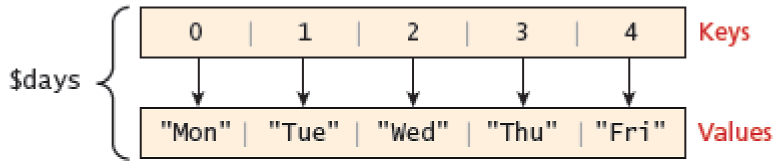
Module IV
Arrays, Classes & Objects, Exception Handling in PHP

4.1 Arrays

PHP supports arrays, like most other programming languages. An array is a data structure that allows the programmer to collect a number of related elements together in a single variable. In PHP an array is an **ordered map**, which associates each value in the array with a key.

An array of key-value pair, containing the days of the week can be visualized as follows –



Array keys in most programming languages are limited to integers, start at 0, and go up by 1. In PHP, keys can be either integers or strings.

Array values are not restricted to integers and strings. They can be any object, type, or primitive supported in PHP.

Defining and Accessing an Array

Declaration of an empty array named days:
$days = array()

Two different ways of defining an array –
Arrayvariable = array(elements separated by comma);
Arrayvariable = [elements separated by comma];

Eg -
$days = array(“Mon”,”Tue”,”Wed”,”Thu”,”Fri”);
$days = [“Mon”,”Tue”,”Wed”,”Thu”,”Fri”]; *// alternate syntax*

$count=array(“one”,”two”,3,4);
$count = [“one”,”two”,3,4];

In these examples, because no keys are explicitly defined for the array, the default key values are 0, 1, 2, . . . , n.

The array elements can also be defined individually using the square bracket notation:
$days = array();
$days[0] = "Mon";

$days[1] = "Tue";
$days[2] = "Wed";
// also alternate approach
$daysB = array();
$daysB[] = "Mon";
$daysB[] = "Tue";
$daysB[] = "Wed";

Arrays are dynamically sized - elements are added to or deleted from the array during run-time. Elements within an array are accessed using the familiar square bracket notation.
Arrayvariable[index];

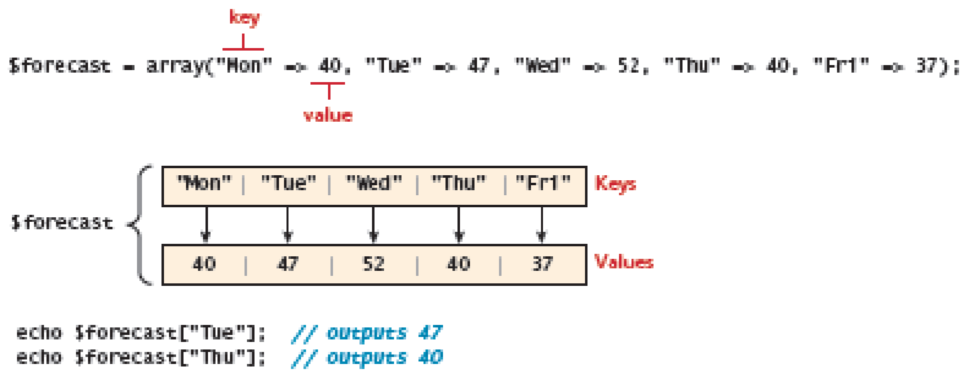
The code below echoes the value of our $days array for the key=1, which results in output of Tue.
echo "Value at index 1 is ". $days[1]; *// index starts at zero*

In PHP, keys can be explicitly defined along with the values. This allows us to use keys other than the classic 0, 1, 2, . . . , n to define the indexes of an array. The array
$days = array(“Mon”,”Tue”,”Wed”,”Thu”,”Fri”); can be defined more explicitly by specifying the keys and values as

$days = array(0 => "Mon", 1 => "Tue", 2 => "Wed", 3 => "Thu", 4=> "Fri");

Where 0,1,2,… are the keys and “Mon”,”Tue”,… are the values.

These types of arrays in PHP are generally referred to as **associative arrays**. Keys must be either integer or string values, but the values can be any type of PHP data type, including other arrays.



```
$forecast = array("Mon" => 40, "Tue" => 47, "Wed" => 52, "Thu" => 40, "Fri" => 37);
```

```
echo $forecast["Tue"];   // outputs 47
echo $forecast["Thu"];   // outputs 40
```

In the above example, the keys are strings (for the weekdays) and the values are weather forecasts for the specified day in integer degrees. To access an element in an associative array, simply use the key value rather than an index:
echo $forecast["Wed"]; *// this will output 52*

Multidimensional Arrays
PHP also supports multidimensional arrays, an array within another array. It can be defined as -

$month = array
(

```
array("Mon","Tue","Wed","Thu","Fri"),
array("Mon","Tue","Wed","Thu","Fri"),
array("Mon","Tue","Wed","Thu","Fri"),
array("Mon","Tue","Wed","Thu","Fri")
);
echo $month[0][3]; // outputs Thu
```

OR

```
$cart = array();
$cart[] = array("id" => 37, "title" => "Burial at Ornans", "quantity" => 1);
$cart[] = array("id" => 345, "title" => "The Death of Marat", "quantity" => 1);
$cart[] = array("id" => 63, "title" => "Starry Night", "quantity" => 1);

echo $cart[2]["title"]; // outputs Starry Night
```

Iterating through an Array

Iteration through the array contents is performed using the loops. Count() function is used to check the size of the array.

When there is nonsequential integer keys or strings as keys (i.e., an associative array), the accessing of elements can't be done using a $i, in such cases foreach loop is used. Foreach loop iterates till the end of array and stops. Each value can be accessed during the iterations, or both the key and value can be accessed, as shown below.

```
// while loop
$i=0;
while ($i < count($days)) {
        echo $days[$i] . "<br>";
        $i++;
}


// do while loop
$i=0;
do {
        echo $days[$i] . "<br>";
        $i++;
} while ($i < count($days));


// for loop
for ($i=0; $i<count($days); $i++) {
        echo $days[$i] . "<br>";
}
```

```
// foreach: iterating through the values
foreach ($forecast as $value) {
        echo $value . "<br>";
}

// foreach: iterating through both the values and the keys
foreach ($forecast as $key => $value) {
        echo "day" . $key . "=" . $value;
}
```

Adding and Deleting Elements

In PHP, arrays are dynamic. The arrays can grow or shrink in size. An element can be added to an array simply by using a key/index that hasn't been used.

```
$days[5] = "Sat";
```

Since there is no current value for key 5, the array grows by one, with the new key/value pair added to the end of our array. If the key had a value already, the new value replaces the value at that key.

