**Rukovanje greškama**

Jedinica: 17 od 19

**Rezime**

* Najčešći uzrok grešaka jeste problem u pristupu spoljnim resursima (fajlovi, baza, korisnički ulaz, strim);
* Greške se predupređuju logikom u kodu, obradom izuzetaka, obradom grešaka;
* Moguće je ručno aktivirati grešku;
* Moguće je napraviti sopstveni hendler za rukovanje greškama, koji će hvatati sve greške ili one određenog tipa i tim hendlerom prepisati ugrađeni hendler;
* Moguće je izbaciti sopstveni izuzetak tokom izvršavanja koda;
* Moguće je napraviti sopstvenu klasu izuzetaka koja nasleđuje klasu Exception;
* Izuzecima se rukuje tako što se glavni kod stavlja u try blok, a alternativni kod u catch blok, pri čemu oba bloka moraju postojati i catch blok mora prihvatiti promenljivu koja je tipa izuzetak.

U ovoj lekciji biće objašnjeni pojmovi koji se odnose na greške koje se mogu javiti u toku programiranja, kao i načine na koje se može rukovati njima.

Rukovanje greškama veoma je bitan element u svakom programskom jeziku na praktično svim platformama. Nikada ne možemo biti potpuno sigurni šta će i kada korisnik uraditi, kao ni kada će sistem i njegovi resursi, iz nekog razloga, omesti naš program u pravilnom izvršavanju.

Upravo iz tog razloga, obaveza programera je da predvidi koje tačke u programu predstavljaju potencijalnu opasnost po njegovo izvršavanje i da te pozicije obezbedi dodatnim zaštitnim sistemima.

Najbolnije tačke jedne aplikacije su trenuci kada se ta aplikacija obraća resursima. A ti resursi, odnosno, njihovo poreklo, najčešće su baza podataka ili fajl sistem. Pored njih, kritični trenuci događaju se i prilikom svakog korisničkog ulaza u aplikaciju (unos podataka), zatim striminga, komunikacije sa uređajima i slično.

Osnovno rukovanje greškama je dosta jednostavno. Treba samo ispoštovati sledeći set pravila:

* + Staviti kritičan deo u poseban blok,
  + Staviti alternativni deo u poseban blok.

To možemo uraditi ručno ili putem *try-catch* blokova, u zavisnosti od tipa greške koji očekujemo i toga da li je u stanju da bude obrađena kao izuzetak.

**Programske greške**

Bez obzira na to koji se programski jezik koristi razlikujemo tri najopštije vrste grešaka, a to su:

* + Sintaksne greške,
  + Greške pri izvršavanju koda,
  + Logičke greške.

**Sintaksne greške –**PHP, kao i svaki drugi programski jezik, poštuje osnovno pravilo koje predstavlja sintaksu jezika. Ukoliko neka od naredbi ne poštuje pravila jezika, reći ćemo da ima *sintaksnu grešku.*Kada PHP skript sadrži sintaksnu grešku PHP – analizator koda neće moći da obradi deo skripta ili čak ceo skript u zavisnosti od ozbiljnosti greške. Pogledajmo sledeći kod:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <?php  if (true)  {  echo 'error is here';  ?> |

Greška koja će se generisati nakon izvršenja imaće sličnu strukturu:

*Parse error: parse error, unexpected $end in ~putanja do vašeg fajla~ on line 2*

Sintaksnu grešku čini zatvorena vitičasta zagrada koja nedostaje. Tačnije, zagrada je otvorena i započeta naredba za izvršavanje, ali, s obzirom na to da zagrada nije zatvorena, analizator nailazi na problem, jer ne zna gde da završi naš skript.

