B.C.A. (Sem – VI)

B.C.A. - 601

Building Application Using PHP

Purushottam Singh

Unit:- 2

Unit – 2

Object Oriented Programming With PHP And Error Handling:-

Basic PHP Construction for OOP:-

- Before we go in detail, lets define important terms related to Object Oriented Programming.
- **Class** - This is a programmer-defined data type, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.
- **Object**- An individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.
- **Member Variable** – These are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions. These variables are called attribute of the object once an object is created.
- **Member function** – These are the function defined inside a class and are used to access object data.
- **Inheritance** – When a class is defined by inheriting existing function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.
- **Parent class** – A class that is inherited from by another class. This is also called a base class or super class.
- **Child Class** – A class that inherits from another class. This is also called a subclass or derived class.
- **Polymorphism** – This is an object oriented concept where same function can be used for different purposes. For example function name will remain same but it make take different number of arguments and can do different task.
- **Overloading** – A type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly functions can also be overloaded with different implementation.
- **Data Abstraction** – Any representation of data in which the implementation details are hidden (abstracted).
- **Encapsulation** – Refers to a concept where we encapsulate all the data and member functions together to form an object.
- **Constructor** – Refers to a special type of function which will be called automatically whenever there is an object formation from a class.
- **Destructor** – Refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

Destructor – refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

Defining PHP Classes:-

- The general form for defining a new class in PHP is as follows –

```
<?php
  class phpClass
  {
      var $var1;
      var $var2 = "constant string";

      function myfunc ($arg1, $arg2)
      {
          Functional Statement-1;
          Functional Statement-2;
          Functional Statement-3;
      }
  }
?>
```

- The special form class, followed by the name of the class that you want to define.
- A set of braces enclosing any number of variable declarations and function definitions.
- Variable declarations start with the special form var, which is followed by a conventional $ variable name; they may also have an initial assignment to a constant value.
- Function definitions look much like standalone PHP functions but are local to the class and will be used to set and access object data.
- The variable **$this** is a special variable and it refers to the same object ie. itself.

- Here is an example which defines a class of Books type –

```
<?php
class Books
{
    /* Member variables.*/
    var $price;
    var $title;

    /* Member functions */
    function setPrice($par)
    {
      $this->price = $par;
    }

    function getPrice()
    {
      echo $this->price ."<br/>";
    }

    function setTitle($par)
    {
      $this->title = $par;
    }

    function getTitle()
    {
      echo $this->title ." <br/>";
    }
}
?>
```

Creating Objects in PHP:-

- Once you defined your class, then you can create as many objects as you like of that class type. Following is an example of how to create object using new operator.

Example:-

```
$physics = new Books;
$maths = new Books;
$chemistry = new Books;
```

Where $physics, $maths and $chemistry are objects of class Book.

- Here we have created three objects and these objects are independent of each other and they will have their existence separately. Next we will see how to access member function and process member variables.

Calling Member Functions:-

- After creating your objects, you will be able to call member functions related to that object. One member function will be able to process member variable of related object only.
- Following example shows how to set title and prices for the three books by calling member functions.

```
$physics->setTitle( "Physics for High School" );
$chemistry->setTitle( "Advanced Chemistry" );
$maths->setTitle( "Algebra" );
$physics->setPrice( 10 );
$chemistry->setPrice( 15 );
$maths->setPrice( 7 );
```

Constructor Functions:-

- Constructor Functions are special type of functions which are called automatically whenever an object is created. So we take full advantage of this behaviour, by initializing many things through constructor functions.
- PHP provides a special function called __construct() to define a constructor. You can pass as many as arguments you like into the constructor function.
- Following example will create one constructor for Books class and it will initialize price and title for the book at the time of object creation.

```
function __construct( $par1, $par2 )
{
  $this->title = $par1;
  $this->price = $par2;
}
```

- Now we don't need to call set function separately to set price and title. We can initialize these two member variables at the time of object creation only.

Destructor:-

- Like a constructor function you can define a destructor function using function __destruct(). You can release all the resources with-in a destructor.