As an alternative to specifying the index, a new element can be added to the end of any existing array by -

```
$days[ ] = "Sun";
```

The advantage to this approach is that you don't have to remember the last index key used.

```
$days = array("Mon","Tue","Wed","Thu","Fri");
$days[7] = "Sat";
print_r($days);
```

The print_r() - will display the non-NULL elements of the array, as shown below -

```
Array ([0] => Mon  [1] => Tue  [2] => Wed  [3] => Thu  [4] => Fri  [7] => Sat)
```

The **NULL** value are not displayed.

Elements can be deleted from the array explicitly by using the **unset**() function –

```
$days = array("Mon","Tue","Wed","Thu","Fri");
unset($days[2]);
unset($days[3]);
print_r($days); // outputs: Array ( [0] => Mon [1] => Tue [4] => Fri )
```

The Null values in arrays can be removed and array is **reindexed by using the array_values**() function.

```
$b = array_values($days);
print_r($b); // outputs: Array ( [0] => Mon [1] => Tue [2] => Fri )
// Array is reindexed.
print_r($days) // outputs: Array ( [0] => Mon [1] => Tue [4] => Fri )
```

Checking If a Value Exists

It is possible to check if an array index contains a value or NULL by using the **isset()** function. The returns true if a value has been set, and false otherwise.

```
$days = array (1 => "Tue", 3 => "Wed", 5 => "Thurs");
if (isset($days[0]))
{
        // The code below will never be reached since $days[0] is not set!
        echo "Monday";
}
if (isset($days[1]))
{
        echo "Tuesday";
}
```

Array Sorting – Array Operations

There are many built-in sort functions, which sort by key or by value in PHP.

sort(arrayname); - sort the array in ascending order **by its values**. The array itself is sorted.

```
$days = array("Mon","Tue","Wed","Thu","Fri");
sort($days);
print_r($days); // outputs: Array ([0] => Fri [1] => Mon [2] => Thu [3] => Tue [4] => Wed)
```

sort() function loses the association between the values and the keys.

asort(arrayname) – sort the array in ascending order **by its values**, association is maintained.
Eg –

```
asort($days);
print_r($days); // outputs: Array ([4] => Fri [0] => Mon [3] => Thu [1] => Tue [2] => Wed)
```

More Array Operations

some key array functions are –

```
array_keys($Arrayname)
array_values($Arrayname)
array_rand($Arrayname, $num=1)
array_reverse($Arrayname)
array_walk($Arrayname, $callback, $optionalParam)
in_array($value, $Arrayname)
shuffle($Arrayname)
```

array_keys($Arrayname): This method returns an indexed array with the values being the *keys* of $Arrayname.
Eg : print_r(array_keys($days))
outputs - Array ([0] => 0 [1] => 1 [2] => 2 [3] => 3 [4] => 4)

array_values($Arrayname): This method returns an indexed array with the values being the *values* of $ Arrayname.
Eg: print_r(array_values($days))
Outputs - Array ([0] => Mon [1] => Tue [2] => Wed [3] => Thu [4] => Fri)

array_rand($Arrayname, $num=1): This function returns an array of, as many **random keys** as are requested. If you only want one, the key itself is returned;
otherwise, an array of keys is returned.
For example, print_r(array_rand($days,2))
might output: Array (3, 0)

array_reverse($Arrayname): This method returns $Arrayname in reverse order. The passed $Arrayname is not altered, a new array of reversed elements is returned.
For example, print_r(array_reverse($days))
outputs: Array ([0] => Fri [1] => Thu [2] => Wed [3] => Tue [4] => Mon)

array_walk($Arrayname, $callback, $optionalParam): This method allows to call a method ($callback), for each value in $Arrayname. One or more additional parameters can also be sent to the function.

The $callback function typically takes two parameters by default, the value, and the key of the array.
Eg - prints the value of each element in the array is shown below.

```
$someA = array("hello", "world");
array_walk($someA, "doPrint");

function doPrint($value,$key)
{
        echo $key . ": " . $value . “<br />”;
}
```

Output:

```
0 : hello
1 : world
```

Additional parameters can be sent to the function as shown below –

```
$someA = array("hello", "world");
array_walk($someA, "doPrint",”one”,”two”,3);

function doPrint($value,$key,$t1,$t2,$t3)
{
        echo $key . ": " . $value . “<br />”;
        echo “$t1  $t2  $t3 <br />”,
}
```

```
Output:
0 : hello
one two 3
1 : world
one two 3
```

in_array($value, $arrayname): This method search the array $ arrayname for a value ($value). It returns true if value is found, and false otherwise.

```
if(in_array(“Wed”,$days))
    echo “Element found”;
else
    echo “Element not found”;
```

shuffle($arrayname): This method shuffles $arrayname in random order and makes it an indexed array.

```
shuffle($days);
print_r($days);  // Array ( [0] => Thu [1] => Fri [2] => Wed [3] => Mon [4] => Tue)
shuffle($days);
print_r($days);  // Array ( [0] => Tue [1] => Mon [2] => Thu [3] => Wed [4] => Fri )
```

Superglobal Arrays

PHP uses special predefined associative arrays called **superglobal variables.** It allows the programmer to easily access HTTP headers, query string parameters, and other commonly needed information. They are called superglobal because these arrays are always accessible, from a function , class or file, without using the global keyword.

