B.C.A. Semester – 4

BCA-402
Building Application Using PHP

UNIT - 2

Object oriented Programming with PHP and Error Handling

Outline

I. Introduction to Object Oriented Programming with PHP

- A. Basic PHP construction for OOP
- B. Advanced OOP features

II. File and File System Functions

III. Session and Cookies

IV. Error Handling and Debugging

- A. General error types and debugging
- B. Displaying PHP errors
- C. Adjusting Error Reporting
- D. Creating Custom Error Handler
- E. PHP debugging techniques

I. Introduction to Object Oriented Programming

Object-oriented programming (OOP) is a programming paradigm that revolves around the concept of objects. Objects are defined as collections of data and methods that act on that data, and they are used to create and maintain complex systems.

OOP allows developers to create objects that can interact with each other and share data, making it easier to develop, maintain, and debug applications. OOP languages also provide features such as encapsulation, polymorphism, and inheritance, which allow developers to write code that is more organized, efficient, and secure.

Basic PHP construction for OOP

A basic PHP OOP construction would include:

- **Class**: A class is a blueprint for creating objects. It defines the properties and methods that an object of that class will have. Classes can be thought of as data types that describe the structure of objects.
- **Objects**: An object is an instance of a class. It is created by using the "new" keyword and provides access to the properties and methods defined in the class.
- **Parent class and Child class:** A parent class is a class that is used as the basis for creating a child class. A child class inherits the properties and methods of the parent class, and can also have its own properties and methods. This is known as inheritance.
- **Member variable:** A member variable is a variable that is defined within a class. It represents a property of the object.

- **Member function:** A member function is a function that is defined within a class. It represents a method that the object can perform.
- **Inheritance**: Inheritance is a mechanism in OOP that allows a child class to inherit the properties and methods of a parent class. This allows for code reuse and organization.
- **Polymorphism**: Polymorphism is a feature of OOP that allows an object to take on different forms depending on the context. This is achieved through the use of method overloading and method overriding.
- **Overloading**: Overloading is a feature of OOP that allows a method to have multiple definitions, depending on the parameters passed to it.
- **Data abstraction:** Data abstraction is a technique in OOP that allows developers to define the structure of an object without revealing its implementation details. This promotes code modularity and security.
- **Encapsulation**: Encapsulation is a technique in OOP that wraps the data and behavior of an object into a single unit, or capsule. This promotes code reusability and reduces code complexity.
- **Constructor**: A constructor is a special type of method in OOP that is automatically called when an object is created. It can be used to initialize the properties of an object.
- **Destructor**: A destructor is a special type of method in OOP that is automatically called when an object is destroyed. It can be used to release resources that are no longer needed.

These are the basic components of a PHP OOP construction, including class definition, objects, inheritance, polymorphism, data abstraction, encapsulation, constructor, and destructor. Understanding these concepts is essential for creating complex applications using OOP in PHP.

Defining a PHP Class:

Syntax: The syntax for defining a PHP class is as follows:

```
class ClassName {
// properties
// methods
}
```

Example: Consider the following example of a class for a car:

```
<?php
```

```php
class Car
{
    // member variables
    public $color;
    public $brand;
    public $model;

    // member functions
    public function startEngine()
    {
        // code to start the engine
    }

    public function accelerate()
    {
        // code to accelerate the car
    }
}
?>
```

In this example, the Car class has two member variables: $color, $brand, and $model. It also has two member functions: startEngine() and accelerate().

Inheritance can be used to create a **child class** for a supercar, for example:

```php
<?php
class Animal
{
  protected $name;
  protected $age;
  public function __construct($name, $age)
  {
    echo "The animal named " . $name . " has been created.\n";
    $this->name = $name;
    $this->age = $age;
  }
  public function eat()
  {
    echo "Nom nom nom\n";
  }
}
class Dog extends Animal
{
  public function eat() # Overriding the eat() method of the parent class.
```

```
    {
      echo "Mmm, this is some tasty dog food!";
    }
    public function bark()
    {
      echo "Woof woof!";
    }
}
$my_dog = new Dog("Tomy", 3);
$my_dog→eat(); // Output: "Nom nom nom"
$my_dog→bark(); // Output: "Woof woof!"
```

Definition of Objects:

Definition: An object is an instance of a class. It represents a specific instance of an object and provides access to the properties and methods defined in the class.

Syntax: The syntax for creating an object in PHP is as follows:

$object = new ClassName();

To create an object of the Car class, we use the following code:

$myCar = new Car();

In this example, we have created an object called $myCar that is an instance of the Car class. We can now access the member variables and methods of the Car class through the $**myCar** object.

