PIEMONTE
FONDI STRUTTURALI E DI INVESTIMENTO EUROPEI 2014/2020
fondo sociale europeo
ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

Errori in PHP e Composer

Corso Backend System Integrator

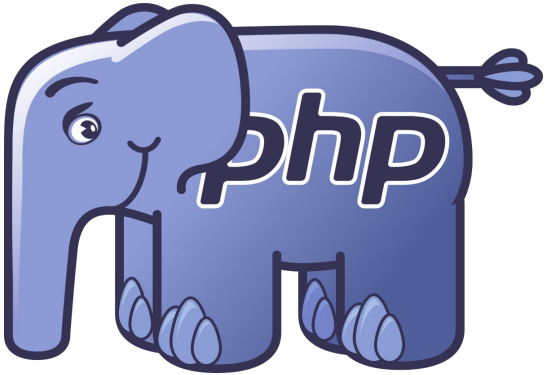
Modulo **Programmazione PHP**

Docente: Dott. Enrico Zimuel

in collaborazione con:
MIUR
UNIONE EUROPEA
REGIONE PIEMONTE
per una crescita intelligente, sostenibile ed inclusiva
www.regione.piemonte.it/europa2020
INIZIATIVA CO-FINANZIATA CON FSE

Programma

- Gestione degli errori
- Gestione di un file di log
- Eccezioni (Exception)
- Composer
- Esempio: creazione di un file di log con Monolog



Errori

- Tutti i programmi generano errori
- E’ importante imparare a gestirli e prevederli
- Ad esempio: se un programma utilizza un database, questo può risultare offline; come si comporterà l’applicazione in questo caso?

Errori generati dal PHP

- Il PHP genera diverse tipologie di errori:
  - **FATAL ERROR**, interrompono l'esecuzione di un programma, bloccanti;
  - **WARNING**, avvertimenti sono degli errori meno gravi, non bloccanti;
  - **NOTICE**, messaggi informativi per migliorare il codice, non bloccanti.

FATAL ERROR

- Quando il PHP non riesce ad eseguire il codice genera un **FATAL ERROR**
- Ad esempio:

```php
<?php
$a = new stdClass();
$a->foo();
// PHP Fatal error: Call to undefined method stdClass::foo()
// in file.php:3
echo "Hello World!\n";
```

WARNING

- Quando il PHP riscontra un problema nel codice ma riesce comunque a proseguire l'esecuzione genera un WARNING
- Ad esempio:

```
echo "Warning error";
// PHP Warning:  include(external_file.php):
// failed to open stream: No such file or directory in ...
include ("external_file.php");
```

NOTICE

- Quando il PHP riscontra un possibile problema nel codice ma riesce comunque a proseguire l'esecuzione genera un NOTICE
- Ad esempio:

```
echo $a;
// PHP Notice: Undefined variable: a in file.php on line 2
```

Error report

- E’ possibile configurare il PHP per far generare o meno una o più tipologie di errori
- I livelli sono gestiti tramite delle costanti numeriche (es. E_WARNING)
- Per impostare il livello di errore si utilizza error_reporting()

error_reporting()

```php
// Turn off all error reporting
error_reporting(0);
// Report simple running errors
error_reporting(E_ERROR | E_WARNING | E_PARSE);
// Reporting E_NOTICE can be good too (to report uninitialized
// variables or catch variable name misspellings ...)
error_reporting(E_ERROR | E_WARNING | E_PARSE | E_NOTICE);
// Report all errors except E_NOTICE
error_reporting(E_ALL & ~E_NOTICE);
// Report all PHP errors
error_reporting(E_ALL);
```

Esempio

```php
error_reporting(E_NOTICE);

include('file.php');
echo $b;
$foo = new Foo();
```

L’elenco completo delle costanti di errore in PHP è reperibile qui

Display errors

- Oltre all'impostazione del livello degli errori il PHP offre anche la possibilità di abilitare o disabilitare la visualizzare degli stessi
- Questa funzionalità è molto utile per gli ambienti di produzione dove non si vogliono fornire agli utenti informazioni messaggi d'errore del PHP, contenenti potenziali informazioni riservate (es. username, password)

Direttiva display_errors

- Si utilizza la direttiva **display_errors** con valori true (1) o false (0) per abilitare o disabilitare la visualizzazione degli errori nello standard output
- Le direttive sono dei parametri che vengono specificati nel file di configurazione **php.ini** o a runtime tramite la funzione ini_set()

```
ini_set("display_errors", 0);
```

Log degli errori

- E’ possibile configurare il PHP per memorizzare gli errori in un file di log
- Questo file di log può essere impostato con la direttiva **error_log** (vuota di default)

Esempio

```
ini_set('display_errors', 0);
ini_set('error_log', 'error.log');
$a = 0;
echo 1/$a;
```