Some of the superglobal variables are –

NAME	DESCRIPTION
$GLOBALS	Array for storing user data that needs superglobal scope
$_COOKIES	Array of cookie data passed to page via HTTP request
$_ENV	Array of server environment data
$_FILES	Array of file items uploaded to the server
$_GET	Array of query string data passed to the server via the URL
$_POST	Array of query string data passed to the server via the HTTP header
$_REQUEST	Array containing the contents of $_GET, $_POST, and $_COOKIES
$_SESSION	Array that contains session data

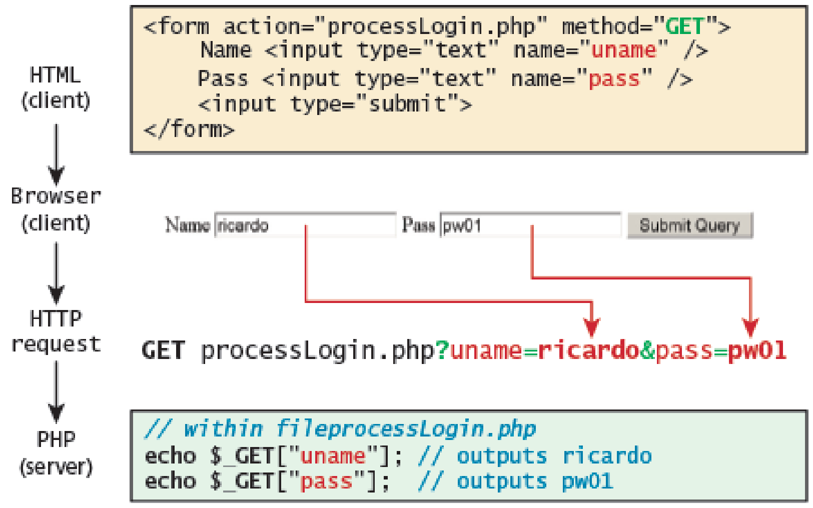
$_SERVER	Array containing information about the request and the server

4.2 $_GET and $_POST Superglobal Arrays

The $_GET and $_POST arrays allows the programmer to access data sent by the client in a query string. They are the most important superglobal variables in PHP.

An HTML form allows a client to send data to the server. That data is formatted such that each value is associated with a name of control defined in the form. If the form was submitted using an HTTP GET request ie. <form method=”get” action=”*server file path*” >, then the resulting URL will contain the data in the query string.
Eg-
https://name-processing.php?firstname=amith&lastname=kumar%20singh
retrieve the values sent through the URL using $_GET[“firstname”] and $_GET[“lastname”]

PHP will populate the superglobal $_GET array using the contents of this query string in the URL.

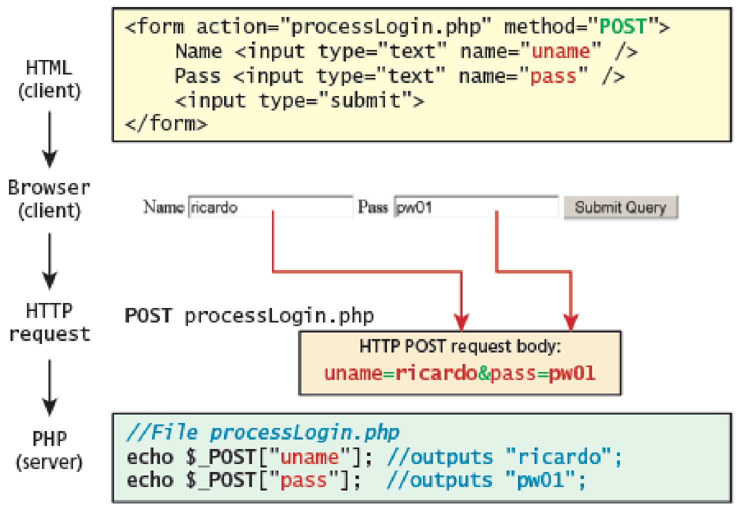


```
<form action="processLogin.php" method="GET">
    Name <input type="text" name="uname" />
    Pass <input type="text" name="pass" />
    <input type="submit">
</form>
```

```
GET processLogin.php?uname=ricardo&pass=pw01
```

```
// within fileprocessLogin.php
echo $_GET["uname"]; // outputs ricardo
echo $_GET["pass"];  // outputs pw01
```

If the form was sent using HTTP POST, then the values will be sent through HTTP POST request body. The values and keys are stored in the $_POST array.



```
<form action="processLogin.php" method="POST">
    Name <input type="text" name="uname" />
    Pass <input type="text" name="pass" />
    <input type="submit">
</form>
```

```
POST processLogin.php
uname=ricardo&pass=pw01
```

```
//File processLogin.php
echo $_POST["uname"]; //outputs "ricardo";
echo $_POST["pass"];  //outputs "pw01";
```

Thus the values passed by the user can easily be accessed at the server side using the global arrays $_GET[] and $_POST[] for ‘get’ and ‘post’ methods respectively.

Determining If Any Data Sent

Forms are used to send the input to the server. At the server side, to check if the user is sending the data by a POST or GET method the global variable $_SERVER['REQUEST_METHOD'] is used.

The variable $_SERVER['REQUEST_METHOD'] contains as a string, indicating the type of HTTP request as GET, POST, etc.

isset() function is used to check if any of the fields are set. It returns true if the value is sent otherwise returns false.

Syntax – isset($POST[fieldname]); or isset($GET[fieldname]);

isset($POST[‘name’]);

Program to display the name and password using php script.

<u>//PHP file named as samplePage.php</u>

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST")
{
        if ( isset($_POST["uname"]) && isset($_POST["pass"]) )
        {
                echo "handling user login now ...";
                $a=$_POST["uname"];
                $b=$_POST["pass"];
```

```
                    echo "username is  $a <br /> password is $b";
          }
}
?>
```

<u>HTML file</u>

```
<!DOCTYPE html>
<html>
<body>
<h1>Some page that has a login form</h1>
<form action="samplePage.php" method="POST">
Name <input type="text" name="uname"/><br/>
Pass <input type="password" name="pass"/><br/>
<input type="submit">
</form>
</body>
</html>
```

Accessing Form Array Data

Sometimes it is required to access multiple values associated with a single name from few controls in form, like checkboxes.

A group of checkboxes will have the same name eg :(name="day"), and multiple values. $_GET['day'] value in the superglobal array *will only contain a single value - the last value from the list.*

To overcome this limitation, change the HTML in the form, ie. change the name attribute for each checkbox from day to day[].

```
<form method="get">
Please select days of the week you are free.<br />
Monday          <input type="checkbox" name="day[ ]" value="Monday" /> <br />
Tuesday         <input type="checkbox" name="day[ ]" value="Tuesday" /> <br />
Wednesday       <input type="checkbox" name="day[ ]" value="Wednesday" /> <br />
Thursday        <input type="checkbox" name="day[ ]" value="Thursday" /> <br />
Friday          <input type="checkbox" name="day[ ]" value="Friday" /> <br />
<input type="submit" value="Submit">
</form>
```

After making this change in the HTML, the corresponding variable $_GET['day'] will now have a value that is of type array. The values are accessed using the foreach loop and count() function returns the length of the array.

