Reutilizarea codului. Funcţii în *PHP*

Reutilizarea codului

PHP furnizează două instrucțiuni foarte simple care permit reutilizarea codului. Utilizând instrucțiunile *require*() sau *include*(), se poate încărca un fişier într-un script PHP. Fişierul inclus poate conține orice componentă ce poate exista într-un script (instrucțiuni PHP, text, *tag*-uri HTML, funcții PHP, clase PHP).

Aceste instrucțiuni lucrează similar incluziunilor de pe partea de server oferite de multe servere web, sau directivelor *#include* din C sau C++.

Instrucțiunile *require*() şi *include*() sunt aproape identice, singura diferență având loc atunci când acestea eşuează, caz în care *require*() conduce la o eroare fatală, iar *include*() întoarce doar un *warning*.

Există două variante ale lui *require*() şi *include*(), denumite *require_once*() şi, respectiv, *include_once*(). Scopul acestor instrucțiuni este cel de a asigura ca un fişier să poată fi inclus o singură dată. Această funcționalitate devine utilă atunci când *require*() şi *include*() sunt folosite pentru a include biblioteci de funcții. Astfel, se evită includerea de două ori a aceleiaşi biblioteci de funcții, lucru care ar conduce la redefinirea funcțiilor şi la apariția unei erori. Pe de altă parte, se recomandă utilizarea instrucțiunilor *require*() şi *include*(), deoarece acestea se execută mai repede.

Extensiile fişierelor incluse

Următoarea linie de cod este stocată în fişierul *reusable.php*:

```
<?php
      echo 'Aici este o instructiune PHP simpla.<br/>';
?>
```

Următorul fragment de cod se află în fişierul *main.php*:

```
<?php
      echo 'Fisierul principal.<br/>';
      require('reusable.php');
      echo 'Incheiere script.<br/>';
?>
```

La încărcarea script-ului *reusable.php* va apărea mesajul „*Aici este o instructiune PHP simpla.*”. La încărcarea script-ului *main.php*, apare: http://193.226.51.37/web/curs4/main.php

La execuția script-ului *main.php* instrucțiunea *require*('*reusable.php*') este înlocuită cu conținutul fișierului *reusable.php*, iar script-ul este apoi executat.

Nu este importantă extensia fișierului referit în instrucțiunea *require*(). Atât timp cât acest fișier nu este încărcat direct, el poate avea orice nume dorim. Dacă includem un *html* printr-o instrucțiune *require*(), va fi procesat codul PHP din interiorul lui. Se recomandă utilizarea unei convenții în privința extensiilor, cum ar fi *.inc* sau *.php*.

Observație: Dacă este inclus un fișier cu o extensie non-standard (de exemplu, *.inc*), acesta este stocat în ierarhia de documente web iar utilizatorii vor putea vizualiza conținutul acestuia direct în browser, ca text, prin urmare ar putea avea acces la parole sau alte informații sensibile. Se recomandă fie ca aceste fișiere să fie stocate în afara ierarhiei de documente, fie să utilizeze extensii standard.

Codul din fișierul inclus trebuie plasat între tag-uri PHP, pentru a putea fi tratat ca atare. Altfel, codul va fi tratat ca text sau ca HTML şi nu va fi executat.

Utilizarea instrucțiunii *require*() pentru *template*-urile site-urilor web

Dacă paginile web ale unei companii au un aspect consistent, putem adăuga *template*-ul şi elementele standard utilizând *require*().

De exemplu, presupunem că site-ul are un număr de pagini, toate cu aspectul următor: http://193.226.51.37/web/curs4/home.php. Atunci când este adăugată o pagină nouă, programatorul poate deschide o pagină existentă, înlocui textul şi salva fişierul cu un nume nou.

Să presupunem că site-ul web are un număr mare de pagini, toate urmând acelaşi stil. Dorim să modificăm acest aspect standard, lucru care devine foarte costisitor pentru un număr mare de pagini.

Reutilizarea directă a secțiunilor HTML comune tuturor paginilor constituie o abordare mai bună şi mai eficientă. Codul sursă al paginii *home* (*home.html*) este: http://193.226.51.37/web/curs4/home_html.txt

Se poate observa un număr de secțiuni distincte de cod în fişierul anterior. *Head*-ul HTML conține definiții CSS (*cascading style sheet*)

utilizate de către pagină. Secțiunea etichetată „*page header*” afișează numele companiei şi logo-ul, „menu” creează bara de navigare a paginii, iar „page content” conține textul specific paginii. Fişierul mai conține *footer*-ul paginii.

