



# HiLCoE School of Computer Science & Technology

## Chapter Three: Advanced Topics In PHP

**Course Title : Web Technologies II**

**Instructor name: Yitayew Solomon**

**E-mail address: [yitayewsolomon3@gmail.com](mailto:yitayewsolomon3@gmail.com)**

# PHP Exceptions



## PHP Exceptions: Explanation with Examples

Exceptions in PHP are used to handle errors in a controlled way. Instead of stopping the execution of the script when an error occurs, exceptions allow you to catch and handle the error gracefully. This makes your code more robust and prevents unexpected terminations.

### What is an Exception?

An **exception** is an object that describes an error or unexpected behavior in a program. When an error occurs, PHP will create an object that represents the exception. This object contains information about the error, like the error message, code, and location (file and line number).

# Cont. ...

## Exception Handling in PHP


In PHP, exceptions are handled using the following keywords:

- `try`: Contains the code that may throw an exception.
- `catch`: Catches and handles the exception.
- `throw`: Used to trigger an exception.
- `finally`: (optional) Contains code that will always execute, regardless of whether an exception was thrown or not.

# Cont. ...

## Basic Structure of Exception Handling

php

 Copy code

```
try {  
    // Code that may throw an exception  
} catch (Exception $e) {  
    // Code to handle the exception  
} finally {  
    // Code that will always execute, regardless of exceptions (optional)  
}
```

# Throwing an Exception

```
C:\xampp\htdocs\webTechnology_2\Exceptions_Date\Exception.php - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Exception.php x
1 <?php
2 // Function that checks the age and throws an exception if the condition is not met
3 function checkAge($age) {
4     if ($age < 18) {
5         // Throw an exception if age is less than 18
6         throw new Exception("Age must be 18 or older.");
7     } else {
8         echo "Age is valid.";
9     }
10 }
11
12 // Try block to handle the exception
13 try {
14     checkAge(16); // This will trigger the exception
15 } catch (Exception $e) {
16     // Catch block to handle the exception
17     echo "Error: " . $e->getMessage();
18 }
19 ?>
20
```

Error: Age must be 18 or older.

## Cont. ...

### Explanation:

- The `checkAge()` function checks whether the input age is at least 18. If not, it throws an exception.
- The `try` block contains the code that might cause an exception ( `checkAge(16)` ).
- The `catch` block catches the exception and displays an error message.

# Example 2: Using Multiple catch Blocks

C:\xampp\htdocs\webTechnology\_2\Exceptions\_Date\multiple\_catch.php - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

multiple\_catch.php

```
1 <?php
2 class CustomException extends Exception {}
3
4 try {
5     $value = 0;
6
7     if ($value == 0) {
8         throw new CustomException("Custom exception: Value cannot be zero.");
9     }
10 } catch (CustomException $e) {
11     echo "Caught Custom Exception: " . $e->getMessage();
12 } catch (Exception $e) {
13     echo "Caught Default Exception: " . $e->getMessage();
14 }
15 ?>
```

Caught Custom Exception: Custom exception: Value cannot be zero.

## Cont. ...


### Explanation:

- The custom exception `CustomException` is defined by extending the base `Exception` class.
- The code throws a `CustomException` if `$value` is zero, and this is caught by the specific `catch` block for `CustomException`.
- If another type of exception was thrown, the general `catch` block for `Exception` would handle it.

### Example 3: Using `finally` Block

The `finally` block will always execute, whether or not an exception was thrown.

php

 Copy code



# Cont. ...

```
C:\xampp\htdocs\webTechnology_2\Exceptions_Date\final_block.php - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

final_block.php

1  <?php
2  function divide($a, $b) {
3      if ($b == 0) {
4          throw new Exception("Division by zero.");
5      }
6      return $a / $b;
7  }
8
9  try {
10     echo divide(10, 0); // This will throw an exception
11 } catch (Exception $e) {
12     echo "Error: " . $e->getMessage();
13 } finally {
14     echo "\nCleaning up..."; // This will always run
15 }
16 ?>
17
```

# Cont. ...

## Explanation:

- In this example, if the second parameter of the `divide()` function is zero, an exception is thrown.
- The `finally` block will always execute, printing "Cleaning up..." regardless of whether an exception occurred.

## Custom Exception Class

You can create your own exception classes by extending the built-in `Exception` class. This allows you to define custom behaviors and error messages.