File and File system Function in PHP

Introduction

File handling is an important part of any programming language, including PHP. In this chapter, we will cover the basics of file handling in PHP, including how to create, open, read, write, and close files. We will also discuss the PHP file system functions, which allow you to perform operations on files and directories, such as creating, deleting, and renaming files and

directories.

Creating and Opening Files:

To create a new file in PHP, you can use the ***fopen()*** function. The function takes two parameters: the first is the name of the file, and the second is the mode in which the file will be opened. Here is an example:

```
$file = fopen("example.txt", "w");
```

In this example, we are creating a new file called "***example.txt***" and opening it in write mode. The **"w"** mode means that the file will be opened for writing, and any existing contents will be erased.

If you want to open an existing file instead of creating a new one, you can use the same function but with a different mode. Here is an example:

```
$file = fopen("example.txt", "r");
```

In this example, we are opening the file "example.txt" in read mode. The "r" mode means that the file will be opened for reading, and you will not be able to write to it.

Reading and Writing Files

Once you have opened a file in PHP, you can read from it or write to it. To read from a file, you can use the fgets() function, which reads a single line from the file. Here is an example:

```
$file = fopen("example.txt", "r");
echo fgets($file);
```

In this example, we are opening the file "example.txt" in read mode and reading the first line of the file using the fgets() function. The line is then printed to the screen using the echo statement.

To write to a file, you can use the fwrite() function. Here is an example:

```
$file = fopen("example.txt", "w");
fwrite($file, "This is a test.");
```

In this example, we are opening the file "example.txt" in write mode and writing the string "This is a test." to the file using the fwrite() function.

Closing Files

It is important to close a file after you are done reading from it or writing to it. To close a file in PHP, you can use the fclose() function. Here is an example:

```
$file = fopen("example.txt", "r");
echo fgets($file);
```

```
fclose($file);
```

In this example, we are opening the file "example.txt" in read mode, reading the first line of the file using the fgets() function, and then closing the file using the fclose() function.

PHP Open File Mode

Mode	Description
r	Read only. Starts at the beginning of the file.
r+	Read/write. Starts at the beginning of the file.
w	Write only. Opens and truncates the file; or creates a new file if it doesn't exist.
w+	Read/write. Opens and truncates the file; or creates a new file if it doesn't exist.
a	Write only. Opens and writes to the end of the file; or creates a new file if it doesn't exist.
a+	Read/write. Preserves file content by writing to the end of the file; or creates a new file if it doesn't exist.
x	Write only. Creates a new file. Returns FALSE and an error if the file already exists.
x+	Read/write. Creates a new file. Returns FALSE and an error if the file already exists.

File System Functions:

In addition to the file handling functions we have discussed, PHP also provides a number of file system functions that allow you to perform operations on files and directories. Here are some of the most commonly used file system functions:

mkdir(): Creates a new directory.
rmdir(): Deletes an existing directory.
rename(): Renames a file or directory.
file_exists(): Checks if a file exists.
is_file(): Checks if a file is a regular file.
is_dir(): Checks if a file is a directory.
scandir(): Returns an array of files and directories in a directory.

III. Session and Cookies

Introduction

Session and cookies are important concepts in web development, and they are commonly used in PHP to store user-specific data. In this PDF content, we will cover the basics of sessions and cookies in PHP, including how to create, set, and retrieve session and cookie data. We will also discuss some best practices for using sessions and cookies in your PHP applications.

Sessions in PHP

- A session is a way to store information about a user across multiple page requests.
- When a user logs in to your website, for example, you can create a session that stores their user ID, username, or other information.
- This information can then be accessed and used throughout the user's session on your website.

Creating a session in PHP is easy. You simply call the session_start() function at the beginning of your PHP script. This function creates a new session or resumes an existing session, depending on whether a session cookie already exists. Here is an example:

```
<?php
session_start();
// Set session variables
$_SESSION["user_id"] = 12345;
$_SESSION["username"] = "john_doe";
?>
```

In this example, we are calling the session_start() function at the beginning of our PHP script, and then setting two session variables using the $_SESSION superglobal array.

To retrieve the session data in another PHP script, you simply call the session_start() function again, and then access the session variables using the $_SESSION superglobal array. Here is an example:

```
<?php
session_start();

// Retrieve session variables
$user_id = $_SESSION["user_id"];
$username = $_SESSION["username"];
?>
```

In this example, we are calling the session_start() function again, and then retrieving the

session variables we set in the previous script.