```
<?php
echo "You submitted " . count($_GET['day']) . "values";
foreach ($_GET['day'] as $d)
```

```
{
	echo $d . ", ";
}
?>
```

The output is the number of days selected and their values.
Eg

You submitted 3 values Monday , Thursday, Friday

Sanitizing Query Strings
The process of checking user input for incorrect or missing information is referred to as the process of **sanitizing user inputs**.
The user can give input in any way – wrong unexpected input, different data type, empty fields etc. The developer should be able to handle all such issues. He should always develop the web by distrusting user input.

The developer must be able to handle the following cases for *every* query string or form value:

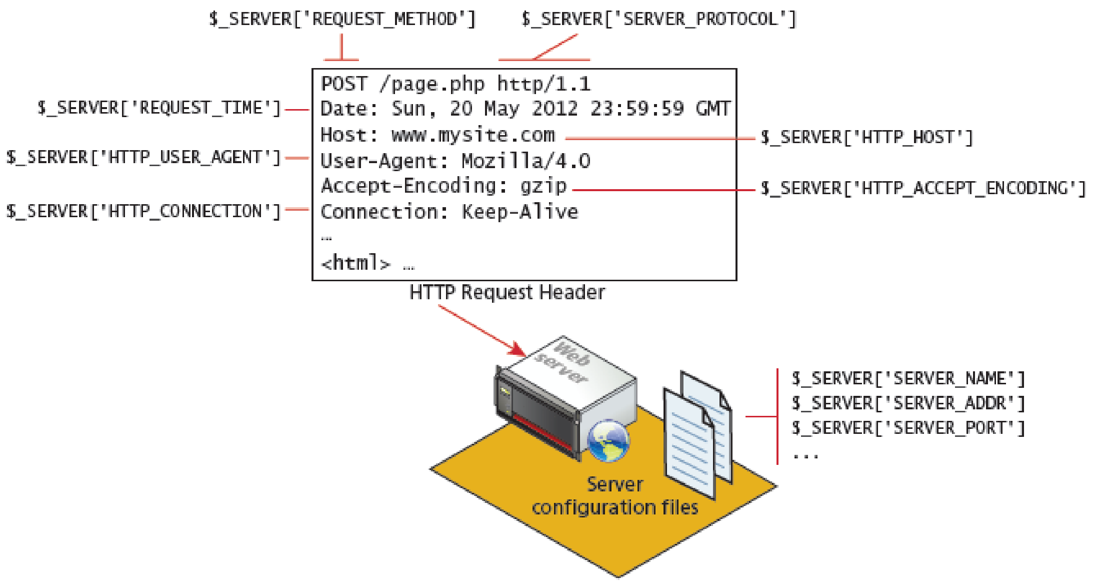
- Query string parameter doesn't exist.
- Query string parameter doesn't contain a value.
- Query string parameter value isn't the correct type.
- Value is required for a database lookup, but provided value doesn't exist in the database table.

4.3 $_SERVER Array

The $_SERVER associative array contains a variety of information. It contains information contained within HTTP request headers sent by the client. It also contains many configuration options for PHP.

Eg. of $_SERVER array and accessing the defined values -

```
echo $_SERVER["SERVER_NAME"] . "<br/>";
echo $_SERVER["SERVER_SOFTWARE"] . "<br/>";
echo $_SERVER["REMOTE_ADDR"] . "<br/>";
```



The $_SERVER array can be classified into keys containing information about the server settings and keys with request header information.

Server Information Keys
In the $_SERVER array,
SERVER_NAME key returns the name of the site that was requested.
SERVER_ADDR key returns the IP of the server.
DOCUMENT_ROOT key returns the file location from which the script is currently running.
SCRIPT_NAME key returns the name of actual script being executed.

Eg: $_SERVER['SERVER_NAME'] ;
$_SERVER['DOCUMENT_ROOT']; etc.

Request Header Information Keys
The web server responds to HTTP requests, and each request contains a request header. The following keys provide programmatic access to the data in the request header-
In the $_SERVER array,
REQUEST_METHOD key returns the request method that was used to access the page: GET, HEAD, POST, PUT.

REMOTE_ADDR key returns the IP address of the requestor.

HTTP_USER_AGENT key returns the operating system and browser used by the client.

HTTP_REFERER key returns the address of the page that referred (linked) us to this page.

Eg: $_SERVER['REQUEST_METHOD'] ;
$_SERVER['REMOTE_ADDR']; etc.

4.4 $_FILES Array

The $_FILES associative array contains information about the file that have been uploaded by the client to the current script. The <input type="file"> element is used to create the user interface for uploading a file from the client to the server. The user interface is only one part of the uploading process. A server script processes the upload file(s) using the $_FILES array.

HTML Requirements for File Uploading-

- The HTML form must use the HTTP POST method, since transmitting a file through the URL is not possible.
- Must add the enctype="multipart/form-data" attribute to the HTML form that is performing the upload so that the HTTP request can submit data in multiple pieces.
- Must include an input tag of type file in the form. This will show up with a browse button beside it so the user can select a file from their computer to be uploaded.

A simple form tag to upload a file to the server -

```
<form enctype='multipart/form-data' method='post'>
<input type='file' name='file1' id='file1' />
<input type='submit' />
</form>
```

Handling the File Upload in PHP

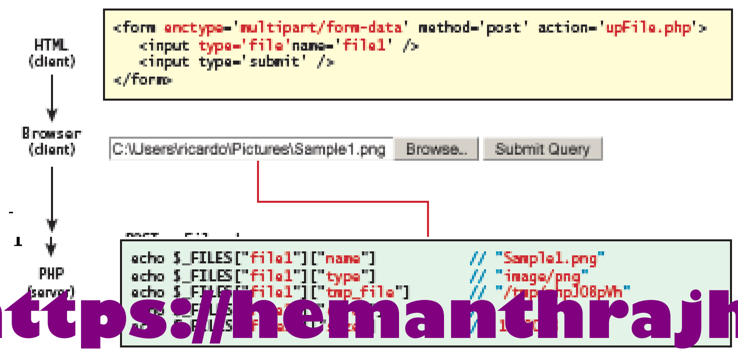
The superglobal $_FILES array is utilized at the server to handle the file uploaded by the client. This array will contain a key=value pair for each file uploaded in the post. The key for each element will be the name attribute from the HTML form, while the value will be an array containing information about the file as well as the file itself. The keys in that array are the name, type, tmp_name, error, and size.

$_FILES[name of file control][other keys]
Different files, uploaded will have a different file control names.

The values for each of the keys are -

- **name** is a string containing the full file name of file, including any file extension.
- **type** defines the MIME type of the file.
- **tmp_name** is the full path to the location on your server where the file is being temporarily stored. The file should be copied to another location if it is required.
- **error** is an integer that encodes many possible errors and is set to UPLOAD_ERR_OK (integer value 0) if the file was uploaded successfully.
- **size** is an integer representing the size in bytes of the uploaded file.

The below figure illustrates the process of uploading a file to the server and how the corresponding upload information is contained in the $_FILES array.



