún error de sintaxis y aparentemente tampoco existe ningún error de lógica, pero cuando ejecutamos el programa podemos obtener un resultado como el siguiente:

Factorial de 6 es 6

El error está en la siguiente instrucción:

```
for( $k = 1 ; $k <= $n; $k++ );
```

No debe ir punto y coma al final de esta línea, porque estaría terminando la instrucción `for`, por lo tanto se trata de un bucle vacío.

1.3 Errores de Ejecución

Los errores de ejecución se producen cuando la aplicación esta en producción o prueba (Testing), y aparecen por una situación anormal durante la ejecución de alguna instrucción.

Por ejemplo:

- Una división por cero.
- No se tiene permiso de escritura sobre un archivo.
- La base de datos no existe.
- No se tiene permiso de acceso a una base de datos.
- Etc.

Ejemplo 3

En este ejemplo veremos el mensaje que muestra PHP cuando se produce un error fatal.

Archivo: Aprendizaje\ejemplo03.php

```
<?php  
  
$cn = mysql_connect("localhost", "root", "admin");  
echo 'Conexión ok.';  
  
?>
```

Cuando ejecutamos este programa, su ejecución se ve interrumpida en la línea 3 por un **Fatal Error** y se muestra el siguiente mensaje:

Fatal error: Call to undefined function `mysql_connect()` in
`C:\PHP100\cap14\Aprendizaje\ejemplo04.php` on line 3

Para interpretar correctamente el error primero debemos analizar el mensaje textualmente, nos dice que no reconoce la función `mysql_connect`, pero si estamos seguros que la función está bien escrita, debemos pensar que la librería de MySQL no se ha cargado, para lo cual debemos revisar el archivo **php.ini**.

Para la prueba de este error se ha deshabilitado la librería de MySQL en el archivo **php.ini**.

Ejemplo 4

En este ejemplo veremos otro tipo de error.

Archivo: Aprendizaje\ejemplo04.php

```
<?php  
  
$n1 = rand(0,20);  
$n2 = rand(0,20);  
  
$d = $n1 / $n2;  
  
echo("$n1 / $n2 = $d");  
  
?>
```

Cuando ejecutamos este programa, y para $n2$ se genera un valor 0 tenemos una operación inválida en la línea 6 la cual se manifiesta con un mensaje de error de tipo **Warning**, pero la ejecución continúa con las siguientes líneas. El resultado se muestra a continuación:

Warning: Division by zero in C:\PHP100\cap14\Aprendizaje
ejemplo04.php on line 6
12 / 0 =

Nota:

Para que los mensajes de error se muestren en pantalla se debe habilitar el parámetro de configuración `DISPLAY_ERRORS`, esto es solo recomendable cuando estamos en la etapa de desarrollo.

Si queremos evitar que se muestre el mensaje de error cuando `DISPLAY_ERRORS` está habilitado podemos usar el carácter arroba (@) al inicio de la instrucción.

Para el ejemplo anterior el error se produce en la siguiente instrucción:

```
$d = $n1 / $n2;
```

Para deshabilitar que muestre el error cuando $n2$ tiene valor 0 sería así:

```
@$d = $n1 / $n2;
```

2 ¿Qué es una Excepción?

Una excepción es un objeto que describe una condición excepcional, es decir, un error que se ha dado en una parte del código.

Cuando se origina un error se produce una condición de excepción, se crea un objeto que representa esa excepción y se lanza al método que ha causado el error. Este método puede elegir entre gestionar él mismo la excepción ó pasarla al método que lo ha invocado. De cualquiera de las dos formas, en un punto determinado se capturará la excepción y se procesará.

Las excepciones deben ser generadas en forma manual por el propio código.

Las excepciones generadas de forma manual se utilizan generalmente para informar acerca de alguna condición de error personalizada, por ejemplo, un error en el proceso de la lógica del negocio.

Ejemplo 5

En este ejemplo pretendo ilustrar el concepto de excepción.

Archivo: Aprendizaje\ejemplo04.php

```
<?php
try {
    // Generación de datos
    $n1 = rand(0,20);
    $n2 = rand(0,20);
    // Validación de datos
    if( $n1 == 0 ) {
        throw new Exception("No hay nada que dividir");
    }
    if( $n2 == 0 ) {
        throw new Exception("No se puede dividir por cero.");
    }
    // Proceso
    $d = $n1 / $n2;
    // Reporte
    echo("$n1 / $n2 = $d");
} catch(Exception $e) {
    echo "Error: {$e->getMessage()}";
}
?>
```


En este ejemplo en la sección **Validación de datos** verificamos los valores generados, si alguno de ellos es 0 (cero) se genera una excepción con la sentencia `throw`, en la sección `catch` se captura la excepción y se muestra el mensaje respectivo.