- Gli errori vengono memorizzati nel file **error.log**:

```
[16-Sep-2021 20:31:02 Europe/Berlin] PHP Fatal error:  Uncaught
DivisionByZeroError: Division by zero
```

Exception

- Il PHP, come molti altri linguaggi, offre la possibilità di gestire gli errori con il modello delle eccezioni (Exception)
- Un’eccezione è un errore che avviene durante l’esecuzione di un’istruzione che può essere intercettato e gestito a livello applicativo
- Le eccezioni in PHP vengono gestite tramite il costrutto **try-catch**

Esempio

```php
function inverse(int $num): float
{
    if ($num === 0) {
        throw new Exception('Divisione per zero');
    }
    return 1/$num;
}
try {
    $a = inverse(0);
} catch (Exception $e) {
    printf("Errore: %s\n", $e->getMessage());
}
```

Catch multipli

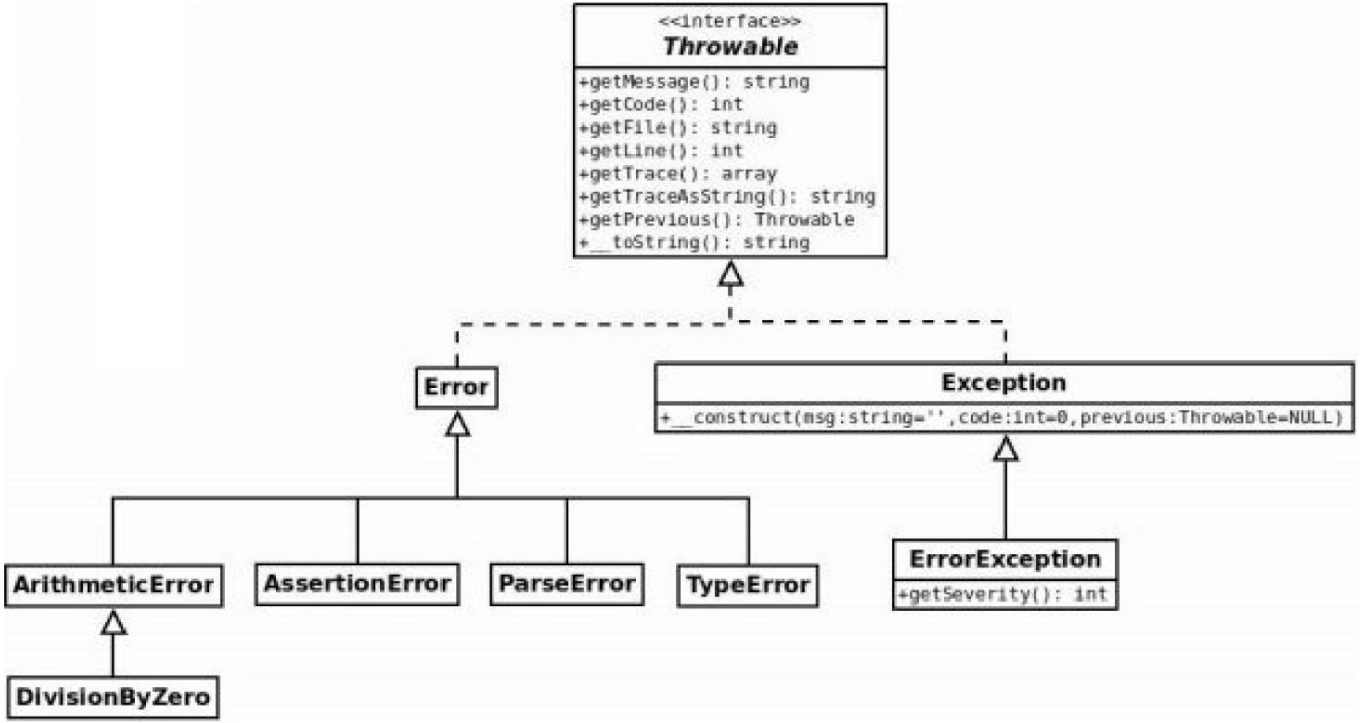
```
try {
    // Some code...
} catch (ExceptionType1 | ExceptionType2 $e) {
    // Code to handle if type1 or type2
} catch (ExceptionType3 $e) {
    // Code to handle if type3
} catch (Exception $e) {
    // Finally if not type1, type2, type3
}
```

Throwable

- Tutte le eccezioni implementano l'interfaccia Throwable

```
interface Throwable {
    public getMessage(): string
    public getCode(): int
    public getFile(): string
    public getLine(): int
    public getTrace(): array
    public getTraceAsString(): string
    public getPrevious(): ?Throwable
    abstract public __toString(): string
}
```