```
<form enctype='multipart/form-data' method='post' action='upFile.php'>
   <input type='file'name='file1' />
   <input type='submit' />
</form>
```

C:\Users\ricardo\Pictures\Sample1.png

```
echo $_FILES["file1"]["name"]          // "Sample1.png"
echo $_FILES["file1"]["type"]          // "image/png"
echo $_FILES["file1"]["tmp_file"]      // "/tmp/phpJ08pVh"
```

Checking for Errors

For every uploaded file, there is an error value associated with it in the $_FILES array. The error values are specified using constant values, which resolve to integers. The value for a successful upload is UPLOAD_ERR_OK. The other errors are –

Error Code	Integer	Meaning
UPLOAD_ERR_OK	0	Upload was successful.
UPLOAD_ERR_INI_SIZE	1	The uploaded file exceeds the `upload_max_filesize` directive in php.ini.
UPLOAD_ERR_FORM_SIZE	2	The uploaded file exceeds the `max_file_size` directive that was specified in the HTML form.
UPLOAD_ERR_PARTIAL	3	The file was only partially uploaded.
UPLOAD_ERR_NO_FILE	4	No file was uploaded. Not always an error, since the user may have simply not chosen a file for this field.
UPLOAD_ERR_NO_TMP_DIR	6	Missing the temporary folder.
UPLOAD_ERR_CANT_WRITE	7	Failed to write to disk.
UPLOAD_ERR_EXTENSION	8	A PHP extension stopped the upload.

File Size Restrictions

Some scripts limit the file size of each upload. There are three main mechanisms for maintaining uploaded file size restrictions: **via HTML in the input form, via JavaScript in the input form, and via PHP coding**.

To restrict the filesize **using HTML form**, add a **hidden** input field before any other input fields in your HTML form with a name of MAX_FILE_SIZE. This technique allows **php.ini** (a file used for configuring PHP settings) maximum file size to be large, while letting files with smaller size. The checking of file size is done at the server side.

```
<form enctype='multipart/form-data' method='post'>
<input type="hidden" name="MAX_FILE_SIZE" value="1000000" />
<input type='file' name='file1' />
<input type='submit' />
</form>
```

The more complete client-side mechanism to prevent a large file from uploading is to prevalidate the form **using JavaScript**.

```
<script>
var file = document.getElementById('file1');
var max_size = document.getElementById("max_file_size").value;
```

```
if (file.files && file.files.length ==1){
        if (file.files[0].size > max_size) {
        alert("The file must be less than " + (max_size/1024) + "KB");
        e.preventDefault();
        }
}
</script>
```

The third mechanism for limiting the uploaded file size is to add a simple check on the server side **using PHP**. This technique checks the file size on the server by simply checking the size field in the $_FILES array.

```
$max_file_size = 10000000;
foreach($_FILES as $fileKey => $fileArray)
{
        if ($fileArray["size"] > $max_file_size) {
                echo "Error: " . $fileKey . " is too big";
        }
        printf("%s is %.2f KB", $fileKey, $fileArray["size"]/1024);
}
```

Limiting the Type of File Upload

To check the type of file, check the file extension and the type of file using type key of $_FILES array.

```
$validExt = array("jpg", "png");
$validMime = array("image/jpeg","image/png");
foreach($_FILES as $fileKey => $fileArray )
{
        $extension = end(explode(".", $fileArray["name"]));
        if (in_array($fileArray["type"],$validMime) && in_array($extension, $validExt))
        {
                echo "all is well. Extension and mime types valid";
        }
        else
        {
                echo $fileKey." Has an invalid mime type or extension";
        }
}
```

Moving the File

PHP function move_uploaded_file() is used to move the file from the temporary file location to the file’s final destination. This function will only work if the source file exists and if the destination location is writable by the web server (Apache). If there is a problem the function will return false.

```
$fileToMove = $_FILES['file1']['tmp_name'];
```

```
$destination = "./upload/" . $_FILES["file1"]["name"];
if (move_uploaded_file($fileToMove,$destination))
{
        echo "The file was uploaded and moved successfully!";
}
else
{
        echo "there was a problem moving the file";
}
```

4.5 Reading/Writing Files

There are two basic techniques for read/writing files in PHP:

- **Stream access** In this technique, just a small portion of the file is read at a time. It is the most memory-efficient approach when reading very large files.
- **All-In-Memory access** In this technique, the entire file is read into memory (i.e., into a PHP variable). While not appropriate for large files, it does make processing of the file extremely easy.

Stream Access

The functions used in this technique are similar to that of C programming.
The function fopen() takes a file location or URL and access mode as parameters. The returned value is a **stream resource (file handle)**.
Some of the common modes are “r” for read, “w” for write, and “c” creates a new file for writing, “rw” for read and write.

The other functions are –

fclose(file handle)- closing the file
fgets(file handle)- To read a single line, returns 0 if no more data
fread(file handle, no.of bytes) - To read an specified amount of data
fwrite(file handle, string)- To write the string to a file
fgetc(file handle) – To read a single character
feof(file handle) –checks for EOF

A program to read from a file and display it -

```
<?php
$f = fopen("sample.txt", "r");
while ($line = fgets($f))
{
        echo $line . "<br>";
}
fclose($f);
?>
```

In-Memory File Access

While the previous approach to reading/writing files requires more care in dealing with the streams, file handles etc. The alternative simpler approach is to read/write the entire contents into the memory with the help of variable.

The alternative functions to process the file are –

Function	Description
file()	Reads the entire file into an array, with each array element corresponding to one line in the file
file_get_contents	Reads the entire file into a string variable
file_put_contents	Writes the contents of a string variable out to a file

The file_get_contents() and file_put_contents() functions allow you to read or write an entire file in one function call. To read an entire file into a variable you can simply use:
$fileAsString = file_get_contents(FILENAME);

To write the contents of a string $writeme to a file, use
file_put_contents(FILENAME, $writeme);

Write a PHP script to retrieve the fields from the text file (EmpID, EmpName, Department). The fields are delimited by comma(,). Display the fields separately.

