UNIT – 7
EXCEPTION HANDLING

UNIVERSITY
OF PETROLEUM
& ENERGY STUDIES

Dr. P S V S Sridhar

Assistant Professor (SS)

Centre for Information Technology

- **Unit 7*:* Exception Handling**

  (1) Introduction to Exception, Exception Handling mechanisms

  (2) Creating Custom Exceptions

  (3) Multiple Catch Blocks

  (4) Exception Propagation, Error Handling in PHP

Introduction to Exception

- An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
- It can also be defined as abnormal event that arises during the execution of a program.
- Exceptions give us much better handling of errors and allow us to customize the behavior of our scripts when an error (Exception) is encountered.
- An exception is not an error.
- An exception, as the name implies, is any condition experienced by your program that is unexpected or not handled within the normal scope of your code.

Try, throw and catch

- To avoid the error in the execution of the code, we need to create the proper code to handle an exception.
- Proper exception code should include:
- **Try** - A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown“.
- **Throw** - This is how you trigger an exception. Each "throw" must have at least one "catch“. The throw is used to throw an exception explicitly.
- **Catch** - A "catch" block retrieves an exception and creates an object containing the exception information.

Exception Handling Mechanism

- Exceptions are caught and handled using a try/catch control construct.
- The first step in constructing an exception handler is to enclose the code that might throw an exception within a try block.

```
try
{
statements;
}
```

If an exception occurs, it is passed to the catch block. The catch block will contain a code to handle the exception.

```
catch(Exceptiontype name)
{
statements;
}
```

The catch block should come immediately after the try block.

Exception Handling Mechanism

```
<?php
try
{
        throw new Exception("Some error message");
}
catch(Exception $e)
{
   echo $e->getMessage();
}
?>
```

Exception Handling Mechanism

```
<?php

//create function with an exception

function checkNum($number)

{

  if($number>1)

  {

    throw new Exception("Value must be 1 or below");

  }

  return true;

  }
```

```
//trigger exception in a "try" block
try
{
  checkNum(2);
  //If the exception is thrown, this text will not be shown
 echo 'If you see this, the number is 1 or below';
  }

//catch exception
catch(Exception $e)
{
  echo 'Message: ' .$e->getMessage();
}
?>
```

getMessage(): **Exception::getMessage — Gets the Exception message**

Exception Handling Mechanism

```
<?php
//create function with an exception
function checkNum($number)
{
if($number>1)
{
throw new Exception("Value must be 1 or below");
}
return true;
}
//trigger exception in a "try" block
try
{
checkNum(2);
//If the exception is thrown, this text will not be shown
echo 'If you see this, the number is 1 or below';
}
//catch exception
catch(Exception $e)
{
echo 'Message: ' .$e->getMessage();
}
?>
```

The code above throws an exception and catches it:

1. **The checkNum() function is created.**
2. **It checks if a number is greater than 1. If it is, an exception is thrown**
3. **The checkNum() function is called in a "try" block**
4. **The exception within the checkNum() function is thrown**
5. **The "catch" block retrives the exception and creates an object ($e) containing the exception information**
6. **The error message from the exception is echoed by calling $e->getMessage() from the exception object**

getMessage(): Exception::getMessage — Gets the Exception message

Exception Handling Mechanism

```
try
{
echo "Current Date<br><br>";
$dateTime = new DateTime("now", new DateTimeZone('sample/Calcutta'));
echo $dateTime->format("d-m-y");
}
catch(Exception $e)
{
echo "provide a valid time zone";
}
```

- Here we get an exception, an exception object is created and it is thrown to the corresponding catch block.
- Exception is a built-in class in PHP.

Creating Custom Exceptions

- can create own customized exception as per requirements of the application.

```
<?php
class NegativeageException extends Exception
{
private $info;
public function __construct($message)
{
$this->info=$message;
}
public function getInfo()
{
return $this->info;
}
}
$stuname="john";
$age=-23;
$mark=56;
```

```
try
{
if($age<=0)
{
throw new NegativeageException("Age
can't be negative");
}
}
catch(NegativeageException $n)
{
echo $n->getInfo();
}
?>
```

The throw is used to throw an exception explicitly.