Throwable


<<interface>>
Throwable
+getMessage(): string
+getCode(): int
+getFile(): string
+getLine(): int
+getTrace(): array
+getTraceAsString(): string
+getPrevious(): Throwable
+__toString(): string
Error
Exception
+__construct(msg:string='',code:int=0,previous:Throwable=NULL)
ArithmeticError
AssertionError
ParseError
TypeError
ErrorException
+getSeverity(): int
DivisionByZero

Esercizio

- Riscrivere l'esercizio della **Login** (email e password) tramite una classe in PHP per la verifica delle credenziali
- Questa classe dovrà generare un’eccezione nel caso in cui le credenziali non siano valide

Composer

- **Composer** è un progetto open source per la gestione delle dipendenze con librerie di terze parti
- E’ diventato uno standard de facto in PHP
- Gestisce anche il sistema di autoloading delle classi

Composer: funzionamento

- Composer utilizza un file di configurazione: **composer.json**
- Questo file è presente nella cartella root di un progetto PHP
- In questo file vengono dichiarate le dipendenze con le librerie di terze parti
- Composer è in grado di scaricare, installare e creare un file di autoload da utilizzare per caricare queste classi con il meccanismo di autoloading

composer.json

```
{
    "require" : {
        "php": ">=8.0",
        "monolog/monolog": "^2.6"
    }
}
```

Nel file **composer.json** è specificato che:

- il progetto ha bisogno del PHP 8.0 o superiore
- il progetto utilizza la libreria Monolog a partire dalla versione 2.6

Semantic versioning

- Il semantic versioning è una notazione per assegnare una versione a un software
- Utilizza la sintassi **MAJOR.MINOR.PATCH** (es. 2.5.11)
  - **MAJOR** (es. 2), nuove funzionalità con BC break
  - **MINOR** (es. 5), nuove funzionalità senza BC break
  - **PATCH** (es. 11), bug fix
- Maggiori informazioni: https://semver.org/

Packagist

- Come fa composer a installare le librerie di terze parti?
- Le scarica da internet tramite i repository packagist.org e github.com
- 343k librerie, 3.4 milioni di versioni e più di 60 miliardi di installazioni (dal 2012)

Comandi di composer

- Se si ha già un **composer.json** nel progetto è sufficiente eseguire il seguente comando per installare le dipendenze:

```
composer install
```

- Se volete aggiungere una dipendenza, es. la libreria Monolog:

```
composer require monolog/monolog
```

vendor

- Quando il composer installa le dipendenze crea un file **vendor/autoload.php** per il caricamento automatico delle classi (autoloading)
- E' sufficiente includere questo file all'inizio di uno script PHP per poter utilizzare tutte le librerie installare da composer senza dover fare nessun require() o include()
- Le librerie di terze parti vengono installate nella cartella **vendor**
- Viene creato un file **composer.lock** nella root del progetto con l'elenco di tutte le librerie installate e le relative versioni

Esempio

```php
require 'vendor/autoload.php';
use Monolog\Logger;
use Monolog\Handler\StreamHandler;

$log = new Logger('esempio');
$log->pushHandler(new StreamHandler('file.log', Logger::WARNING));
$log->warning('Foo');
$log->error('Bar');
```

file.log

```
[2022-06-05T21:31:44.522329+02:00] esempio.WARNING: Foo [] []
[2022-06-05T21:31:44.523229+02:00] esempio.ERROR: Bar [] []
```

Autoloading delle classi

- Oltre a generare l'autoloading per le librerie installate nella directory vendor, composer può anche gestire il caricamento delle classi di progetto
- E' possibile aggiungere nel composer.json la chiave **autoload** specificando il percorso e la modalità di autoloading da utilizzare (es. PSR-4)

Esempio

```
{
    "autoload": {
        "psr-4": {"App\\": "src/"}
    }
}
```

App è il namespace associato alle mie classi e i file si trovano nella cartella **src**

PSR-4

- Il namespace individua la cartella dove sono memorizzati i file
- Ogni classe deve essere memorizzata in un unico file il cui nome deve coincidere al nome della classe
- I file sono memorizzati seguendo il percorso dei namespace
- Es. utilizzando l'esempio precedente di **PSR-4**, la classe **App\Controller\User** si troverà nella cartella **src/Controller** e il file sarà **User.php**

Composer update

E' possibile aggiornare le dipendenze tramite il comando:

```
composer update
```

Per aggiornare solo la sezione di autoloading è possibile utilizzare il comando:

```
composer dump-autoload
```

Esercizio (da consegnare)

- Utilizzare la libreria per memorizzare in un file di log gli accessi ad una pagina di Login (vedi esercizi precedenti)
- In particolare nel file di log dovranno essere memorizzati gli accessi con l'email dell'utente loggato e gli insuccessi, nel caso di errore delle credenziali
- Per il file di log è necessario utilizzare la libreria Monolog

Grazie dell’attenzione!

Per informazioni:
enrico.zimuel@its-ictpiemonte.it

POR Piemonte
FSE 2014-2020