```
<?php
$filearr = file("Emp.txt");
Foreach($filearr as $empdata)
{
        $EmpFields = explode(',', $empdata);
        $id= $EmpFields[0];
        $name = $EmpFields[1];
        $dept = $EmpFields[2];
        Echo "ID of Employee is $id <br /> Name is $name <br /> Department is $dept<br />";
}
```

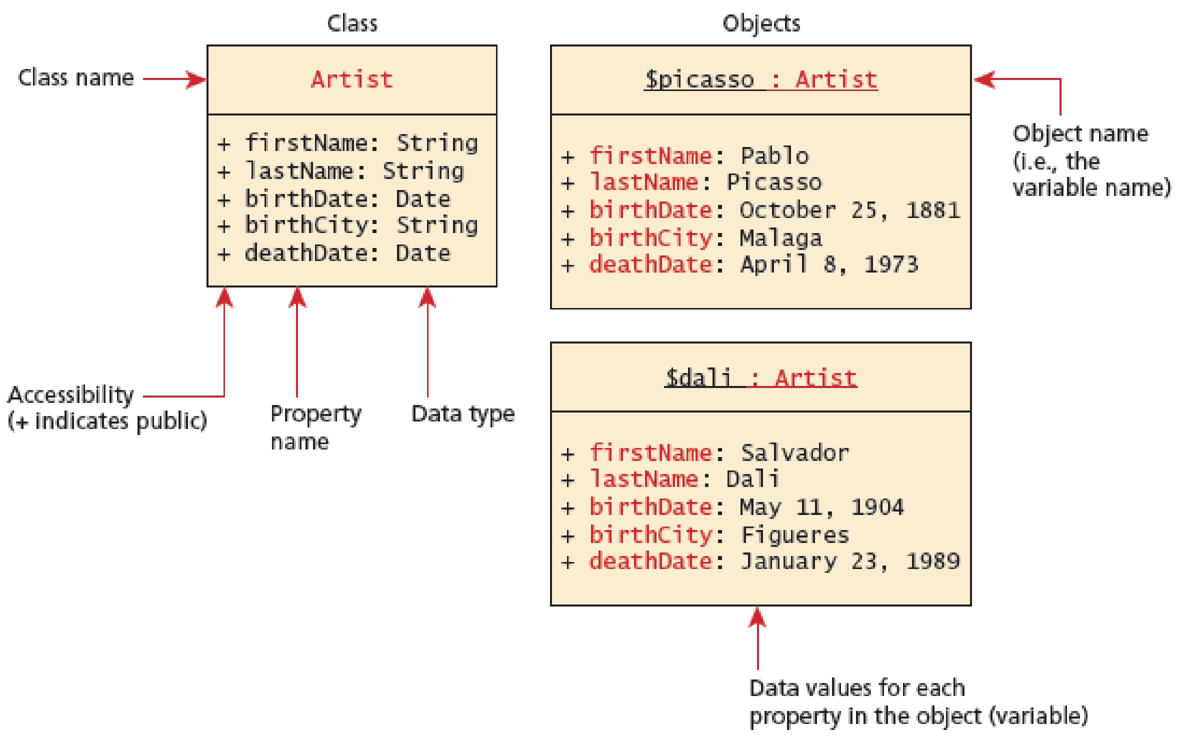
PHP Classes and Objects

4.6 Object-Oriented Overview

PHP is a full-fledged object-oriented language like Java and C++.
Class is a user defined data-type which has data members and member functions.
Each variable created from a class is called an **object** or **instance**, and each object maintains its own set of variables.

The Unified Modeling Language
The standard diagramming notation for object-oriented design is **UML (Unified Modeling Language)**. UML is a set of diagramatic notations for visualizing, specifying, constructing and documenting the components of software system.
It was created by OMG(Object Management Group) in 1997.

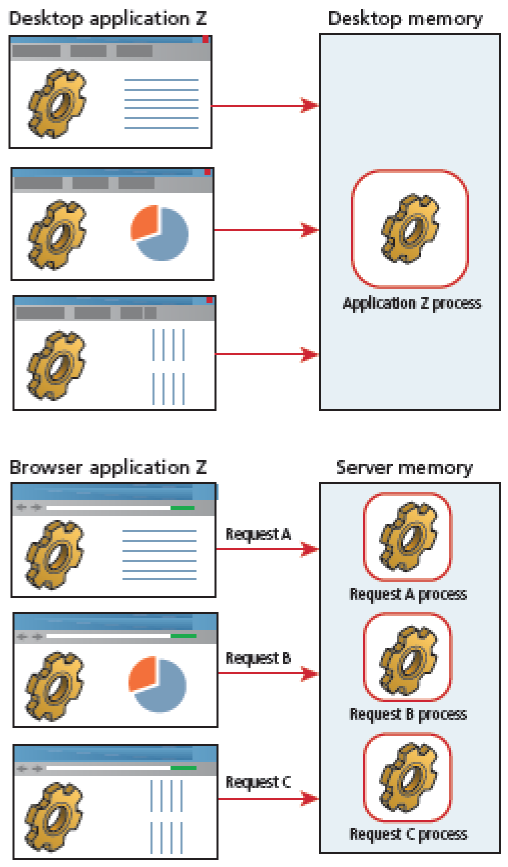
Consider the ‘Artist’ class, with firstname, lastname, birthdate, birthcity and deathdate as members.



Differences between Server and Desktop Objects
One important distinction between web programming and desktop application programming is that the objects created exists only until a web script is terminated. While desktop software can load an object into memory and make use of it for several user interactions. Object must be recreated and loaded into memory for each request in web applications.

In a server, there are thousands of users making requests at once, so objects are destroyed upon responding to each request and also the memory is shared between many simultaneous requests.

It is possible to have objects persist between multiple requests in a server using serialization, which rapidly stores and retrieves object .
The below figure shows the lifetimes of objects in memory between a desktop and a browser application



4.7 Classes and Objects in PHP

Developers can create their own classes in PHP. Classes should be defined in their own files so they can be imported into multiple scripts. A class file is named as **classname.class.php**. Any PHP script can make use of an external class by including the class file in the script.

Defining Classes
The PHP syntax for defining a class uses the class keyword followed by the classname and { } braces. The properties and methods of the class are defined within the braces.
Eg –