**Greška pri izvršavanju koda –**Otkriti i ispraviti grešku koja se javlja prilikom izvršavanja koda može biti teže od očekivanog. Ako vaš kod sadrži sintaksnu grešku, analizator će je otkriti čim pokušate da izvršite taj kod. Međutim, greška koja se javlja prilikom izvršavanja koda ne potiče uvek iz vašeg skripta, pa je nekada nije tako lako uočiti. Razlozi ovakvih grešaka mogu biti razni događaji i naredbe, ali se mogu nalaziti i između uzajamnih delovanja skripta. Često će se desiti da preuzimanjem podataka i obradom nekih već izgenerisanih informacija dođe do računske greške koja će ispisati:

*Warning: Division by zero.... ..... on line ...*

Moramo priznati da bi retko ko poželeo da deli sa nulom, ali i da se često desi ovakav problem koji predstavlja grešku pri izvršavanju.

**Logičke greške –**Najteže greške koje je najteže i otkloniti iz razloga što se vodite vašom logikom i očekujete da vaš kod radi ono što ste zamislili, međutim, i najmanja greška može izazvati apsolutni preokret izvršene naredbe. Uzrok može biti obična greška pri kucanju na primer:

|  |  |
| --- | --- |
| 1  2  3  4 | for ( $i = 0; $i < 10; $i++ );  {  echo ' Hello <br/ >';  } |

Kod koji smo prikazali u primeru je sasvim ispravan i izvršava se bez problema. Problem se javlja u samom izvršenju naredbe i krije se u **;** na kraju for naredbe. Rešenje naredbe biće izvršenje for petlje 10 puta bez ikakvog rezultata i izvršenje echo naredbe samo jednom. Verujemo da je korisnik pisanjem ovog koda očekivao rezultat u kome će naredba echo sa tekstom 'Hello' biti izvršena u okviru for petlje i ispisana deset puta na izlazu.

**Ručna obrada grešaka na najnižem nivou**

Pre nego što pristupimo nekom resursu, najbolje je ispitati da li taj resurs postoji i, na osnovu toga nastaviti delovanje programa. Na primer:

|  |  |
| --- | --- |
| 1  2  3 | <?php  $file = fopen("myFile.txt", "r");  ?> |

Ukoliko se fajl myFile.txt ne nalazi u folderu aplikacije, ovaj kod će izazvati grešku. Zato bi prethodno trebalo proveriti da li fajl postoji i, u odnosu na to, obezbediti ga:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php  if(!file\_exists("myFile.txt"))   {    $file = fopen("myFile.txt", "w");    fclose( $file );   }  $file = fopen("myFile.txt", "r");  ?> |

Ili, jednostavno, prekinuti izvršavanje programa:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <?php  if(!file\_exists("myFile.txt "))   {   die("Unknown file");   }  else   {   $file = fopen("myFile.txt ", "r");   }  ?> |

**die()**

Ova funkcija trenutno prekida izvršenje programa uz emitovanje (ili bez njega) korisnički definisane poruke. Ukoliko, naime, dalje izvršenje programa zavisi od resursa koje treba da pročitamo (na primer, od fajla myFile.txt u prethodnim primerima), onda svakako treba prekinuti dalje izvršavanje programa, ukoliko tog resursa nema.

U ovom primeru, ukoliko fajla nema, biće ispisana poruka greške, zatim poruka funkcije die (kraj) i izvršavanje će biti prekinuto.

|  |  |
| --- | --- |
| 1  2  3 | <?php  $file = fopen("myFile.txt ", "r") or die("end");  ?> |

**Ručno emitovanje grešaka**

U prethodnom primeru prekidali smo izvršavanje programa funkcijom die(). Takođe, moguće je i emitovanje sopstvene poruke o grešci bez prekidanja izvršavanja. Ova linija koda će na poziciji na kojoj se nalazi emitovati poruku o grešci.  
  
trigger\_error("My error");

Tip na ovaj način emitovane greške će biti 1024 (Notice), što možda nije nivo ozbiljnosti koji želimo da predočimo korisniku. Da bismo ovo promenili, možemo eksplicitno, kao drugi parametar, uneti jedan od tri tipa grešaka koje ova funkcija prihvata i čije konvencije bi trebalo da poštujemo pri korišćenju:

* + E\_USER\_WARNING (512) – Greška koja utiče na rad aplikacije (npr. nismo pronašli resurse), ali aplikacija može da nastavi sa radom,
  + E\_USER\_NOTICE (1024) – Samo obaveštenje; koristi se kada želimo korisnika da upozorimo na neku anomaliju u aplikaciji,
  + E\_USER\_ERROR (256) – Ova greška treba da prethodi prekidu izvršavanja programa.