Inheritance:-

- PHP class definitions can optionally inherit from a parent class definition by using the extends clause. The syntax is as follows –

```
class Child extends Parent
{
  <definition body>
}
```

- The effect of inheritance is that the child class (or subclass or derived class) has the following characteristics –
  - Automatically has all the member variable declarations of the parent class.
  - Automatically has all the same member functions as the parent, which (by default) will work the same way as those functions do in the parent.
- Following example inherit Books class and adds more functionality based on the requirement.

```
class Novel extends Books
{
  var $publisher;

  function setPublisher($par)
  {
    $this->publisher = $par;
  }

  function getPublisher()
  {
    echo $this->publisher. "<br />";
  }
}
```

- Now apart from inherited functions, class Novel keeps two additional member functions.

Function Overriding:-

- Function definitions in child classes override definitions with the same name in parent classes. In a child class, we can modify the definition of a function inherited from parent class.
- In the following example getPrice and getTitle functions are overridden to return some values.

```
function getPrice() {
  echo $this->price . "<br/>";
  return $this->price;
}
```

```
function getTitle(){
  echo $this->title . "<br/>";
  return $this->title;
}
```

Public Members:-

- Unless you specify otherwise, properties and methods of a class are public. That is to say, they may be accessed in three possible situations -
  - From outside the class in which it is declared
  - From within the class in which it is declared
  - From within another class that implements the class in which it is declared
- Till now we have seen all members as public members. If you wish to limit the accessibility of the members of a class then you define class members as private or protected.

Private members:-

- By designating a member private, you limit its accessibility to the class in which it is declared. The private member cannot be referred to from classes that inherit the class in which it is declared and cannot be accessed from outside the class.
- A class member can be made private by using private keyword infront of the member.

```
class MyClass
{
  private $car = "skoda";
  $driver = "SRK";

  function __construct($par)
  {
    // Statements here run every time
    // an instance of the class
    // is created.
  }

  function myPublicFunction()
  {
    return("I'm visible!");
  }

  private function myPrivateFunction()
  {
    return("I'm  not visible outside!");
  }
}
```

- When MyClass class is inherited by another class using extends, myPublicFunction() will be visible, as will $driver. The extending class will not have any awareness of or access to myPrivateFunction and $car, because they are declared private.

Protected members:-

- A protected property or method is accessible in the class in which it is declared, as well as in classes that extend that class. Protected members are not available outside of those two kinds of classes. A class member can be made protected by using protected keyword in front of the member.

```
class MyClass
{
  protected $car = "skoda";
  $driver = "SRK";

  function __construct($par)
  {
    // Statements here run every time
    // an instance of the class
    // is created.
  }
```

```
    function myPublicFunction()
    {
      return("I'm visible!");
    }

    protected function myPrivateFunction()
    {
      return("I'm  visible in child class!");
    }
  }
```

Interfaces:-

- Interfaces are defined to provide a common function names to the implementers. Different implementors can implement those interfaces according to their requirements. You can say, interfaces are skeletons which are implemented by developers.

```
interface Mail
{
  public function sendMail();
}
```

Then, if another class implemented that interface, like this –

```
class Report implements Mail
{
  // sendMail() Definition goes here
}
```

Abstract Classes:-

- An abstract class is one tnat cannot be instantiated, only inherited. You declare an abstract class with the keyword abstract, like this -
- When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child; additionally, these methods must be defined with the same visibility.

```
abstract class MyAbstractClass
{
  abstract function myAbstractFunction()
  {
      // Functional Statements;
  }
}
```

- Note that function definitions inside an abstract class must also be preceded by the keyword abstract. It is not legal to have abstract function definitions inside a non-abstract class.