# Cont. ...

```
C:\xampp\htdocs\webTechnology_2\Exceptions_Date\Custom_Exception.php - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

Custom_Exception.php
1 <?php
2 // Custom Exception Class
3 class InvalidInputException extends Exception {
4     public function errorMessage() {
5         // Error message
6         return "Error on line " . $this->getLine() . " in " . $this->getFile()
7             . ": " . $this->getMessage();
8     }
9 }
10
11 try {
12     $input = "invalid";
13
14     if ($input != "valid") {
15         // Throw custom exception
16         throw new InvalidInputException("Invalid input provided.");
17     }
18 } catch (InvalidInputException $e) {
19     // Display custom error message
20     echo $e->errorMessage();
21 }
22 ?>
```

Error on line 16 in C:\xampp\htdocs\webTechnology\_2\Exceptions\_Date\Custom\_Exception.php: Invalid input provided.

# Cont. ...

## Explanation:

- `InvalidInputException` extends the `Exception` class and overrides the `errorMessage()` method to provide a custom error message.
- When the input is not "valid," the custom exception is thrown and handled, displaying the custom error message.

## Benefits of Exception Handling

- **Graceful error management:** Exceptions allow your application to handle errors without crashing.
- **Separation of concerns:** Exception handling separates the logic of handling errors from the regular code logic.
- **Improved debugging:** Exception objects contain detailed information about the error, making debugging easier.



# Cont. ...

## Methods

When catching an exception, the following table shows some of the methods that can be used to get information about the exception:

Method	Description
getMessage()	Returns a string describing why the exception was thrown
getPrevious()	If this exception was triggered by another one, this method returns the previous exception. If not, then it returns <i>null</i>
getCode()	Returns the exception code
getFile()	Returns the full path of the file in which the exception was thrown
getLine()	Returns the line number of the line of code which threw the exception

# PHP Object-Oriented Programming (OOP)



## PHP Object-Oriented Programming (OOP)

PHP supports Object-Oriented Programming (OOP) principles, allowing developers to create objects that encapsulate both data (attributes) and functionality (methods). OOP in PHP enables modular, reusable, and efficient code by organizing the program into objects.

---

# Basics Of OOP in PHP

## 1. Key Concepts in OOP

- **Class:** A blueprint for creating objects. It defines properties (attributes) and methods (functions).
- **Object:** An instance of a class. When a class is defined, objects can be created from it.
- **Inheritance:** The ability of a class to inherit properties and methods from another class.
- **Encapsulation:** Restricting access to the internal state of an object and exposing only necessary functionality.
- **Polymorphism:** The ability to define methods that behave differently depending on the context (overriding or overloading methods).
- **Abstraction:** Hiding the implementation details and showing only the necessary features of an object.

# Class in PHP

## 1. Class in PHP

A **class** is a blueprint or template for creating objects. It defines the structure (properties) and behavior (methods) that the objects instantiated from it will have. Think of a class as a recipe, while an object is the actual dish made from that recipe.

### Key Components of a Class:

- **Properties:** Variables that store the object's data (also known as attributes or fields).
- **Methods:** Functions that define the behavior of the object (what actions it can perform).
- **Access Modifiers:** Keywords like `public`, `private`, and `protected` that control the visibility and accessibility of properties and methods.




# Example

## Creating a Class

A class in PHP is defined using the `class` keyword, and it contains properties and methods.

php

 Copy code

```
<?php
class Car {
    // Properties
    public $make;
    public $model;

    // Constructor method to initialize properties
    public function __construct($make, $model) {
        $this->make = $make;
        $this->model = $model;
    }

    // Method to display car details
    public function display() {
        return "This car is a " . $this->make . " " . $this->model;
    }
}
?>
```



# Object in PHP

## 2. Object in PHP

An **object** is an instance of a class. Once a class is defined, objects can be created using the `new` keyword. Objects inherit the properties and methods defined by their class. Each object can have its own values for the class's properties, making them unique.

### Key Concepts:


- **Instantiation:** The process of creating an object from a class.
- **Object State:** The current values of the object's properties.
- **Object Behavior:** The actions or operations (methods) that an object can perform.

# Example

## Creating an Object

An **object** is created by instantiating a class using the `new` keyword.

php

 Copy code

```
<?php
// Creating an object of the Car class
$myCar = new Car("Toyota", "Corolla");

// Accessing the object's method
echo $myCar->display(); // Output: This car is a Toyota Corolla
?>
```

In this example:

- The class `Car` defines two properties (`make` and `model`) and a method (`display()`).
- The constructor method (`__construct()`) is used to initialize the object's properties when it's created.



# Cont. ...

## 3. Key Characteristics of Classes and Objects

- **Encapsulation:** Bundling the data (properties) and the functions (methods) that manipulate the data into a single unit (the object). This helps protect the internal state of the object by restricting access using access modifiers.
- **Reusability:** Classes can be reused to create multiple objects, each having its own state (data) while sharing the same behavior (methods).
- **Modularity:** By breaking down a program into smaller, self-contained classes, OOP promotes cleaner, more modular code, which is easier to maintain and extend.