Na primer:

*trigger\_error("My error",* E\_USER\_WARNING*);*

**Ručno rukovanje greškama**

Pored sopstvene aktivacije poruke o grešci, možemo napraviti i funkciju koja će presresti grešku i biti izvršena umesto ugrađene funkcije za rukovanje greškama, odnosno, sopstveni rukovalac greškama.

Ovakva funkcija prihvata parametre, broj greške, poruku greške, fajl koji je izazvao grešku, liniju greške, dump (niz varijabli i njihovih vrednosti aktivnih u trenutku greške). Ovi parametri su opcioni, ali njihov redosled treba poštovati pri izgradnji funkcije.

|  |  |
| --- | --- |
| 1  2  3  4 | function mojHendler($brojGreske, $porukaGreske)   {    echo $brojGreske . " " . $porukaGreske;   } |

Nakon što napravimo funkciju za koju želimo da rukuje greškama, potrebno je da joj dodelimo [inicijalnu vrednost](https://www.link-elearning.com/linkdl/opisPojma.php?id=139110) kao podrazumevani rukovalac greškama:

|  |  |
| --- | --- |
| 1 | set\_error\_handler("mojHendler"); |

Nakon toga, sve greške će biti, umesto u ugrađeni rukovalac, prosleđivane u našu ručno kreiranu funkciju.

Lista kodova (brojeva) greške, u odnosu na funkciju trigger\_error, ovde je malo proširena i sastoji se od 7 kodova:

* + E\_WARNING (2) - Greška koja ne prekida rad aplikacije, ali utiče (nema resursa...),
  + E\_NOTICE (8) – Samo obaveštenje,
  + E\_USER\_ERROR (256) – Fatalna greška, prekida se izvršenje,
  + E\_USER\_WARNING (512),
  + E\_USER\_NOTICE (1024),
  + E\_RECOVERABLE\_ERROR (4096) – Fatalna greška, koju je moguće obraditi i koja ne ostavlja trajne posledice po aplikaciju,
  + E\_ALL (8191) – Sve vrste grešaka.

Greške iz ove liste možemo priključiti čak i samom hendleru, kroz parametar funkcije set\_error\_handler:

set\_error\_handler("mojHendler", E\_USER\_NOTICE);

U ovom slučaju, kroz naš hendler će proći samo greške tipa E\_USER\_NOTICE, dok će ostale greške biti obrađivane kroz ugrađeni hendler.

Evo primera kompletnog koda:

function mojHendler( $brojGreske, $porukaGreske )  
{  
echo "Broj greske: " . $brojGreske . "<br>Poruka greske: " . $porukaGreske;   
 }  
//prepisivanje sistemskog hendlera korisnickim  
set\_error\_handler("mojHendler");  
//izazivanje greske unosom neinicijalizovane varijable  
echo( $mojaVarijabla );

Nakon uhvaćene greške, najbolje je proslediti je u neki log u bazi ili na fajl sistemu. Loger za neki specifičan sistem (na primer, da pamtimo logove u bazi) možemo napisati ručno, dok za standardne tipova loga možemo koristiti i ugrađenu funkciju error\_log().

Ova funkcija prima do četiri parametra: poruku greške, tip logovanja, destinaciju i zaglavlje.