Si el valor de `$n1` es 0 se tiene el siguiente resultado:

Error: No hay nada que dividir

Si el valor de `$n2` es 0 se tiene el siguiente resultado:

Error: No se puede dividir por cero.

Si los valores de `$n1` y `$n2` son diferentes de 0 el resultado tiene el siguiente formato:

`20 / 2 = 10`

3 Gestión de Excepciones

3.1 Palabras Claves

La gestión de excepciones se realiza a través de tres palabras claves: `try`, `catch` y `throw`.

Palabra Clave	Descripción
<code>try</code>	Dentro del bloque <code>try</code> se deben incluir las sentencias que se quieren controlar, en otras palabras, las que podrían desencadenar un error.
<code>catch</code>	El bloque <code>catch</code> captura las excepciones que se desencadenan en el bloque <code>try</code> . En este bloque se procesa la excepción.
<code>throw</code>	Para generar excepciones en forma manual debemos utilizar una instrucción <code>throw</code> .

3.2 Esquema General

El esquema general para el manejo de excepciones es el siguiente:

```
try {  
    // Sentencias a controlar  
} catch ( Exception $e ) {  
    // Control de excepción  
}
```

En el bloque `try` debemos hacer las validaciones del caso y generar la excepción respectiva cuando se encuentre un error, una aplicación lo tenemos en el ejemplo 5.

3.3 Generar Excepciones

- **Caso 1**

En este caso primero creamos el objeto de tipo `Exception` y luego lo lanzamos con la instrucción `throw`.

```
$e = new Exception( "Mensaje" );  
throw $e;
```

- **Caso 2**

En este otro caso creamos y lanzamos el objeto `Exception` en la misma instrucción `throw`.

```
throw new Exception( "Mensaje" );
```

3.4 Propagación de Excepciones

En muchos casos necesitamos que la excepción detectada en una función se propague a otros niveles según la pila de llamadas, esto se logra con la instrucción `throw`, el esquema es el siguiente:

```
public function método( .. ) {  
    try {  
        // Sentencias a controlar  
    } catch ( Exception $e ) {  
        throw $e;  
    }  
};
```

4 Creación de un Log de Errores

Para administrar de manera efectiva una aplicación, es necesario tener un registro de cualquier problema que haya podido ocurrir durante su operación, es por eso que se hace imprescindible contar con un log de errores.

Un log de errores, es un archivo que almacena los errores que se han producido durante el funcionamiento de la aplicación, añadir un log de errores a nuestra solución nos permitirá controlar cuando se ha producido un error para corregirlo y evitar que se repita en el futuro.

Para manejar un log de errores debemos utilizar la función **error_log**, su sintaxis es la siguiente:

```
error_log ( string mensaje [, int tipo_mensaje [, string destino  
[, string cabeceras_extra]]] )
```

El significado de sus parámetros es el siguiente:

- **mensaje:**

Se refiere al mensaje de error que se quiere registrar.

- **tipo_mensaje:**

Indica a dónde debe ir el mensaje. Los tipos de mensaje posibles son los siguientes:

Tipo	Descripción
0	Indica que mensaje es enviado al registro de sistema de PHP, usando el mecanismo de registro del Sistema Operativo o un archivo, dependiendo del valor de la directiva de configuración error_log . Esta es la opción predeterminada.
1	Indica que mensaje es enviado por correo electrónico a la dirección en el parámetro destino . Este es el único tipo de mensaje en donde el cuarto parámetro, cabeceras_extra , es usado.
2	Ya no se usa.
3	Indica que mensaje es agregado al final del archivo destino . Un salto de línea no es agregado automáticamente al final de la cadena mensaje.
4	Indica que el mensaje es enviado directamente al controlador SAPI logging.

- **destino:**

Indica el destino del mensaje. Su significado depende del parámetro `tipo_mensaje` como se describió anteriormente.

- **cabeceras_extra:**

Este parámetro solo es usado cuando el parámetro `tipo_mensaje` es definido en 1. Este tipo de mensaje usa la misma función interna que usa `mail()`.

En caso que quisiéramos enviar el mensaje a un archivo, la instrucción tiene el siguiente formato:

```
error_log ( "mensaje", 3, "archivo" ) ;
```

Aquí tenemos un ejemplo:

```
error_log ( "Mensaje de Prueba", 3, "eureka.log" ) ;
```

5 Proyecto Ejemplo

5.1 Requerimiento de Software

EurekaBank es una institución financiera y está requiriendo una aplicación que permita a su personal de atención al cliente brindar información sobre sus cuentas.