# Cont. ...

## 4. Relationship Between Class and Object

- **Class:** Defines the **structure** (properties) and **behavior** (methods) that all objects of that type will share.
- **Object:** Represents an **instance** of a class, having its own unique values for the class's properties.

An analogy would be:

- **Class:** A blueprint for a house.
- **Object:** A specific house built from that blueprint.

Each house (object) may have different colors or interiors (values of properties), but they are all built from the same blueprint (class).

# PHP OOP - Constructor

## PHP OOP - Constructor


In Object-Oriented Programming (OOP), a **constructor** is a special function within a class that is automatically called when an object is created. It is used to initialize object properties or perform any setup tasks required when the object is instantiated.

### Syntax of a Constructor in PHP

In PHP, the constructor is defined using the `__construct()` method.

# Cont. ...

php

 Copy code

```
class ClassName {  
    // Properties  
    public $property1;  
    public $property2;  
  
    // Constructor method  
    public function __construct($param1, $param2) {  
        $this->property1 = $param1;  
        $this->property2 = $param2;  
    }  
}
```

- The `__construct()` method is called automatically when an object of the class is created.
- The constructor often accepts parameters to set initial values for object properties.

# Cont. ...

```
C:\xampp\htdocs\webTechnology_2\Constructor.php - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

Constructor.php x

1 <?php
2 class Car {
3     public $brand;
4     public $model;
5
6     // Constructor to initialize properties
7     public function __construct($brand, $model) {
8         $this->brand = $brand;
9         $this->model = $model;
10    }
11
12    // Method to display car details
13    public function getCarDetails() {
14        return "Brand: " . $this->brand . ", Model: " . $this->model;
15    }
16 }
17
18 // Creating a new object of the Car class
19 $car1 = new Car("Toyota", "Corolla");
20 echo $car1->getCarDetails(); // Output: Brand: Toyota, Model: Corolla
21 ?>
```

Brand: Toyota, Model: Corolla



# Cont. ...

## Explanation:

- The `Car` class has two properties: `brand` and `model`.
- The constructor `__construct()` is used to set the values of `brand` and `model` when a new object is created.
- When the object `$car1` is instantiated, the constructor is automatically called, initializing the properties with "Toyota" and "Corolla".
- The method `getCarDetails()` returns the car's brand and model.

## Example 2: Constructor with Default Values

Constructors can also have default parameter values. If no value is passed during object creation, the default value will be used.

# Cont. ...

```
C:\xampp\htdocs\webTechnology_2\Constructor_Default_Value.php - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Constructor_Default_Value.php x
1 <?php
2 class Bike {
3     public $brand;
4     public $model;
5
6     // Constructor with default values
7     public function __construct($brand = "Honda", $model = "CBR") {
8         $this->brand = $brand;
9         $this->model = $model;
10    }
11
12    // Method to display bike details
13    public function getBikeDetails() {
14        return "Brand: " . $this->brand . ", Model: " . $this->model;
15    }
16 }
17
18 // Creating a new object with default values
19 $bike1 = new Bike();
20 echo $bike1->getBikeDetails(); // Output: Brand: Honda, Model: CBR
21
22 // Creating a new object with specified values
23 $bike2 = new Bike("Yamaha", "R1");
24 echo $bike2->getBikeDetails(); // Output: Brand: Yamaha, Model: R1
25 ?>
```

Brand: Honda, Model: CBRBrand: Yamaha, Model: R1

# Cont. ...

## Explanation:

- The constructor has default values for the `brand` and `model` properties ( `Honda` and `CBR` ).
- If no values are provided when creating the object, the default values are used.
- In `$bike1`, the default constructor values are used, while in `$bike2`, specific values are passed during object creation.

## Constructor Overloading

PHP does not support method overloading (having multiple constructors with different signatures).

However, you can achieve similar behavior by using default values or checking the number of arguments within the constructor.

# Cont. ...

```
C:\xampp\htdocs\webTechnology_2\Constructor_Overloading.php - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Constructor_Overloading.php x
1 <?php
2 class Computer {
3     public $brand;
4     public $model;
5
6     // Constructor with logic to handle different arguments
7     public function __construct($brand = null, $model = null) {
8         if ($brand !== null && $model !== null) {
9             $this->brand = $brand;
10            $this->model = $model;
11        } else {
12            $this->brand = "Dell";
13            $this->model = "XPS";
14        }
15    }
16 }
```

# Cont. ...

```
C:\xampp\htdocs\webTechnology_2\Constructor_Overloading.php • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

Constructor_Overloading.php •

17 // Method to display computer details
18 public function getComputerDetails() {
19     return "Brand: " . $this->brand . ", Model: " . $this->model;
20 }
21 }
22
23 // Creating object with no arguments (uses default values)
24 $computer1 = new Computer();
25 echo $computer1->getComputerDetails(); // Output: Brand: Dell, Model: XPS
26
27 // Creating object with arguments
28 $computer2 = new Computer("Apple", "MacBook Pro");
29 echo $computer2->getComputerDetails(); // Output: Brand: Apple, Model: MacBook Pro
30 ?>
31
```

# PHP OOP - Destructor

## PHP OOP - Destructor


In Object-Oriented Programming (OOP), a **destructor** is a special function that is automatically called when an object is destroyed or when the script ends. It is mainly used to perform any cleanup tasks, such as closing database connections, freeing up resources, or performing any other actions before the object is completely removed from memory.