Static Keyword:-

- Declaring class members or methods as static makes them accessible without needing an instantiation of the class. A member declared as static can not be accessed with an instantiated class object (though a static method can).

```
<?php
  class Foo {
    public static $my_static = 'foo';

    public function staticValue() {
      return self::$my_static;
    }
  }
  print Foo::$my_static . "\n";
  $foo = new Foo();

  print $foo->staticValue() . "\n";
?>
```

Final Keyword:-

- PHP 5 introduces the final keyword, which prevents child classes from overriding a method by prefixing the definition with final. If the class itself is being defined final then it cannot be extended.
- Following example results in Fatal error: Cannot override final method BaseClass::moreTesting()

```
<?php

  class BaseClass
{
    public function test()
    {
      echo "BaseClass::test() called<br>";
    }

    final public function moreTesting()
    {
      echo "BaseClass::moreTesting() called<br>";
    }
  }

  class ChildClass extends BaseClass
  {
    public function moreTesting() {
    echo "ChildClass::moreTesting() called<br>";
    }
  }
?>
```

Calling parent constructors (Constructor in derived class):-

- Instead of writing an entirely new constructor for the subclass, let's write it by calling the parent's constructor explicitly and then doing whatever is necessary in addition for instantiation of the subclass. Here's a simple example –

```
class Name {
  var $_firstName;
  var $_lastName;

  function Name($first_name, $last_name) {
    $this->_firstName = $first_name;
    $this->_lastName = $last_name;
  }

  function toString() {
    return($this->_lastName .", " .$this->_firstName);
  }
}

class NameSub1 extends Name {
  var $_middleInitial;

  function NameSub1($first_name, $middle_initial, $last_name) {
    Name::Name($first_name, $last_name);
    $this->_middleInitial = $middle_initial;
  }

  function toString() {
    return(Name::toString() . " " . $this->_middleInitial);
  }
}
```

- In this example, we have a parent class (Name), which has a two-argument constructor, and a subclass (NameSub1), which has a three-argument constructor.

- The constructor of NameSub1 functions by calling its parent constructor explicitly using the :: syntax (passing two of its arguments along) and then setting an additional field. Similarly, NameSub1 defines its non constructor toString() function in terms of the parent function that it overrides.
- NOTE – A constructor can be defined with the same name as the name of a class. It is defined in above example.

File and File System Functions:-

- The filesystem functions allow you to access and manipulate the filesystem.
- The filesystem functions are part of the PHP core. There is no installation needed to use these functions.
- When specifying a path on Unix platforms, a forward slash (/) is used as directory separator.
- On Windows platforms, both forward slash (/) and backslash (\) can be used.
- The behavior of the filesystem functions is affected by settings in php.ini.

Function	Description
basename()	Returns the filename component of a path
chgrp()	Changes the file group
chmod()	Changes the file mode
chown()	Changes the file owner
clearstatcache()	Clears the file status cache
copy()	Copies a file
delete()	See unlink() or unset()
dirname()	Returns the directory name component of a path
disk_free_space()	Returns the free space of a directory
disk_total_space()	Returns the total size of a directory
diskfreespace()	Alias of disk_free_space()
fclose()	Closes an open file
feof()	Tests for end-of-file on an open file
fflush()	Flushes buffered output to an open file
fgetc()	Returns a character from an open file
fgetcsv()	Parses a line from an open file, checking for CSV fields
fgets()	Returns a line from an open file
fgetss()	Returns a line, with HTML and PHP tags removed, from an open file
file()	Reads a file into an array
file_exists()	Checks whether or not a file or directory exists
file_get_contents()	Reads a file into a string
file_put_contents()	Writes a string to a file
fileatime()	Returns the last access time of a file
filectime()	Returns the last change time of a file