La información que puede brindar es con respecto a lo siguiente:

- Estado de una cuenta
- Saldo de una cuenta
- Últimos 10 movimientos de una cuenta

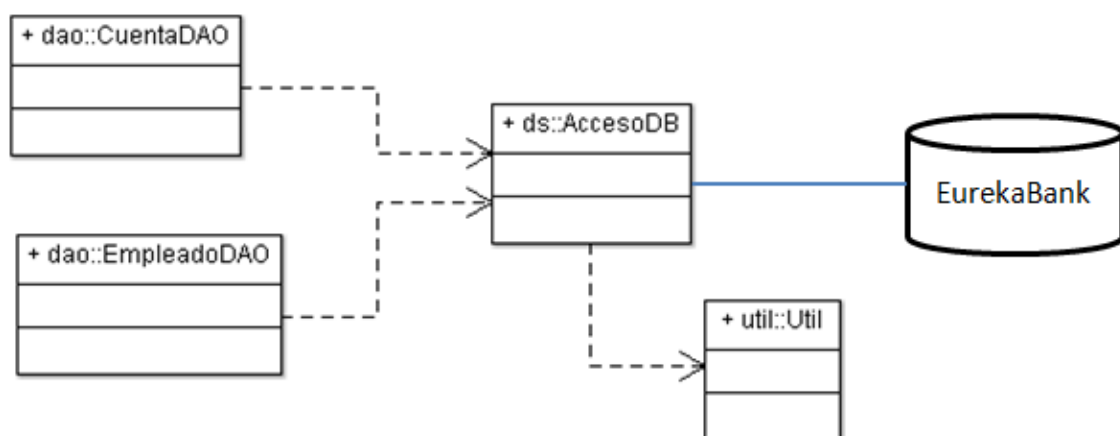
Es necesario que la aplicación cuente con una opción de inicio de sesión antes de poder hacer cualquier consulta.

La base de datos utilizada para este proyecto es **EurekaBank**.

5.2 Abstracción

▪ Acceso a la base de datos

El acceso a la base de datos será utilizando el patrón de diseño DAO, a continuación tenemos el diagrama de clases:



- Clase: Util

Esta clase proporciona un método para grabar errores en el log de la aplicación.

- Clase: AccesoDB

Esta clase representa la conexión con la base de datos y proporciona los métodos necesarios para ejecutar sentencias SQL.

El acceso a una instancia de esta clase está controlado mediante el patrón de diseño Singleton.

- **Clase: EmpleadoDAO**

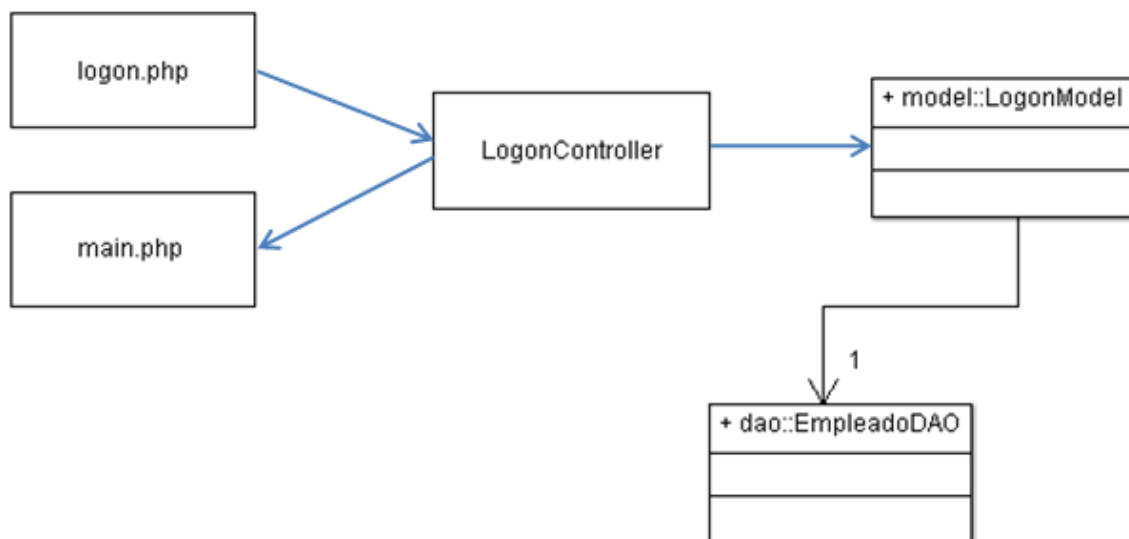
Esta clase proporciona los métodos necesarios para operar sobre la tabla de empleados.