Poruka greške je bilo koja korisnička poruka za koju želimo da bude zapamćena u logu. Tip logovanja predstavlja, zapravo, destinaciju slanja ovog loga, pri čemu je:

* + 0 – Default (koristi se sistemski definisan log),
  + 1 – Mail (log entry se šalje na mejl),
  + 3 – Fajl (log entry se upisuje u fajl),
  + U slučaju da je tip destinacije 1 ili 3 (mejl ili fajl), potrebni su i dodatni parametri (zaglavlje i destinacija za mail i adresu destinacioni fajl za fajl).

<?php  
error\_log("Moja greska", 1, "ciljniMail@mail.com", "From: mojMail@mail.com");  
?>

<?php  
error\_log("Moja greska", 3, "c:/myFile.log");  
?>

**Klasa Exception**

U okviru PHP-a postoji ugrađena klasa Exception. Konstruktor klase Exception prihvata dva parametra, prvi čini tekst poruke o grešci, dok drugi parametar predstavlja broj greške.

Osim samog [konstruktora](https://www.link-elearning.com/linkdl/opisPojma.php?id=139098), klasa Exception ima na raspolaganju i sledeće metode:

* + getCode() – vraća broj greške koji je bio prosleđen konstruktoru,
  + getMessage() – vraća tekst poruke koja je bila prosleđena konstruktoru,
  + getFile() – pozivajućem kodu vraća punu putanju datoteke u kojoj je generisan izuzetak,
  + getLine() – vraća broj reda u kodu u kome je generisan izuzetak,
  + getTrace() – vraća niz s podacima o stablu pozivanja (ili *backtrace* pokazuje koje su se sve funkcije izvršavale u trenutku generisanja izuzetka) koji omogućavaju utvrđivanje mesta na kome je generisan izuzetak,
  + getTraceAsString() – vraća iste podatke kao getTrace(), formatirane kao znakovni niz,
  + \_toString() – omogućava da se naredbi echo direktno prosledi sadržaj objekta Exception, sa svim podacima koje daju navedene metode.

|  |
| --- |
| **Napomena**  Kako bismo ispratili kontekst lekcije, pomenuli smo i klasu Exception, koja je inače deo objektnog koncepta koji se ne obrađuje u ovom kursu. Zbog toga će neki pojmovi ostati nedorečeni. |

**Rukovanje izuzecima**

Nije uvek moguće rešavanje problema sa resursima (i sličnih) putem uslovnih izvršavanja koda. Ponekad je jednostavno potrebno pokušati nešto da bi se znalo da li to nešto može da se uradi. U tom slučaju, treba nam mehanizam koji će reagovati tek nakon dolaska do greške, a da pritom ne poremeti normalan rad programa. Ovakav mehanizam postoji i naziva se rukovalac izuzecima.

Rukovalac izuzecima je pojam blisko vezan za objektno orijentisano programiranje, jer, u stvari, izuzetak jeste nešto što je karakteristično za određeni događaj određene klase.

Vratimo se na primer sa početka lekcije, gde smo pokušali da pročitamo nepostojeći fajl:

|  |  |
| --- | --- |
| 1  2  3 | <?php  $file = fopen("myFile.txt", "r");  ?> |

... gde smo problem nepostojećeg fajla rešili tako što smo postavili uslov ispred čitanja:

|  |  |
| --- | --- |
| 1 | if(file\_exists("myFile.txt")); |

Recimo da je sve ovo sada deo jedne klase koja rukuje fajlovima.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php   function readFunc($file)   {      if(file\_exists($file))      $file = fopen($file, "r");   }  ?> |

Poziv funkcije readFunc() ove klase neće prikazati grešku, što je očekivano. Takođe, mogli smo i ubaciti logiku koja bi izbacila poruku o nepostojanju fajla. Ali, da bi naša greška o nepostojećem fajlu bila u konvenciji izuzetaka, odnosno, da bi korisnici klase (ukoliko je izuzetak izbačen iz klase) ili mi sami mogli da je dobavimo kao standardan izuzetak, standardnim metodama za hvatanje izuzetaka, potrebno je da i objekat koji emitujemo prilikom obrade ove naše greške bude tipa izuzetak.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <?php  function read($file)   {      if(!file\_exists($file))      throw new Exception("File do not exist");  else      $file = fopen($file, "r");    }  read("myFile.txt");  ?> |