Fişierul poate fi împărțit în 3 fişiere: *header.php*, *home.php* şi *footer.php*. Fişierele *header.php* şi *footer.php* conțin cod care va fi reutilizat în alte pagini.

Fişierul *home.php* înlocuieşte *home.html* cuprinzând conținutul unic al paginii şi două instrucțiuni *require*(), care încarcă *header.php* şi *footer.php*: http://193.226.51.37/web/curs4/home_php.txt

Fişierul *header.php* conține definițiile CSS utilizate de către pagină, tabelele care afişează numele companiei şi meniurile de navigare: http://193.226.51.37/web/curs4/header_php.txt

Fişierul *footer.php* conține tabelul care afişează *footer*-ul în partea de jos a paginii: http://193.226.51.37/web/curs4/footer_php.txt

Această abordare ne permite să obținem un site web cu un aspect consistent, în care o pagină nouă, având acelaşi stil, se poate obține astfel:

```
<?p'hp require('header.php'); ?>
     Here is the content for this page
<?php require('footer.php'); ?>
```

Cel mai important lucru este acela că, după ce au fost create mai multe pagini care utilizează acelaşi *header* şi *footer*, fişierele pot fi modificate cu uşurință. Indiferent cât de mare este modificarea adusă, aceasta trebuie efectuată o singură dată.

Exemplul anterior utilizează doar cod HTML în corpul, antetul şi subsolul fişierului, însă acest lucru nu este necesar, putându-se folosi instrucțiuni PHP pentru generarea dinamică a unor părți din pagină.

Dacă dorim ca un fişier să fie tratat ca fişier text sau HTML şi să nu determine execuția de cod PHP, putem folosi funcția *readfile*(). Aceasta afişează conținutul unui fişier fără a-l parsa. Utilizarea acestei funcții poate fi o măsură importantă de precauție în cazul în care sunt folosite intrări furnizate de către utilizator.

Utilizarea opțiunilor *auto_prepend_file* şi *auto_append_file*

Adăugarea *header*-ului şi a *footer*-ului în fiecare pagină se poate realiza şi în alt mod, cu ajutorul opțiunilor *auto_prepend_file* şi *auto_append_file* din fişierul *php.ini*. Setând valorile acestora la fişierele corespunzătoare *header*-ului, respectiv *footer*-ului, ne asigurăm că acestea vor fi încărcate înainte şi după fiecare pagină. Fişierele incluse în acest mod

se vor comporta ca şi cum ar fi fost adăugate cu ajutorul instrucțiunii *include*(); dacă fişierul lipseşte, va apărea un *warning*.

De exemplu, setarea acestor opțiuni în *Windows* se realizează astfel:

```
auto_prepend_file = "c:/Program Files/Apache Software
Froundation/Apache2.2//include/header.php"
auto_append_file = "c:/Program Files/Apache
Group/Apache2/include/footer.php"
```

Dacă este utilizat un server web *Apache*, se pot utiliza diferite opțiuni de configurare pentru directoare individuale. Pentru aceasta, este necesar ca server-ul să permită suprascrierea fişierului său principal de configurare. Pentru stabilirea opțiunilor anterioare pentru un director, trebuie creat un fişier denumit *.htaccess* în directorul respectiv. Acest fişier va trebui să conțină următoarele două linii (cu o sintaxă uşor diferită față de cea din *php.ini*):

```
php_value auto_prepend_file "/home/username/include/header.php"
php_value auto_append_file "/home/username/include/footer.php"
```

Stabilirea opțiunilor în fişierul *.htaccess*, în loc de *php.ini*, oferă multă flexibilitate. Astfel, pot fi afectate doar setările care se referă la directoarele proprii de pe un calculator partajat. Server-ul web nu trebuie repornit şi nu sunt necesare drepturi de administrator. Un dezavantaj al metodei care utilizează fişiere *.htaccess* este acela că fişierele sunt citite şi parsate de fiecare dată când un fişier din acel director este accesat, şi nu doar o singură dată la început.