- **Clase: CuentaDAO**

- Esta clase proporciona los métodos necesarios para operar sobre las cuentas de los clientes.

- **Inicio de Sesión**

Para el inicio de sesión se está utilizando el patrón de diseño MVC, tal como se ilustra a continuación:



- **Programa: logon.php**

En este programa tenemos el formulario que permite al empleado ingresar su nombre de usuario y clave para que pueda ingresar al sistema.

- **Programa: main.php**

En este programa tenemos el menú con las opciones de la aplicación, actúa como un centro de control.

- **Programa: LogonController**

Se trata del controlador para el proceso logueo, para este caso debido a que atenderá un solo requerimiento se trata de un programa y no de una clase.

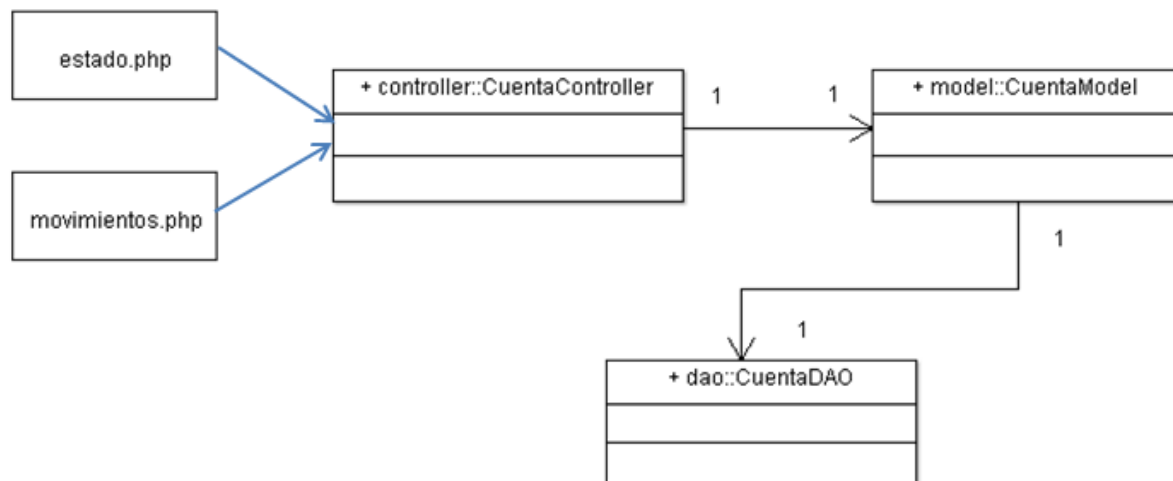
- **Clase: LogonModel**

En esta clase tenemos la programación del método **validar** que se encarga de comunicarse con la capa DAO (EmpleadoDAO) para hacer la validación del **usuario** y la **clave** ingresados por el empleado a través del browser.

- **Procesos de la Aplicación**

Los procesos que debe ejecutar la aplicación son básicamente 2, el primero de ellos es consultar los datos de una cuenta y el segundo consultar sus movimientos.

Para el desarrollo de estos procesos también se está utilizando el patrón de diseño MVC, tal como se ilustra a continuación:



- **Programa: estado.php**

Se trata de un programa que muestra el formulario para que el empleado ingrese el número de cuenta y muestra sus datos, como su saldo y su estado (activa o cancelada).

- **Programa: movimientos.php**

Se trata de un programa que muestra el formulario para que el empleado ingrese el número de cuenta y muestra sus movimientos.

- **Clase: CuentaController**

Se trata del controlador para los procesos relacionados con las cuentas, atiende los requerimientos de los programas estado.php y movimientos.php.

- **Clase: CuentaModel**

Es la clase que resuelve los requerimientos relacionados con las cuentas. Se encarga de comunicarse con la clase **CuentaDAO** para los servicios de base de datos.