filegroup()	Returns the group ID of a file
fileinode()	Returns the inode number of a file
filemtime()	Returns the last modification time of a file
fileowner()	Returns the user ID (owner) of a file
fileperms()	Returns the permissions of a file
filesize()	Returns the file size
filetype()	Returns the file type
flock()	Locks or releases a file
fnmatch()	Matches a filename or string against a specified pattern
fopen()	Opens a file or URL
fpassthru()	Reads from an open file, until EOF, and writes the result to the output buffer
fputcsv()	Formats a line as CSV and writes it to an open file
fputs()	Alias of fwrite()
fread()	Reads from an open file
fscanf()	Parses input from an open file according to a specified format
fseek()	Seeks in an open file
fstat()	Returns information about an open file
ftell()	Returns the current position in an open file
ftruncate()	Truncates an open file to a specified length
fwrite()	Writes to an open file
glob()	Returns an array of filenames / directories matching a specified pattern
is_dir()	Checks whether a file is a directory
is_executable()	Checks whether a file is executable
is_file()	Checks whether a file is a regular file
is_link()	Checks whether a file is a link
is_readable()	Checks whether a file is readable
is_uploaded_file()	Checks whether a file was uploaded via HTTP POST
is_writable()	Checks whether a file is writeable
is_writeable()	Alias of is_writable()
lchgrp()	Changes group ownership of symlink
lchown()	Changes user ownership of symlink
link()	Creates a hard link
linkinfo()	Returns information about a hard link

lstat()	Returns information about a file or symbolic link
mkdir()	Creates a directory
move_uploaded_file()	Moves an uploaded file to a new location
parse_ini_file()	Parses a configuration file
parse_ini_string()	Parses a configuration string
pathinfo()	Returns information about a file path
pclose()	Closes a pipe opened by popen()
popen()	Opens a pipe
readfile()	Reads a file and writes it to the output buffer
readlink()	Returns the target of a symbolic link
realpath()	Returns the absolute pathname
realpath_cache_get()	Returns realpath cache entries
realpath_cache_size()	Returns realpath cache size
rename()	Renames a file or directory
rewind()	Rewinds a file pointer
rmdir()	Removes an empty directory
set_file_buffer()	Sets the buffer size of an open file
stat()	Returns information about a file
symlink()	Creates a symbolic link
tempnam()	Creates a unique temporary file
tmpfile()	Creates a unique temporary file
touch()	Sets access and modification time of a file
umask()	Changes file permissions for files
unlink()	Deletes a file

PHP Session:-

- A session is a way to store information (in variables) to be used across multiple pages.
- Unlike a cookie, the information is not stored on the users computer.
- When you work with an application, you open it, do some changes, and then you close it.
- This is much like a Session. The computer knows who you are.
- It knows when you start the application and when you end.
- But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.
- Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.
- So; Session variables hold information about one single user, and are available to all pages in one application.
- Tip: If you need a permanent storage, you may want to store the data in a database.
- A session is started with the session_start() function.
- Session variables are set with the PHP global variable: $_SESSION.

- Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

- Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").
- Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).
- Also notice that all session variable values are stored in the global $_SESSION variable:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>
```

IMP

Cookies:-

- A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer.
- Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.
- A cookie is created with the setcookie() function.

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

- Only the name parameter is required. All other parameters are optional.
- The following example creates a cookie named "user" with the value "Shrikant". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).
- We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

```
<?php
$cookie_name = "user";
$cookie_value = "Shrikant";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
```

```
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
   echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
   echo "Cookie '" . $cookie_name . "' is set!<br>";
   echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
</body>
</html>
```

- To modify a cookie, just set (again) the cookie using the setcookie() function.
- To delete a cookie, use the setcookie() function with an expiration date in the past.

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
```

IMP

Error Handling and Debugging:-

General Error Types and Debugging:-

- When developing Web applications with PHP and MySQL, you end up with potential bugs in one of four or more technologies.
- You could have HTML issues, PHP problems, SQL errors, or MySQL mistakes. To be able to fix the bug, you must first determine in what realm the bug resides.
- PHP errors are the ones you'll see most often, as this language will be at the heart of your applications. PHP errors fall into three general areas:
  1. Syntactical
  2. Run Time
  3. Logical

Displaying PHP Errors:-

- Error handling is the process of catching errors raised by your program and then taking appropriate action. If you would handle errors properly then it may lead to many unforeseen consequences.
- It is very simple in PHP to handle an error.
- All the errors and their respective levels can be set using following PHP built-in library function where level can be any of the value defined for respective error.

```
int error_reporting ( [int $level] )
```