Utilizarea funcțiilor în PHP

O funcție reprezintă un modul independent de cod care furnizează o interfață pentru apel, efectuează anumite acțiuni şi, opțional, returnează un rezultat.

Apelarea funcțiilor

Cel mai simplu apel este cel la o funcție (*function_name*) care nu are parametri şi ignoră valorile care ar putea fi returnate este:

```
function_name();
```

Funcția *phpinfo*() este apelată în acest mod şi este adesea utilă pentru testare deoarece afişează versiunea instalată de PHP, informații despre PHP, server-ul web şi valorile diferitelor variabile PHP sau ale server-ului.

Funcțiile care necesită parametri sunt apelate plasând datele sau variabilele între paranteze. Exemple de apeluri:

```
function_name('parameter'); //parametru sir de caractere
function_name(2);
function_name(7.993);
function_name($variable); //variabila poate avea orice tip,
                          //inclusiv vector sau obiect
```

Un parametru poate avea orice tip de date, dar funcțiile prevăd de obicei tipuri particulare de date pentru parametri.

Prototipul unei funcții furnizează numărul de parametri ai unei funcții şi tipurile de date ale acestora. De exemplu, prototipul funcției *fopen* este următorul:

```
resource fopen ( string filename, string mode
                 [, bool use_include_path [, resource context]])
```

În fața numelui funcției este specificat tipul de date al rezultatului. Parametrii funcției se află între paranteze şi sunt specificați împreună cu tipurile lor de date. Pentru parametrii opționali, acestora fie le sunt furnizate valori, fie sunt ignorați, caz în care sunt utilizate valorile lor implicite. Dacă o funcție are mai mult de un parametru opțional, pot fi ignorați doar parametrii situați cel mai la dreapta (în cazul funcției *fopen*, nu putem furniza valoare pentru *context* şi ignora parametrul *use_ignore_path*).

Apelarea unei funcții nedefinite

Dacă este apelată o funcție care nu există, va apărea un mesaj de eroare: http://193.226.51.37/web/curs4/undefined.php

Motivele pentru care o astfel de eroare poate apărea sunt următoarele:

- numele funcției nu a fost scris corect
- funcția nu există în versiunea de PHP utilizată
- funcția apelată face parte dintr-o extensie PHP care nu a fost încărcată.

Case-sensitivity

Apelurile la funcții nu sunt *case-sensitive*. Spre deosebire de acestea, numele variabilelor sunt *case-sensitive*.

Funcţii definite de utilizator

O declaraţie de funcţie începe cu cuvântul cheie *function*, specifică numele funcţiei, parametrii acesteia şi codul care va fi executat la fiecare apel al funcţiei.

Exemplu:

```
function my_function() {
      echo 'My function was called';
}
```

Funcţiile definite de utilizator sunt valabile doar în script-ul în care au fost declarate. Se recomandă să existe un fişier sau set de fişiere ce conţin funcţiile utilizate, iar apoi să avem o instrucţiune *require*() în script-urile în care funcţiile sunt apelate, astfel încât acestea să devină disponibile.

Corpul funcţiei (specificat între {}) poate conţine orice este permis într-un script PHP, incluzând apeluri de funcţii, declaraţii de variabile noi, funcţii, instrucţiuni *require*() sau *include*(), declaraţii de clase şi cod HTML. Dacă dorim să ieşim din PHP în cadrul unei funcţii şi să scriem cod HTML, acest lucru se poate realiza în acelaşi mod ca în script, cu un tag PHP de încheiere urmat de codul HTML.

Exemplu: Codul următor este o modificare a celui din exemplul precedent şi furnizează acelaşi rezultat.

```
<?php
function my_function() {
?>
      My function was called
<?php
}
?>
```

Numele funcţiilor

Numele funcţiilor trebuie să fie scurte, însă descriptive. Restricţiile care au loc asupra numelor funcţiilor sunt următoarele:

- două funcţii nu pot avea acelaşi nume
- numele funcţiei poate conţine doar litere, cifre şi _
- numele funcţiei nu poate începe cu o cifră.

Multe limbaje permit supraîncărcarea funcţiilor. În PHP acest lucru nu este posibil. Pe de altă parte, funcţiile definite de utilizator există doar în script-urile în care au fost declarate, prin urmare putem reutiliza numele unei

funcţii într-un alt fişier, însă acest lucru ar conduce la confuzie şi ar trebui evitat.