5.3 Estructura del Proyecto en NetBeans

Como se puede apreciar en el grafico de la derecha, tenemos todos los componentes organizados en carpetas

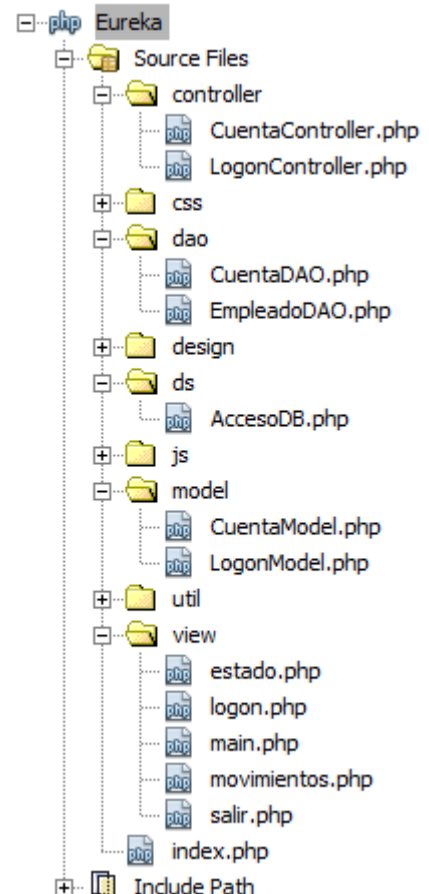
En la carpeta **view** tenemos los programas que generan el HTML que se envía al navegador.

En la carpeta **controller** tenemos los componentes encargados del control de los requerimientos.

Las clases que se encuentran en la carpeta **model** se encarga de resolver los requerimientos de los usuarios.

Las clases que se encuentran en la carpeta **dao** proveen los servicios de base de datos.

En la raíz del proyecto tenemos el programa **index.php**, éste es el que inicia la ejecución del proyecto, y en este caso hace un redireccionamiento al programa **logon.php** que se encuentra en la carpeta **view**



5.4 Codificación

5.4.1 Capa: View

Archivo: Eureka\view\logon.php

```
<?php
session_start();
require_once '../util/Session.php';
$error = Session::getAttribute2("error");
$usuario = Session::getAttribute2("usuario");

?>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" type="text/css" href="../css/estilos.css">
    <title></title>
  </head>
  <body>
    <h1>EurekaBank</h1>
    <h2>Ingreso al Sistema</h2>
    <form method="post" action="../controller/LogonController.php">
      <table width="273" cellspacing="0">
        <tr>
          <td width="53">Usuario</td>
          <td width="110">
            <input name="usuario" type="text" class="CampoEdicion"
              id="usuario" size="15" maxlength="15"
              value="<?php echo $usuario; ?>">
          </td>
          <td width="102"></td>
        </tr>
        <tr>
          <td>Clave</td>
          <td>
            <input name="clave" type="password" class="CampoEdicion"
              id="clave" size="15" maxlength="15">
          </td>
          <td>
            <input name="btnIngresar" type="submit" class="Boton"
              id="btnIngresar" value="Ingresar">
          </td>
        </tr>
      </table>
    </form>
    <p class="mensajeError"><?php echo($error); ?></p>
  </body>
</html>
```

Archivo: Eureka\view\main.php

```
<?php

session_start();
require_once '../util/Session.php';

// Controla el inicio de sesión
if( Session::NoExistsAttribute("empleado") ) {
  header("location: logon.php");
}
```

```
$emp = Session::getAttribute("empleado");

?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" type="text/css" href="../css/estilos.css">
    <script type="text/javascript" src="../js/main.js"></script>
    <title></title>
  </head>
  <body onLoad="initPage()">
    <h1>EurekaBank</h1>
    <p>Usuario: <?php echo $emp["vch_emplusuario"]; ?></p>
    <table width="500 px" border="2" cellspacing="0">
      <tr class="Menu01">
        <td width="165">
          <a href="javascript: cargarPagina('estado.php')">Estado Cuenta</a>
        </td>
        <td width="165">
          <a href="javascript: cargarPagina('movimientos.php')">Movimientos</a>
        </td>
        <td width="156">
          <a href="salir.php">Salir</a>
        </td>
      </tr>
    </table>
    <p>
      <iframe width="500 px" height="300 px" id="work"></iframe>
    </p>
  </body>
</html>
```

Archivo: Eureka\view\estado.php

En el campo oculto de nombre **action** se indica el nombre del que se debe ejecutar de la clase controladora que se especifica en el atributo **action** del formulario.

```
<?php

session_start();
require_once '../util/Session.php';

$cuenta = Session::getAttribute2("cuenta");
$error = Session::getAttribute2("error");

?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
```