- Following is the way you can create one error handling function –

```
<?php
  function handleError($errno, $errstr,$error_file,$error_line) {
    echo "<b>Error:</b> [$errno] $errstr - $error_file:$error_line";
    echo "<br />";
    echo "Terminating PHP Script";

    die();
  }
?>
```

- Once you define your custom error handler you need to set it using PHP built-in library set_error_handler function.
- Now let examine our example by calling a function which does not exist.

```
<?php
  error_reporting( E_ERROR );

  function handleError($errno, $errstr,$error_file,$error_line) {
    echo "<b>Error:</b> [$errno] $errstr - $error_file:$error_line";
```

```
        echo "<br />";
        echo "Terminating PHP Script";

        die();
      }

      //set error handler
      set_error_handler("handleError");

      //trigger error
      myFunction();
    ?>
```

Adjusting Error Reporting:-

- You can write your own function to handling any error. PHP provides you a framework to define error handling function.
- This function must be able to handle a minimum of two parameters (error level and error message) but can accept up to five parameters (optionally: file, line-number, and the error context) –

```
error_function(error_level,error_message, error_file,error_line,error_context);
```

- **error_level** - Required - Specifies the error report level for the user-defined error. Must be a value number.
- **error_message** - Required - Specifies the error message for the user-defined error.
- **error_file** - Optional - Specifies the file name in which the error occurred.
- **error_line** - Optional - Specifies the line number in which the error occurred.
- **error_context** - Optional - Specifies an array containing every variable and their values in use when the error occurred.

Exception Handling:-

- PHP 5 has an exception model similar to that of other programming languages.
- Exceptions are important and provide a better control over error handling.
- **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".
- **Catch** – A "catch" block retrieves an exception and creates an object containing the exception information.
- Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php
  try {
    $error = 'Always throw this error';
    throw new Exception($error);

    // Code following an exception is not executed.
    echo 'Never executed';
  }catch (Exception $e) {
    echo 'Caught exception: ',  $e->getMessage(), "\n";
  }

  // Continue execution
  echo 'Hello World';
?>
```

- In the above example $e->getMessage function is used to get error message.

Creating Custom Error Handler:-

- You can define your own custom exception handler. Use following function to set a user-defined exception handler function.

```
string set_exception_handler ( callback $exception_handler )
```

- Here exception_handler is the name of the function to be called when an uncaught exception occurs. This function must be defined before calling set_exception_handler().

```
<?php
  function exception_handler($exception)
{
  echo "Uncaught exception: " , $exception->getMessage(), "\n";
}

  set_exception_handler('exception_handler');
  throw new Exception('Uncaught Exception');

  echo "Not Executed\n";
?>
```

PHP Debugging Techniques:-

- Programs rarely work correctly the first time.
- Many things can go wrong in your program that causes the PHP interpreter to generate an error message.
- To make error messages display in the browser, set the display_errors configuration directive to On.
- PHP defines some constants you can use to set the value of error_reporting such that only errors of certain types get reported: E_ALL (for all errors except strict notices), E_PARSE (parse errors), E_ERROR (fatal errors), E_WARNING (warnings), E_NOTICE (notices), and E_STRICT (strict notices).
- There are following points which need to be verified while debugging your program.
  1. **Missing Semicolons** – Every PHP statement ends with a semicolon (;).
  2. **Not Enough Equal Signs** – When you ask whether two values are equal in a comparison statement, you need two equal signs (==).
  3. **Misspelled Variable Names** – If you misspelled a variable then PHP understands it as a new variable. Remember: To PHP, $test is not the same variable as $Test.
  4. **Missing Dollar Signs** – A missing dollar sign in a variable name is really hard to see, but at least it usually results in an error message so that you know where to look for the problem.
  5. **Troubling Quotes** – You can have too many, too few, or the wrong kind of quotes. So check for a balanced number of quotes.
  6. **Missing Parentheses and curly brackets** – They should always be in pairs.
  7. **Array Index** – All the arrays should start from zero instead of 1.