Exemplu: Următoarele nume de funcţii sunt permise:

```
name()
name2()
name_three()
_namefour()
```

Următoarele nume de funcţii nu sunt legale:

```
5name()
name-six()
fopen() // exista deja
```

Observaţie: Deşi *$name* nu este un nume de funcţie valid, apelul

```
$name();
```

poate fi corect, în funcţie de valoarea lui *$name*. PHP ia valoarea stocată în *$name*, caută o funcţie cu acel nume şi o apelează (funcţie variabilă).

Utilizarea parametrilor

Următoarea funcţie cere un parametru (un vector unidimensional) şi îl afişează sub forma unui tabel.

```
function create_table($data) {
      echo "<table border=\"1\">";
      reset($data); // Pointeaza la inceput
      $value = current($data);
      while ($value) {
            echo "<tr><td>".$value."</td></tr>\n";
            $value = next($data);
      }
      echo "</table>";
}
```

Apelul funcţiei:

```
$my_array = array('Line one.', 'Line two.', 'Line three.');
create_table($my_array);
```

produce următorul rezultat: http://193.226.51.37/web/curs4/create_table.php

Funcţiile definite de utilizator pot avea parametri multipli, iar unii dintre aceştia pot fi opţionali. Modificăm funcţia anterioară astfel încât să putem specifica bordura sau alte atribute ale tabelului:

```
<?php
function create_table2($data, $border=1, $cellpadding=4,
                       $cellspacing=4 ) {
      echo "<table border=\"".$border."\"
      cellpadding=\"".$cellpadding."\"
```

```
      cellspacing=\"".$cellspacing."\">";
      reset($data);
      $value = current($data);
      while ($value) {
            echo "<tr><td>".$value."</td></tr>\n";
            $value = next($data);
      }
      echo "</table>";
}
```

Această funcție poate fi apelată în următoarea secvență de cod:

```
$my_array = array('Line one.','Line two.','Line three.');
create_table2($my_array, 3, 8, 8);
```

Primul parametru al funcției este obligatoriu, iar următoriii 3 sunt opționali deoarece le-au fost specificate valori implicite. Rezultatul apelului anterior este : http://193.226.51.37/web/curs4/create_table2.php

Se pot declara funcții cu număr variabil de parametri. Putem afla câți parametri au fost transmişi, precum şi valorile acestora, cu ajutorul a 3 funcții: *func_num_args*(), *func_get_arg*() şi *func_get_args*().

Fie funcția:

```
function var_args() {
      echo "Numarul de parametri:";
      echo func_num_args();
      echo "<br />";
      $args = func_get_args();
      foreach ($args as $arg) {
            echo $arg."<br />";
      }
}
```

Această funcție arată numărul de parametri care i-au fost transmişi şi afişează valorile acestora. Funcția *func_num_args*() returnează numărul de argumente transmise. Funcția *func_get_args*() întoarce un vector de argumente. Alternativ, argumentele pot fi accesate unul câte unul cu ajutorul funcției *func_get_arg*(), căreia îi transmitem numărul argumentului pe care dorim să îl accesăm (începând cu 0).

Domeniu de vizibilitate

Atunci când anumite variabile trebuie folosite în interiorul unui fişier inclus, acestea trebuie declarate în script înainte de instrucțiunea *require*() sau *include*(). Când folosim o funcție, transmitem explicit acele variabile funcției.

Domeniul unei variabile controlează modul în care o variabilă este vizibilă şi utilizabilă. Limbajele de programare au diferite reguli privind domeniul de vizibilitate al variabilelor. Regulile din PHP sunt următoarele:

- Variabilele declarate în interiorul unei funcţii sunt vizibile începând cu instrucţiunea care le declară, până la încheierea funcţiei (}). Acesta se numeşte domeniul funcţiei, iar variabilele de acest tip se numesc variabile locale.
- Variabilele declarate în afara funcţiilor sunt vizibile începând cu instrucţiunea care le declară, până la sfârşitul fişierului, dar nu în interiorul funcţiilor. Un astfel de domeniu se numeşte domeniu global, iar variabilele de acest tip se numesc variabile globale.
- Variabilele speciale superglobale sunt vizibile atât în interiorul cât şi în exteriorul funcţiilor.
- Utilizarea instrucţiunilor *require*() şi *include*() nu afectează domeniul de vizibilitate. Dacă instrucţiunea este folosită în interiorul unei funcţii, se aplică domeniul funcţiei. În caz contrar, se aplică domeniul global.
- Cuvântul cheie *global* poate fi folosit pentru a obţine explicit dacă o variabilă definită sau utilizată în cadrul unei funcţii va avea domeniu global.
- Variabilele pot fi şterse manual apelând *unset*(*$nume_variabila*). O variabilă nu se mai află în domeniu dacă a fost aplicată funcţia *unset*().