Multiple Catch Blocks

- It is possible for a script to use multiple exceptions to check for multiple conditions.
-
```
$stuname="john";
$age=23;
$mark=400;
try
{
if($age<=0)
{
throw new NegativeageException("Age can't be negative");
}
if($mark>100)
{
throw new Exception();
}
}
catch(NegativeageException $n)
{
echo $n->getInfo();
}
catch(Exception $e)
{
echo "The marks should range from 0 to 100";
}
```

Exception Propagation

- When an exception occurs and if it is not caught by any of the catch blocks, it looks for its handler and if it fails to find one it just propagates from one method to the other and ends up crashing the program.

```
function displaydate()
{
try
{
echo "Current Date<br><br>";
$dateTime = new DateTime("now", new DateTimeZone('sample/Calcutta'));
echo $dateTime->format("d-m-y");
}
catch(MyException $e)
{
}
}
try
{
echo "Today's date"."<br>";
echo displaydate();
}
catch(Exception $n)
{
echo "provide a valid time zone";
}
```

Exception re throw

- Sometimes, when an exception is thrown, you may wish to handle it differently than the standard way. It is possible to throw an exception a second time within a "catch" block.
- A script should hide system errors from users. System errors may be important for the coder, but is of no interest to the user. To make things easier for the user you can re-throw the exception with a user friendly message:

```
<?php
class customException extends Exception
{
public function errorMessage()
{
//error message
$errorMsg = $this->getMessage().' is not a valid E-Mail address.';
return $errorMsg;
}
}

$email = "someone@example.com";
```

Exception re throw

```
try
{
try
{
//check for "example" in mail address
if(strpos($email, "example") !== FALSE)
{
//throw exception if email is not valid
throw new Exception($email);
}
}
catch(Exception $e)
{
//re-throw exception
throw new customException($email);
}
}

catch (customException $e)
{
//display custom message
echo $e->errorMessage();
}
?>
```

Error Handling in PHP

- Errors are the most common event a developer faces when programming. To reduce the number of errors in your code, proper error handling is essential in your web application. PHP generates three types of errors, depending on severity:
- **Notice:** These errors are not serious and do not create a serious problem. Notices are PHP‟s way to tell you that the code it runs may be doing something unintentional, such as reading that undefined variable.
- **Warning:** Failed code has created an error, but does not terminate execution. Typical examples are missing function parameters, when you include a file that does not exists.
- **Fatal error:** A serious error condition has rendered the script unable to run. A fatal error terminates the script. Examples are out-of-memory errors, uncaught exceptions, or class declaration.

Error Handling in PHP

- Each type of error is also represented by a constant that can be referred to within your code as E_USER_NOTICE, E_USER_WARNING, and E_USER_ERROR. The error-reporting level can be manually defined within a script using the error_reporting() function.
- To display all types of error including notices;

  error_reporting(E_ALL);

- To display only fatal errors;

  error_reporting(E_ USER_ERROR);

Error Handling in PHP

```
error_reporting(E_USER_ERROR);
$fp=fopen("employeedetails.dat","r");
if($fp)
{
echo "Reading";
}
else
{
echo "File does not exist";
}
```

If "error_reporting" statement is not available the following message will be displayed.

Warning: fopen(employeedetails.dat) [function.fopen]: failed to open stream: No such file or directory
in C:\xampp\htdocs\user_error.php on line 3
File does not exist

Defining an error handler

- We can write our own function to handle any error.
- By creating a function that designs a custom error message and then setting that function as the default error handler, you can avoid the awkward and unprofessional display of errors to a user.
- This function must be able to handle a minimum of two parameters (error level and error message).

```
function error_handler($errno, $errmessage)
{
echo "Sorry for the inconvenience, Please try again later!!!!!!!!";
}
```

we have defined our custom error handler, we simply need to refer the function within the code using the set_error_handler() function.

```
set_error_handler("error_handler");
```

Defining an error handler

```
<?php
function error_handler($errno, $errmessage)
{
echo "Sorry for the inconvenience, Please try again later!!!!!!!!";
}
set_error_handler("error_handler");
include("LoanDetails.html");
?>
```

the above code if the file does not exist we will get a message „Sorry for the inconvenience, Please try again later!!!!!!!"

Thanks