```
class Employee{
public $EmpName;
public $EmpID;
public $Dept;
public $Designation;
public $salary;
}
```

Each property in the class is declared using one of the keywords public, protected, or private access specifiers followed by the property or variable name.

Instantiating Objects
Once a class has been defined, any number of instances of that class can be created, using the **new** keyword.

To create two new instances of the Employee class called $emp1 and $emp2, you instantiate two new objects using the new keyword as follows:

```
$emp1 = new Employee();
$emp2 = new Employee ();
```

Properties
The objects can access and modify the properties using an arrow (->).

```
$emp1 = new Employee();
$emp2 = new Employee ();
$emp1->EmpName = "Pablo";
$emp1->EmpID = "M123";
$emp1->Dept = "Testing";
$emp1-> Designation = "Manager";
$emp1->salary = "35K";
```

Constructors
A **constructor** is a member function of a class which initializes objects of a class. Objects can be done by using above method also, but if there are many objects then it is difficult to initialize.

In PHP, constructors are defined as functions with the name __construct() (*two* underscores __ before the word construct.).

```
class Employee{
        public $EmpName;
        public $EmpID;
        public $Dept;
        public $Designation;
        public $salary;
        function __construct($name,$id,$dept,$desig,$sal)
        {
                $this->EmpName = $name;
                $this->EmpID = $id;
                $this->Dept = $dept;
                $this-> Designation = $desig;
                $this->salary = $sal;
        }
}
```

In the constructor each parameter is assigned to an internal class variable using the $this-> syntax. Inside a class $this syntax must be used to reference all properties and methods associated with this particular instance of a class.
The object is created as follows –

```
$emp1 = new Employee("Pablo","M123","Testing","Manager","35K");
$emp2 = new Employee ("Salvador","M124","Testing","Trainee",”10K”);
```

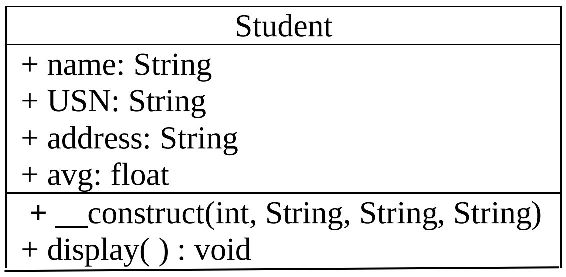
Methods

In object-oriented concept, the operations on properties of objects are performed within the **methods** defined in a class. Methods are like functions, except they are associated with a class. They define the tasks each instance of a class can perform and are useful since they associate behavior with objects.

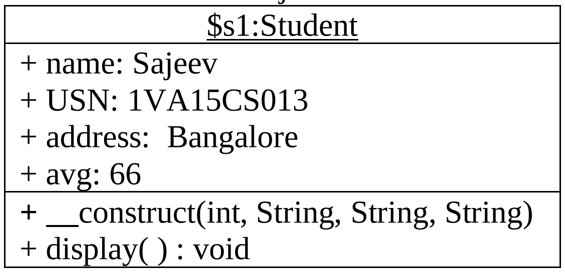
The below code uses the display function to display the details of an object.

```
<?php
Class Student
{
        public $name;
        public $USN;
        public $address;
        public $avg;

        function __construct($n,$usn,$add,$avg)
        {
                $this->name = $n;
                $this->USN= $usn;
                $this->address = $add;
                $this->avg = $avg;
        }
        function display()
        {
            echo “name is $this->name <br /> USN is $this->USN <br />”;
             echo “Address is $this->address <br /> Avg is $this->avg <br />”;
        }
}

$s1 = new Student("Sajeev", "1VA15CS013","Bangalore",66);
$s1->display();
?>
```

Class



Object

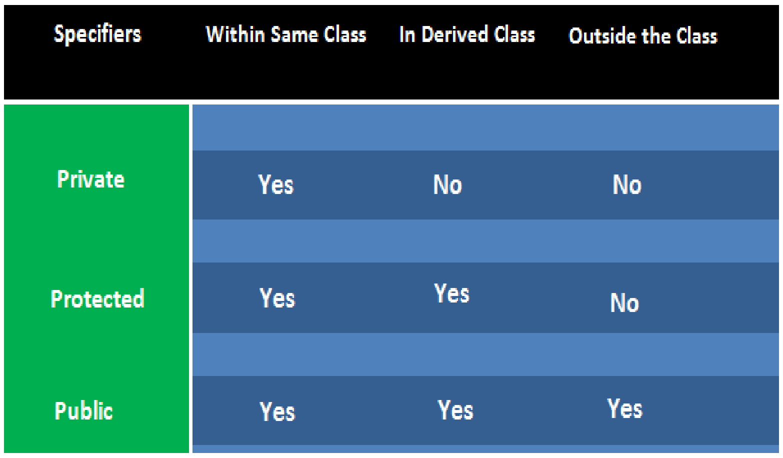


_ _toString( ) method – Any function which returns a string, can be renamed as _ _toString( ). The function is called by just using the object as a variable.
Eg-

```
$s1 = new Student( );
Echo $s1;   //calls the _ _toString( ) function and displays the string returned.
```

Visibility
The **visibility** of a property or method determines the accessibility of a **class member** (i.e., a property or method) and can be set to public, private, or protected.
The public keyword means that the property or method is accessible to any code anywhere that has a reference to the object.
The private keyword sets a method or variable to only be accessible from within the class. This means that we cannot access or modify the property from outside of the class, even if it is referenced with an object.
The protected keyword sets a method or variable to be accessible from within the class and from its derived classes.

In UML, the "+" symbol is used to denote public properties and methods, the "–" symbol for private members, and the "#" symbol for protected members.



Specifiers	Within Same Class	In Derived Class	Outside the Class
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

Static Members
A **static** member is a property or method that all instances of a class share. Unlike an instance property, where each object gets its own value for that property, there is only one value for a class's static property. It is common for all objects.

A variable is declared static by including the static keyword in the declaration:
public static $StudentCount = 0;

A static property is accessed within a class using self::syntax. This is an indication to programmer to understand that the variable is static and is not associated with an instance ($this). This static variable is accessed outside the class, without any instance of an Student object by using the class name, that is, Student::$StudentCount.

Static methods are similar to static properties in that they are globally accessible (if public) and are not associated with particular objects. Static methods cannot access instance members. Static methods are called using the same double colon syntax as static properties.

Static members are used when some data or operations are independent of the instances of the class.