Următorul cod nu produce nicio ieşire. În cadrul funcţiei este declarată o variabilă *$var*, al cărei domeniu va fi, astfel, domeniul funcţiei. Referirea la *$var* în exteriorul funcţiei creează de fapt o nouă variabilă *$var*, al cărei domeniu este global şi care va fi vizibilă până la sfârşitul fişierului.

```
function fn() {
      $var = "contents";
}
      fn();
echo $var; //$var nu a primit nicio valoare
```

Exemplul următor declară o variabilă în exteriorul funcţiei şi încearcă să o utilizeze în interiorul acesteia.

```
<?
function fn() {
      echo "inside the function, \$var = ".$var."<br />";
      $var = "contents 2";
      echo "inside the function, \$var = ".$var."<br />";
}
```

```
$var = "contents 1"; //domeniu global
fn(); //creează variabila locala $var
echo "outside the function, \$var = ".$var."<br />";
```

Fragmentul de cod anterior va afişa:

```
inside the function, $var =
inside the function, $var = contents 2
outside the function, $var = contents 1
```

Dacă dorim ca o variabilă creată în interiorul unei funcții să fie globală, putem utiliza cuvântul cheie *global*:

```
function fn() {
      global $var;
      $var = "contents";
      echo "inside the function, \$var = ".$var."<br />";
}
fn();
echo "outside the function, \$var = ".$var."<br />";
```

În acest fel, variabila *$var* va exista în exteriorul funcției, după ce aceasta este apelată. Fragmentul de cod anterior va afişa:

```
inside the function, $var = contents
outside the function, $var = contents
```

Cuvântul cheie *global* poate fi utilizat la începutul unui script, la prima utilizare a unei variabile, pentru a declara că domeniul acesteia este întregul script. Acesta este un mod de utilizare mai obişnuit al cuvântului cheie *global*.

Transfer prin referință şi transfer prin valoare

Considerăm că avem următoarea funcție pentru incementarea unei valori:

```
function increment($value, $amount = 1) {
      $value = $value +$amount;
}
```

Această funcție nu realizează nimic. Următorul fragment de cod va afişa valoarea 10:

```
$value = 10;
increment ($value);
echo $value;
```

Conținutul variabilei *$value* nu a fost modificat din cauza regulilor referitoare la domeniu. Codul precedent creează variabila *$value*, care conține valoarea 10, apoi apelează funcția *increment*(). Variabila *$value* din

funcție este creată atunci când funcția este apelată. Valoarea lui *$value* devine 11 în interiorul funcției, până la sfârşitul acesteia. După aceasta, se revine în codul apelant, în care *$value* este o variabilă diferită, cu domeniu global, şi prin urmare nemodificată.

Un mod de a rezolva această problemă constă în declararea lui *$value* ca variabilă globală în interiorul funcției, dar aceasta ar însemna că, pentru a utiliza această funcție, variabila pe care dorim să o incrementăm trebuie să se numească *$value*.

Acesta este transferul prin valoare al parametrilor funcției: la transferul unui parametru, este creată o nouă variabilă ce conține valoarea transferată. Valoarea poate fi modificată în funcție, însă valoarea variabilei originale, din exteriorul funcției, rămâne nemodificată.

Transferul prin referință al parametrilor constituie o abordare mai bună. Atunci când un parametru este transferat unei funcții, în loc de a crea o nouă variabilă, funcția primeşte o referință la variabila originală. Această referință are un nume de variabilă şi este utilizată exact ca o variabilă. Diferența este că, în loc de a avea o valoare proprie, doar referă valoarea unei variabile. Orice modificare adusă referinței afectează şi variabila originală.

Faptul că un parametru este transferat prin referință este specificat prin specificarea caracterului & în fața numelui parametrului. Nu va apărea nicio modificare în apelul funcției.