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <link rel="stylesheet" type="text/css" href="../css/estilos.css">
  <title></title>
</head>
<body>
  <h2>Estado de una Cuenta</h2>
  <form name="form1" method="post" action="../controller/CuentaController.php">
    <input type="hidden" name="action" id="action" value="consultarCuenta"/>
    <label for="cuenta">Nro. Cuenta</label>
    <input name="cuenta" type="text" class="CampoEdicion" id="cuenta"
      size="8" maxlength="8">
    <input name="btnConsultar" type="submit" class="Boton"
      id="btnConsultar" value="Consultar">
  </form>
  <?php if( $cuenta ) { ?>
  <div>
    <h3>Resultado</h3>
    <table width="185" border="1" cellspacing="0">
      <tr>
        <td width="63">Cuenta</td>
        <td width="116"><?php echo $cuenta["chr_cuencodigo"]; ?></td>
      </tr>
      <tr>
        <td>Estado</td>
        <td><?php echo $cuenta["vch_cuenestado"]; ?></td>
      </tr>
      <tr>
        <td>Saldo</td>
        <td><?php echo $cuenta["dec_cuensaldo"]; ?></td>
      </tr>
    </table>
  </div>
  <?php } ?>
  <p class="MensajeError"><?php echo $error; ?></p>
</body>
</html>
```

Archivo: Eureka\view\movimientos.php

En el campo oculto de nombre **action** se indica el nombre del que se debe ejecutar de la clase controladora que se especifica en el atributo **action** del formulario.

```
<?php

session_start();
require_once '../util/Session.php';

$lista = Session::getAttribute2("movimientos");
```

```
$error = Session::getAttribute2("error");

?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" type="text/css" href="../css/estilos.css">
    <title></title>
  </head>
  <body>
    <h2>Movimientos de una Cuenta</h2>
    <form name="form1" method="post" action="../controller/CuentaController.php">
      <input type="hidden" name="action" id="action"
        value="consultarMovimientos"/>
      <label for="cuenta">Nro. Cuenta</label>
      <input name="cuenta" type="text" class="CampoEdicion" id="cuenta"
        size="8" maxlength="8">
      <input name="btnConsultar" type="submit" class="Boton"
        id="btnConsultar" value="Consultar">
    </form>
    <?php if( $lista ) { ?>
    <div>
      <h3>Resultado</h3>
      <table width="333" border="1" cellspacing="0">
        <tr class="TablaTitulo">
          <td width="68">Nro. Mov.</td>
          <td width="95">Fecha</td>
          <td width="66">Tipo</td>
          <td width="86">Importe</td>
        </tr>
        <?php foreach( $lista as $rec ) { ?>
        <tr class="TablaDato">
          <td><?php echo $rec["int_movinumero"]; ?></td>
          <td><?php echo $rec["dtm_movifecha"]; ?></td>
          <td><?php echo $rec["chr_tipocodigo"]; ?></td>
          <td><?php echo $rec["dec_moviimporte"]; ?></td>
        </tr>
        <?php } ?>
      </table>
    </div>
    <?php } ?>
  </body>
</html>
```

Archivo: Eureka\view\salir.php

```
<?php
session_start();
```

```
session_unset();
session_destroy();

header("location: logon.php");
?>
```

5.4.2 Capa: Controller

Archivo: Eureka\controller\LogonController.php

```
<?php

// Este controlador solo atiende un requerimiento.
// El requerimiento que atiende es el de inicio de sesión.

session_start();
require_once '../model/LogonModel.php';
require_once '../util/Session.php';

try {
    // Datos
    $usuario = $_REQUEST["usuario"];
    $clave = $_REQUEST["clave"];
    // Proceso
    $model = new LogonModel();
    $recEmpl = $model->validar($usuario, $clave);
    Session::setAttribute("empleado", $recEmpl);
    $target = "../view/main.php";
} catch (Exception $e) {
    Session::setAttribute("error", $e->getMessage());
    Session::setAttribute("usuario", $usuario);
    $target = "../view/logon.php";
}
header("location: $target");
?>
```

Archivo: Eureka\controller\CuentaController.php

```
<?php

session_start();
require_once '../model/CuentaModel.php';
require_once '../util/Session.php';

/*
```

```
* En la mayoría de casos un controlador atiende
* varios requerimientos desde diferentes formularios.
*
* Este controlador atenderá dos tipos de requerimientos:
* - El primero será sobre los datos de una cuenta
* - El segundo será sobre los movimientos de una cuenta
*
* El formulario debe enviar un parámetro de nombre action
* con el nombre del método a ejecutar.
*
* El controlador no debe acceder directamente a la capa DAO,
* sino que debe hacerlo a través de la capa MODEL.
*/
```