Cookies in PHP

A cookie is a way to store information on a user's computer or device. Cookies are often used to store user preferences, such as language or layout settings. Cookies can also be used to store user-specific data, similar to sessions.

Creating a cookie in PHP is done using the setcookie() function. This function takes three parameters: the name of the cookie, the value of the cookie, and the expiration time of the cookie. Here is an example:

```
<?php
// Set a cookie
setcookie("username", "john_doe", time() + 3600);
?>
```

In this example, we are setting a cookie with the name "username" and the value "john_doe". The expiration time of the cookie is set to one hour from the current time, using the time() function.

To retrieve the cookie data in another PHP script, you simply access the cookie using the **$_COOKIE** superglobal array. Here is an example:

```
<?php
// Retrieve a cookie
$username = $_COOKIE["username"];
?>
```

In this example, we are retrieving the value of the "username" cookie we set in the previous script.

Best Practices for Using Sessions and Cookies

- When using sessions and cookies in your PHP applications, it is important to follow some best practices to ensure the security and reliability of your application. Here are some best practices to consider:
- Only store necessary data: Avoid storing sensitive data such as passwords or credit card numbers in sessions or cookies.
- Use secure cookies: When creating cookies, set the secure and httponly flags to ensure that the cookie is only transmitted over a secure connection and cannot be accessed by JavaScript.
- Use session_regenerate_id() function: Call the session_regenerate_id() function periodically to prevent session fixation attacks.
- Set session and cookie timeouts: Set a reasonable timeout period for your sessions

and cookies to ensure that they expire after a certain amount of time.

Error Handling: General error types and debugging

Types of errors in PHP

An error is a mistake in a program that may be caused by writing incorrect syntax or incorrect code. An error message is displayed on your browser containing the filename along with location, a message describing the error, and the line number in which the error has occurred.

There are usually different types of errors. In PHP, mainly four types of errors are considered:

1. Syntax Error or Parse Error
2. Fatal Error
3. Warning Error
4. Notice Error

We will discuss all these errors in detail with examples:

Syntax Error or Parse Error

A syntax error is a mistake in the syntax of source code, which can be made by programmers due to their lack of concern or knowledge. It is also known as **Parse error**. The compiler is used to catch the syntax error at compile time.

Example 1: Missing semicolon

```
<?php
echo "Bob: I'm Bob. How are you?"
?>
```

Output

Parse error: syntax error, unexpected 'echo' (T_ECHO), expecting ',' or ';' in C:\xampp\htdocs\program\fatalerror.php on line 2

Explanation: In this above example, a semicolon (;) was missing in **line 5**. So, it generated a parse error and displayed an error message on the browser as given in the output.

Example 2: Missing dollar symbol

```
<?php
/*------------------syntax error-------------------*/
car = "Tesla";
```

```
    echo car;
```

Output

Parse error: syntax error, unexpected '=' in C:\xampp\htdocs\program\fatalerror.php on line 5

Explanation: In this above example, the dollar ($) symbol was missing in **line 5**. So, it generated a parse error and displayed an error message on the browser as given in the output.

Fatal Error

- A fatal error is another type of error, which is caused by the use of an undefined function.
- The PHP compiler understands the PHP code but also recognizes the undefined function.
- This means that when a function is called without providing its definition, the PHP compiler generates a fatal error.

A fatal error is generated when a function is called without its definition. See the below example containing the fatal error -

Example: Calling undefined function

```
<?php
/*-----------------fatal error-------------------*/
function add($f1, $f2) {
   $sum = $f1 + $f2;
   echo "Addition:" . $sum;
}

$f1 = 23;
$f2 = 56;

//call the function that is not defined
//generate fatal error
catch_fatal_error();
//echo "Fatal Error";
?>
```

In the above code, we have defined the add() function but called another function, which is **catch_fatal_error()**. Therefore, it generates a fatal error and prints an error message on the browser as given below:

Output

Fatal error: Uncaught Error: Call to undefined function catch_fatal_error() in C\xampp\htdocs\program\fatalerror.php:15 Stack trace: #0 {main} thrown in C:\xampp\htdocs\program\fatalerror.php on line 13

Warning Error

A warning is generated when the programmer tries to include a missing file. The PHP function calls that missing file, which does not exist. The warning error does not stop/prevent the execution of the program.

The main reason behind generating a warning error is to pass an incorrect number of parameters to a function or to include a missing file.