Pentru ca exemplul anterior să funcționeze corect, funcția trebuie modificată astfel:

```
function increment(&$value, $amount = 1) {
      $value = $value +$amount;
}
```

Utilizarea cuvântului cheie *return*

Cuvântul cheie *return* poate fi folosit pentru a întoarce o valoare furnizată de funcție sau pentru a opri execuția unei funcții. Atunci când o funcție se încheie, controlul va trece la următoarea instrucțiune după apelul funcției.

O condiție referitoare la o eroare este un motiv obişnuit pentru a încheia execuția unei funcții. De exemplu, funcția următoare returnează maximul dintre două numere, însă execuția sa se opreşte dacă vreunul dintre numere lipseşte:

```
function larger( $x, $y ) {
      if ((!isset($x)) || (!isset($y))) {
            echo "This function requires two numbers.";
            return;
```

```
      }
      if ($x>=$y) {
            echo $x."<br/">;
      } else {
            echo $y."<br/">;
      }
}
```

Funcția predefinită *isset*() spune dacă o variabilă a fost creată şi i-a fost acordată o valoare.

Funcția anterioară poate fi rescrisă astfel încât să returneze (nu să afişeze) valoarea cea mai mare:

```
function larger ($x, $y) {
      if ((!isset($x)) || {!isset($y))) {
            return false;
      } else if ($x>=$y) {
            return $x;
      } else {
            return $y;
      }
}
```

Observație: Valoarea returnată de un apel la funcția anterioară trebuie testată cu ===, pentru a nu se crea o confuzie între 0 şi false.

Recursivitate

PHP pemite definirea funcțiilor recursive. Astfel de funcții sunt utile pentru navigarea în structurile de date dinamice, cum ar fi listele înlănțuite sau arborii.

Este posibil ca recursivitatea să fie utilizată în locul iterațiilor, însă trebuie avut în vedere faptul că funcțiile recursive sunt mai lente şi utilizează mai multă memorie decât iterațiile, prin urmare ar trebui utilizate acestea din urmă.

Exemplu: Inversarea unui şir de caractere (2 variante)

```
<?php
function reverse_r($str) {
      if (strlen($str)>0) {
            reverse_r(substr($str, 1));
      }
      echo substr($str, 0, 1);
      return;
}

function reverse_i($str) {
      for ($i=1; $i<=strlen($str); $i++) {
            echo substr($str, -$i, 1);
```

```
        }
        return;
}
```

Spații de nume (*namespaces*)

Un spațiu de nume este un container abstract care reține un grup de identificatori; în PHP, aceasta înseamnă că spațiile de nume pot conține funcțiile, constantele şi clasele pe care le definim. Există câteva avantaje ale spațiilor de nume:

- Toate funcțiile, clasele şi constantele din cadrul unui *namespace* sunt prefixate automat cu numele acestuia.
- Numele claselor, funcțiilor şi ale constantelor sunt rezolvate la *runtime*, la prima căutare care are loc în *namespace*, înainte de a se trece la domeniul global.

Tratarea excepțiilor

Ideea de bază privind tratarea excepțiilor se referă la execuția codului care poate genera erori în interioul unui bloc *try*. Acesta este o secțiune de cod de forma următoare:

```
try
{
// cod sursa
}
```

Dacă apare o eroare în interiorul unui bloc *try*, putem declanşa o excepție. Anumite limbaje, precum *Java*, declanşează excepții în mod automat în anumite cazuri. În PHP, excepțiile trebuie declanşate manual, astfel:

```
throw new Exception('message', code);
```

Cuvântul cheie *throw* declanşează mecanismul de tratare a excepțiilor. Acesta este o construcție a limbajului şi nu o funcție, însă trebuie să îi transferăm o valoare, care este un obiect. În cazul cel mai simplu, se instanțiază clasa predefinită *Exception*, al cărei constructor primeşte doi parametri: un mesaj şi un cod. Ambii parametri sunt opționali.

La sfârşitul blocului *try*, este nevoie de cel puțin un bloc *catch*, a cărui formă este:

```
catch (typehint exception)
```

```
{
      // handle exception
}
```

Putem avea mai multe blocuri *catch*, fiecare tratând câte un tip diferit de excepție, asociate unui singur bloc *try*. Obiectul transferat în blocul *catch* este cel trimis în cadrul instrucțiunii *throw* care a declanşat excepția. Atunci când este declanşată o excepție, PHP caută blocul *catch* corespunzător. Dacă avem mai multe blocuri catch, *obiectele* transferate fiecăruia dintre acestea ar trebui să fie de tipuri diferite.

