Control de Errores

Tipos de Errores

- **Error de análisis**: Es un problema con la sintaxis, por ejemplo, falta un **;**
- **Error Fatal**: Llamar a una función que no se ha definido.
- **Advertencia**: Algo va mal pero el intérprete puede continuar. Por ejemplo, número erróneo de argumentos en una función.
- **Aviso**: Imprimir una variable sin inicializar.
- **Aviso estricto**: Sobre el estilo de código se daban en los cambios de PHP 4 a PHP 5. Es difícil que se den.

Manejo de Errores

- Directivas a nivel de php.ini
  - **display_errors** = On y para enviar los errores al log del Servidor activar la directiva:
  - **log_errors** a On.
  - **error_reporting**: controla los tipos de errores que informa PHP. Por defecto es:
    - **E_ALL & ~E_NOTICE & ~E_STRICT**
    - Se traduce por: muestra todos menos los avisos y los avisos estrictos.
    - ~ niega.

Modos de error_reporting

- **E_ALL**: todos los errores excepto los estrictos.
- **E_PARSE**: Errores de análisis.
- **E_ERROR**: Errores fatales.
- **E_WARNING**: Advertencias.
- **E_NOTICE**: Avisos.
- **E_STRICT**: Avisos estrictos.

La función die

- **die** (mensaje): Se coloca con instrucciones que puedan generar errores:
  - Por ejemplo: si vamos a abrir un fichero y no existe.
  - La forma de utilizarla sería algo así:
    - fopen(fichero, modo) **or die**(‘se ha producido un error al abrir el fichero’);

Gestión de Excepciones

- Esquema de las Excepciones:

```
try {
    // Código que puede generar excepciones.
} catch (ClaseExcepcion1 $exc1){
    // Procesamiento de las excepciones de la clase 1
} [catch (ClaseExcepcion2 $exc2) {
    // Procesamiento de las excepciones de la clase 2.
}]
```

Gestión de las Excepciones

- Las excepciones las podemos provocar cuando se dé una situación anormal en nuestro código.
  - Con throw new Exception($mensaje, $codigo)

- Al lanzar la Excepción el código termina y ejecuta el código del constructor de la clase Exception con los parámetros que la hayamos pasado.

La clase Exception

```
class Exception {
    protected $message = 'Unknow exception';
    protected $code = 0;
    protected $file;
    protected $line;

    function __construct ($message=null, $code=0);
    final function getMessage();
    final function getCode();
    final function getFile();
    final function getLine();

    function __toString();  → Redefinible.
}
```

Podemos crear nuestras propias excepciones heredando de la clase Exception.

Ejemplo

```
class BD_Error_Recuperable   extends Exception {};

class ConexionBD {
  // Asignación de códigos de error
  const BD_ERROR_CONEXION  = 805;

function __construct(…) {
try {
      // Asignamos las propiedades
      $this->servidorBD = $servidor;
      $this->baseDatos  = $bd;
      $this->usuarioBD  = $usuario;
      $this->pclave     = $pclave;

      // Establecemos la conexión con el servidor de BD
      $id = @mysql_connect($this->servidorBD,
                           $this->usuarioBD, $this->pclave);
      if (!$id)
        throw new BD_Error_Recuperable("Conexión con el servidor $servidor",
                                       self::BD_ERROR_CONEXION);

      // Seguimos trabajando con el servidor
      // ...

      return ($id);
    }
```

Esta clase implementa una conexión a la base de datos. Y en caso de que haya error lanza una Excepción.

La @ ignora la línea en caso de Error.

Constante de Clase