Ovako emitovanu grešku (izuzetak) lako hvatamo upotrebom try-catch bloka. Potrebno je samo da naznačimo u kom delu predviđamo događanje izuzetka (u ovom slučaju, to je poziv metode readFunc()) i stavimo ga u try blok, a zatim dodamo alternativni kod (koji treba da se izvrši ako try blok ne uspe) u catch blok. Catch blok prihvata parametar tipa Exception, koji sadrži svojstva i metode za razotkrivanje određenih detalja vezanih za sam Exception.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | try  {  read("myFile.txt");  }  catch(Exception $e)  {  echo $e;  } |

Umesto new Exceptiona, koji smo izbacili iz funkcije readFunc(), mogli smo da izbacimo i instancu sopstvene klase izuzetaka. Naravno, ukoliko smo je prethodno napravili; a možemo je napraviti [nasleđivanjem klase](https://www.link-elearning.com/linkdl/opisPojma.php?id=139117) Exception i dodavanjem svojih elemenata:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | class myException extends Exception   {   public function errorMessage()    {     return "my exception";    }   } |

|  |
| --- |
| **Napomena**  Kako ćemo se klasama, metodama i nasleđivanjem detaljnije baviti kasnije, u ovom trenutku je dovoljno samo zapamtiti gore korišćenu sintaksu. |

**Vežba 1**

***Problem:***

Data je sledeća promenljiva:

|  |  |
| --- | --- |
| 1 | $whiteList = array("index.php", "index1.php", "index2.php"); |

Potrebno je napraviti funkciju includeFile, koja će prihvatati naziv fajla, proveriti da li taj naziv postoji u nizu i da li postoji fajl sa tim imenom. Ukoliko fajl ne postoji u nizu ili na fajl sistemu, potrebno je prekinuti funkciju emitovanjem izuzetka. U suprotnom, treba učitati fajl funkcijom include ili require.

Potrebno je izvršiti poziv funkcije u try catch bloku.

***Rešenje:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <?php      $whiteList = array("index.php","index1.php","index2.php");      function includeFile($file)          {              global $whiteList;              if(!in\_array($file, $whiteList)){                  throw new Exception("No file in whitelist");              }else{                  if(!file\_exists($file)){                      throw new Exception("Filename is not valid");                  }else{                      include $file;                  }              }            }      try      {          includeFile("index1.php");      }      catch(Exception $ex)      {          echo $ex->getMessage();      }  ?> |

Promenljiva $whiteList predstavlja belu listu za naš primer. Odmah po kreiranju ovog niza pristupamo kreiranju funkcije includeFile. Funkcija kao parametar prihvata string koji predstavlja naziv fajla.  
  
Najpre se proverava da li traženi fajl ne pripada listi. Ukoliko se to dogodi, generiše se novi exception i definiše se poruka. Ako ovaj uslov ostane neispunjen, znači da je naziv fajla dozvoljen, ali treba proveriti da li se on nalazi na fajl sistemu:

|  |  |
| --- | --- |
| 1 | !file\_exists($file) |

Ukoliko je ovaj uslov ispunjen, generiše se novi exception sa novom porukom, dok se u suprotnom vrši inkludovanje fajla.  
  
Sada treba pozvati funkciju i iskontrolisati eventualne greške. Zato poziv funkcije smeštamo u try-catch blok:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | try  {      includeFile("index1.php");  }  catch(Exception $ex)  {      echo $ex->getMessage();  } |

Ukoliko je exception generisan, biće prikazana poruka koju on nosi upotrebom metode getMessage().

**Koji je broj fatalne greške koju je moguće obraditi i koja ne ostavlja trajne posledice po aplikaciju?**

 4096

 1024

 2048

 512