Un bloc *catch* poate declanşa, la rândul lui, alte excepții.

Exemplu:

```
<?php
try {
      throw new Exception("A terrible error has occurred", 42);
}
catch (Exception $e) {
      echo "Exception ". $e->getCode(). ": ".
                        $e->getMessage()."<br />".
            " in ". $e->getFile(). " on line ".
                    $e->getLine(). "<br />";
}
?>
```

Exemplul anterior produce următorul rezultat:
http://193.226.51.37/web/curs4/basic_exception.php

În exemplul anterior au fost utilizate câteva metode ale clasei *Exception*, care va fi discutată în continuare.

Clasa *Exception*

Constructorul acestei clase acceptă 2 parametri: mesajul şi codul erorii. Pe lângă acesta, clasa furnizează următoarele metode predefinite:

- *getCode*() – returnează codul erorii;
- *getMessage*() – returnează mesajul erorii;
- *getFile*() – returnează calea absolută a fişierului în care a apărut excepția;
- *getTrace*() – returnează un vector care conține un *backtrace*, acolo unde excepția a fost declanşată (funcțiile care se executau la momentul apariției excepției);
- *getTraceAsString*() – aceeaşi informație ca *getTrace*(), formatată ca string;

- *__toString*() – permite afişarea unui obiect *Exception*, furnizând informațiile din metodele de mai sus.

Excepții definite de utilizator

Înclauza *throw* poate fi transmis orice obiect. Acest lucru poate fi util la *debug*, dacă un anumit obiect creează probleme.

De cele mai multe ori, se extinde clasa *Exception*. Pentru aceasta, este necesar să îi cunoaştem structura:
http://www.php.net/manual/en/class.exception.php

Cele mai multe dintre metodele publice ale clasei *Exception* sunt finale, adică nu pot fi suprascrise. Putem crea propria subclasă, dar nu putem schimba comportamentul metodelor de bază. Funcția *__toString*() poate fi modificată, astfel încât se poate modifica modul de afişare al excepției.

Următoarea clasă implementează o excepție definită de utilizator:

```
<?php
class myException extends Exception
{
      function __toString()
      {
        return "<table border=\"1\">
                  <tr>
                  <td><strong>Exception ".$this->getCode()."
                  </strong>: ".$this->getMessage()."<br />"."
                        in ".$this->getFile()." on line ".
                        $this->getLine()."
                  </td>
                  </tr>
                </table><br />";
      }
}
try
{
      throw new myException("A terrible error has occurred", 42);
}
catch (myException $m)
{
      echo $m;
}
?>
```

Diferența dintre clasa anterioară şi *Exception* constă în faptul că afişarea se face într-un mod mai plăcut:
http://193.226.51.37/web/curs4/user_defined_exception.php

Exemple de excepții în site-ul web exemplu (componente auto)

Anterior, am studiat modul în care comenzile lansate prin intermediul site-ului web pot fi stocate într-un fişier. Operațiile I/O asupra fişierelor reprezintă o sursă de apariție frecventă a excepțiilor.

În codul corespunzător procesării formularului, pot apărea 3 probleme referitoare la fişiere:

- fişierul nu poate fi deschis
- nu poate fi obținută o blocare asupra fişierului
- nu se poate scrie în fişier.

Creem câte o clasă corespunzătoare fiecărui tip de excepție prevăzut; clasele vor fi salvate în fişierul *file_exceptions.php*:

http://193.226.51.37/web/curs4/file_exceptions_php.txt

Fişierul *processorder.php* este rescris astfel încât să utilizeze aceste excepții: http://193.226.51.37/web/curs4/processorder_php.txt

Au fost incluse doar două blocuri *catch*: unul care tratează excepțiile *fileOpenException*, iar celălalt tratează excepțiile de tip *Exception*. Deoarece celelelate două tipuri de excepții nu sunt tratate explicit, însă moştenesc clasa *Exception*, vor fi tratate de către al doilea bloc *catch*.

Dacă este declanşată o excepție pentru care nu există un bloc *catch* corespunzător, PHP va raporta o eroare fatală.