Example: Include missing file

```
<?php
$name = 'Naruto';
echo "Warning Error: ";

//include a file in the code
include('ninja.php');
?>
```

Output

Warning Error: Warning: include(ninja.php): failed to open stream: No such file or directory in C:\xampp\htdocs\program\fatalerror.php on line 7

Warning: include(): Failed opening 'jtp.php' for inclusion (include_path='C:\xampp\php\PEAR') in C:\xampp\htdocs\program\fatalerror.php on line 7

Explanation: In this example, we tried to include a file in our program, which does not exist. So, it generated a warning and displayed an error message.

Notice Error

A notice error is the same as a warning error. When the program contains something wrong, the notice error occurs. But it allows/continues the execution of the program with a notice error. Notice error does not prevent the execution of the code. **For example** - access to an undefined variable.

Generally, a notice error occurs when we try to use or access a variable that is undefined. See the below example to understand it-

Example 2: Access undefined variable

```
<?php
/*-----------------syntax error-----------------*/
car = "Tesla";
echo car;
```

Output

PHP Parse error: syntax error, unexpected token "=" in C:\xampp\htdocs\sem4\unit4\err.php on line 3
Parse error: syntax error, unexpected token "=" in C:\xampp\htdocs\sem4\unit4\err.php on line 3

Explanation: In this above example, we were trying to use a variable **$automobile**, which was not defined. Therefore, it generated a notice "**Undefined variable**" and continued the execution of the program.

Displaying PHP errors

To display PHP errors on a web page, you can use the `error_reporting()` and `ini_set()` functions. Here's an example:

```
error_reporting(E_ALL);
ini_set('display_errors', 1);
```

This code sets the error reporting level to show all types of errors, and enables the display of errors on the web page. This is useful for debugging your code during development, but should be disabled in production environments to prevent sensitive information from being exposed to users.

Example :

Another option is to log errors to a file instead of displaying them on the web page. This can be done using the `error_log()` function. Here's an example:

```
error_reporting(E_ALL);
ini_set('log_errors', 1);
ini_set('error_log', '/var/log/php_errors.log');
```

This code sets the error reporting level to show all types of errors, and enables logging of errors to the file `/var/log/php_errors.log`. This is a more secure way to handle errors in production environments, as it keeps sensitive information hidden from users but still allows you to debug your code if necessary.

Adjusting Error Reporting.

- To adjust error reporting in PHP, you can use the `error_reporting()` function to set the level of errors that are reported.
- You can also use the `ini_set()` function to adjust the `display_errors` and `log_errors` settings to control whether errors are displayed on the screen or logged to a file.

Creating Custom error handle

What is an Exception?

- An exception is an object that describes an error or unexpected behaviour of a PHP script.
- Exceptions are thrown by many PHP functions and classes.
- User defined functions and classes can also throw exceptions.
- Exceptions are a good way to stop a function when it comes across data that it cannot use.

Throwing an Exception

The `throw` statement allows a user defined function or method to throw an exception. When an exception is thrown, the code following it will not be executed.

If an exception is not caught, a fatal error will occur with an "Uncaught Exception" message.

The try...catch Statement

To avoid the error from the example above, we can use the `try...catch` statement to catch exceptions and continue the process.

Syntax

```
try {
  code that can throw exceptions
} catch(Exception $e) {
  code that runs when an exception is caught
}
```

Example

Show a message when an exception is thrown:

```php
<?php
function divide($dividend, $divisor) {
  if($divisor == 0) {
    throw new Exception("Division by zero");
  }
  return $dividend / $divisor;
}

try {
  echo divide(5, 0);
} catch(Exception $e) {
  echo "Unable to divide.";
}
?>
```

PHP debugging techniques

Here are some common debugging techniques in PHP:

- **Print statements**: You can use `echo` or `var_dump` statements to print the value of a variable or the output of a function. This can help you see what's going on in your code and identify where errors are occurring.
- **Debugging tools:** PHP has a number of debugging tools that you can use to step through your code, set breakpoints, and inspect variables at runtime. Some popular PHP debugging tools include Xdebug, Zend Debugger, and PHP Debug Bar.
- **Error reporting**: As mentioned earlier, you can adjust the error reporting level in PHP to show different types of errors. This can help you identify errors that may be occurring in your code.
- **Logging:** You can use logging to record information about your code as it runs. This can be useful for debugging and troubleshooting issues that may be occurring in your code.
- **Unit testing**: Unit testing involves creating automated tests that verify the behavior of individual components of your code. This can help you catch errors early and ensure that your code is working as expected.

By using these techniques, you can identify and fix errors in your PHP code and create more reliable and robust applications.