```
$action = $_REQUEST["action"];
$controller = new CuentaController();
$target = call_user_func(array($controller,$action));
header("location: $target");
return;

class CuentaController {

    public function consultarCuenta(){
        try {
            $cuenta = $_REQUEST["cuenta"];
            $model = new CuentaModel();
            $recCuenta = $model->consultarCuenta($cuenta);
            Session::setAttribute("cuenta", $recCuenta);
        } catch (Exception $e) {
            Session::setAttribute("error", $e->getMessage());
        }
        return "../view/estado.php";
    }

    public function consultarMovimientos(){
        try {
            $cuenta = $_REQUEST["cuenta"];
            $model = new CuentaModel();
            $movi = $model->consultarMovimientos($cuenta);
            Session::setAttribute("movimientos", $movi);
        } catch (Exception $e) {
            Session::setAttribute("error", $e->getMessage());
        }
        return "../view/movimientos.php";
    }
}
?>
```

5.4.3 Capa: Model

Archivo: Eureka\model\LogonModel.php

```
<?php
require_once '../dao/EmpleadoDAO.php';

class LogonModel {

    // Valida los datos del empleado.
    // Retorna el registro del empleado como un arreglo asociativo
    public function validar( $usuario, $clave ){
        try {
            $dao = new EmpleadoDAO();
            $rec = $dao->consultarPorUsuario($usuario);
            if($rec == null){
                throw new Exception("Usuario no existe.");
            }
            if($rec["vch_emplclave"] != $clave){
                throw new Exception("Clave incorrecta.");
            }
            return $rec;
        } catch (Exception $e) {
            throw $e;
        }
    }
}

?>
```

Archivo: Eureka\model\CuentaModel.php

```
<?php
require_once '../dao/CuentaDAO.php';

class CuentaModel {

    public function consultarCuenta($cuenta) {
        try {
            $dao = new CuentaDAO();
            $rec = $dao->consultarCuenta($cuenta);
            return $rec;
        } catch (Exception $e) {
            throw $e;
        }
    }
}
```



```
public function consultarMovimientos($cuenta) {
    try {
        $dao = new CuentaDAO();
        $lista = $dao->consultarMovimientos($cuenta);
        return $lista;
    } catch (Exception $e) {
        throw $e;
    }
}
}
?>
```

5.4.4 Capa: DAO

Archivo: Eureka\dao\EmpleadoDAO.php

```
<?php
require_once '../ds/AccesoDB.php';

class EmpleadoDAO {

    // Consulta los datos de un empleado por su nombre de usuario
    public function consultarPorUsuario( $usuario ){
        try {
            $query = "select * from empleado where vch_emplusuario='$usuario'";
            $db = AccesoDB::getInstancia();
            $lista = $db->executeQuery($query);
            $rec = null;
            if( count($lista) == 1 ){
                $rec = $lista[0];
            }
            return $rec;
        } catch (Exception $e) {
            throw $rec;
        }
    }
}
}
?>
```

Archivo: Eureka\dao\CuentaDAO.php

```
<?php
require_once '../ds/AccesoDB.php';
```

```
// Clase que provee los servicios con las cuentas
class CuentaDAO {

    // Retorna los datos de la cuenta en un arreglo asociativo
    public function consultarCuenta($cuenta) {
        try {
            $query = "select * from cuenta where chr_cuencodigo='$cuenta'";
            $db = AccesoDB::getInstancia();
            $lista = $db->executeQuery($query);
            $rec = null;
            if( count($lista) == 1 ) {
                $rec = $lista[0];
            }
            return $rec;
        } catch (Exception $e) {
            throw $rec;
        }
    }

    // Retorna los movimientos de una cuenta en un arreglo
    public function consultarMovimientos($cuenta) {
        try {
            $query = "select * from movimiento where chr_cuencodigo='$cuenta'";
            $db = AccesoDB::getInstancia();
            $lista = $db->executeQuery($query);
            return $lista;
        } catch (Exception $e) {
            throw $rec;
        }
    }
}