### Syntax of a Destructor in PHP

In PHP, a destructor is defined using the `__destruct()` method. Unlike constructors, destructors do not take any arguments.

# Cont. ...

php

 Copy code

```
class ClassName {  
    // Destructor method  
    public function __destruct() {  
        // Cleanup code  
    }  
}
```

- The `__destruct()` method is called automatically when:
  - The object is no longer referenced.
  - The script ends.
  - `unset()` is called on the object.

# Cont. ...

```
C:\xampp\htdocs\webTechnology_2\Destructor.php - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

Destructor.php x
1 <?php
2 class Car {
3     public $brand;
4     public $model;
5
6     // Constructor to initialize properties
7     public function __construct($brand, $model) {
8         $this->brand = $brand;
9         $this->model = $model;
10        echo "Car is created: " . $this->brand . " " . $this->model . "<br>";
11    }
12
13    // Destructor to perform cleanup tasks
14    public function __destruct() {
15        echo "Car is destroyed: " . $this->brand . " " . $this->model . "<br>";
16    }
17 }
18
19 // Creating a new object of the Car class
20 $car1 = new Car("Toyota", "Corolla");
21 // Destructor is called automatically at the end of the script
22 ?>
```

Car is created: Toyota Corolla  
Car is destroyed: Toyota Corolla



## Cont. ...

### Explanation:

- The constructor initializes the `brand` and `model` properties and prints a message when a new `Car` object is created.
- The destructor prints a message when the object is destroyed, either at the end of the script or when it is no longer referenced.

### Example 2: Destructor with Cleanup Tasks

In a real-world scenario, destructors can be used to perform cleanup tasks such as closing files or database connections.

# Cont. ...

C:\xampp\htdocs\webTechnology\_2\Destructor\_Cleanup.php - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Destructor\_Cleanup.php

```
1 <?php
2 class FileHandler {
3     private $file;
4
5     // Constructor to open a file
6     public function __construct($filename) {
7         $this->file = fopen($filename, "w");
8         echo "File opened: " . $filename . "<br>";
9     }
10
11    // Destructor to close the file
12    public function __destruct() {
13        if ($this->file) {
14            fclose($this->file);
15            echo "File closed.<br>";
16        }
17    }
18 }
19
20 // Creating a new FileHandler object
21 $fileHandler = new FileHandler("example.txt");
22
23 // Destructor will automatically close the file when the script ends
24 ?>
```

File opened: example.txt  
File closed.

## Cont. ...


### Explanation:

- The constructor opens a file using `fopen()`.
- The destructor automatically closes the file using `fclose()` when the object is destroyed, ensuring that resources are properly freed.

### Example 3: Manually Destroying an Object with `unset()`

You can manually destroy an object using the `unset()` function, which triggers the destructor.

php

 Copy code

# Cont. ...

```
C:\xampp\htdocs\webTechnology_2\Destructor_Manual.php - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

Destructor_Manual.php
1 <?php
2 class Bike {
3     public $brand;
4
5     // Constructor to initialize the brand
6     public function __construct($brand) {
7         $this->brand = $brand;
8         echo "Bike is created: " . $this->brand . "<br>";
9     }
10
11    // Destructor to clean up
12    public function __destruct() {
13        echo "Bike is destroyed: " . $this->brand . "<br>";
14    }
15 }
16
17 // Creating a new object of the Bike class
18 $bike1 = new Bike("Yamaha");
19
20 // Manually destroying the object
21 unset($bike1); // Destructor is called here
22
23 echo "Script continues...<br>";
24 ?>
```

# Cont. ...

## Explanation:

- The object `$bike1` is manually destroyed using `unset()`, which triggers the destructor immediately.
- After the object is destroyed, the script continues to execute.

## Key Points about Destructors in PHP:

1. **Automatic Invocation:** Destructors are called automatically at the end of a script or when an object is no longer in use.
2. **No Parameters:** Destructors do not take any arguments.
3. **Manual Invocation:** You can trigger the destructor manually by using `unset()` to destroy the object before the script ends.
4. **Use for Cleanup:** Destructors are often used for cleanup tasks like closing file handles, database connections, or releasing other resources.

# Thank you!

Appreciate your action.