?>
```

5.5 Ejecución del Proyecto

5.5.1 Proceso de Logueo

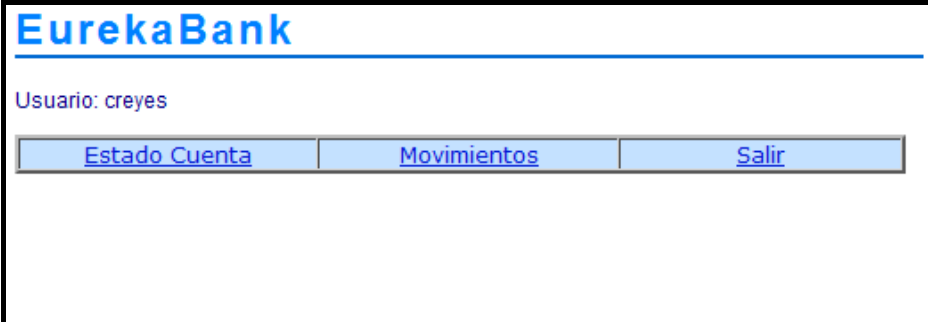
Cuando ejecutamos la aplicación lo primero que debemos hacer es iniciar sesión, la interfaz de usuario (IU) es la siguiente:



Debe ingresar su nombre de usuario y clave, de ser correctos se mostrara la IU principal de la aplicación.

5.5.2 Interfaz de Usuario Principal

Si en la IU de logueo ingreso los datos correctos pasará a la IU principal que se muestra en el siguiente gráfico.



EurekaBank

Usuario: creyes

Estado Cuenta	Movimientos	Salir
-------------------------------	-----------------------------	-----------------------

En la IU principal tiene una barra de menú con tres opciones:

- **Estado Cuenta:** Permite ver el estado de una cuenta y sus saldo.
- **Movimientos:** Permite consultar los movimientos de una cuenta.
- **Salir:** Cierra la sesión del usuario y carga la IU de logueo.

5.5.3 Consultar el Estado de una Cuenta

Cuando hacemos clic en la opción **Estado Cuenta** en la IU principal, accedemos a la IU **Estado de una Cuenta** como se muestra a continuación:



EurekaBank

Usuario: creyes

Estado Cuenta	Movimientos	Salir
-------------------------------	-----------------------------	-----------------------

Estado de una Cuenta

Nro. Cuenta

Lo que debemos hacer es ingresar un número de cuenta correcto y hacer clic en el botón consultar, y debemos tener el estado actual de la cuenta, como se muestra a continuación:



EurekaBank

Usuario: creyes

[Estado Cuenta](#) [Movimientos](#) [Salir](#)

Estado de una Cuenta

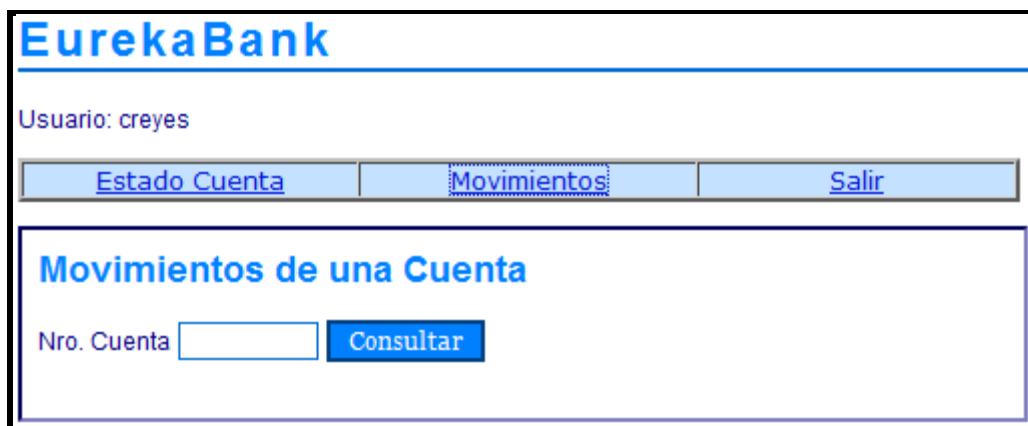
Nro. Cuenta

Resultado

Cuenta	00100001
Estado	ACTIVO
Saldo	6900.00

5.5.4 Consultar Movimiento de una Cuenta

Para consultar los movimientos de una cuenta debe hacer clic en la opción **Movimientos**, y obtiene la siguiente IU:



EurekaBank

Usuario: creyes

[Estado Cuenta](#) [Movimientos](#) [Salir](#)

Movimientos de una Cuenta

Nro. Cuenta

Luego debe ingresar el número de cuenta y hacer clic en el botón **consultar**, y obtendrá el resultado:

EurekaBank

Usuario: creyes

[Estado Cuenta](#)[Movimientos](#)[Salir](#)

Movimientos de una Cuenta

Nro. Cuenta [Consultar](#)

Resultado

Nro. Mov.	Fecha	Tipo	Importe
1	2008-01-06	001	2800.00
2	2008-01-15	003	3200.00
3	2008-01-20	004	800.00
4	2008-02-14	003	2000.00
5	2008-02-25	004	500.00
6	2008-03-03	004	800.00
7	2008-03-15	003	1000.00

Finalmente, para finalizar su sesión deba hacer clic en la opción **Salir**.