```
<?php
Class Student
{
        public $name;
        public $USN;
        public $address;
        public $avg;
        public static $StudentCount = 0;

        function __construct($n,$usn,$add,$avg)
        {
                $this->name = $n;
                $this->USN= $usn;
                $this->address = $add;
                $this->avg = $avg;
                self::$StudentCount++;
        }
}

$s1 = new Student("Sajeev", "1VA15CS013","Bangalore",66);
Echo Student::$StudentCount;
?>
```

Output:

1

In the UML notation the shared property is underlined to indicate its static nature.

Class

Student
+ StudentCount: int + name: String + USN: String + address: String + avg: float
+ __construct(int, String, String, String)

Object

$s1:Student
+ self::$StudentCount + name: Sajeev + USN: 1VA15CS013 + address: Bangalore + avg: 66
+ __construct(int, String, String, String)

Class Constants

Constant values are stored more efficiently as class constants using ‘const’ keyword. There will be the constant values as long as they are not calculated or updated.

Eg - const EARLIEST_DATE = 'January 1, 1200';
Unlike all other variables, constants don't use the $ symbol when declaring or using them. They can be accessed both inside and outside the class using self::EARLIEST_DATE in the class and classReference::EARLIEST_DATE outside.

4.8 Object-Oriented Design

The object-oriented design of software offers many benefits in terms of modularity, testability, and reusability.

Objects can be reused across the program. The software is easier to maintain, as any changes in the structure need to change only the class. OO concepts enables faster development and easier maintenance of the software.

Data Encapsulation

Object-oriented design enables the possibility of **encapsulation** (hiding), that is, restricting access to an object's internal components (properties and methods). Another way of understanding encapsulation is: it is the hiding of an object's implementation details.

In properly encapsulated class, the properties and methods, are hidden using the private access specifier and these properties and methods are accessed to outside the class using the public methods. Thus we can restrict the usage of required properties and methods.

The hidden properties are accessed and modified using public methods commonly called **getters and setters** (or accessors and mutators). A getter returns a variable's value does not modify the property. It is normally called without parameters, and returns the property from within the class.
Eg -

```
public function getFirstName() {
        return $this->firstName;
}
```

Setter methods modify properties, and allow extra logic to be added to prevent properties from being set to strange values.

Eg –

```
public function setFirstName($name) {
        $this->firstName = $name;
}
```

The below example shows the modified Student class with getters and setters. Some of the properties are private. These properties cannot be accessed or assigned from outside the class, the getters and setters.

```
<?php
Class Student
{
        private $name;
        private $USN;
        public $address;
        public $avg;

function __construct($n,$usn,$add,$avg)
{
        $this->name = $n;
        $this->USN= $usn;
        $this->address = $add;
        $this->avg = $avg;
}

public function getname() { return $this->name; }
public function getUSN() { return $this->USN; }

public function setname($fullname) { $this->name=$fullname; }
public function setUSN($usn) { $this->USN=$usn; }
}

$s1 = new Student("Sajeev", "1VA15CS013","Bangalore",66);
$s1->setname("Adithya");
$s1->avg = 77;              //setting avg directly , where as name is set using the function
$name = $s1->getname();
$usn= $s1->getUSN();
echo "name is $name <br /> USN is $usn <br />";
echo "address is $s1->address <br /> average is $s1->avg";

?>
```

Note: '$s1->address' and '$s1->avg' – can be accessed directly, without using any get function.

Two forms of the UML class diagram for our data encapsulated class are –

Class

Student
- name: String - USN: String + address: String + avg: float
+ __construct(int, String, String, String) + getname() : String + getUSN() : String + setname(String) : void + setUSN(String) : void

Class

Student
- name: String - USN: String + address: String + avg: float
+ __construct(int, String, String, String)

The first UML diagram includes, all the getter and setter methods. UML diagram may also exclude the getter and setter methods from a class as shown in the second diagram.

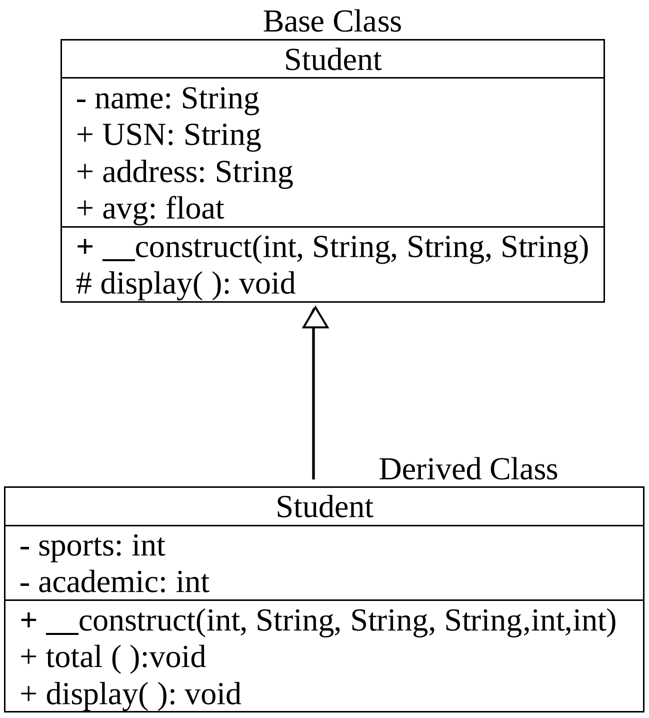
Inheritance

Inheritance enables you to create new PHP classes that reuse, extend, and modify the behavior that is defined in another PHP class. A class that is inheriting the properties and methods from another class is said to be a **subclass,** a **derived class** or a **child class**. The class that is being inherited is called a **superclass ,** a **base class or a parent class.**

When a class inherits from another class, it inherits all of its public and protected methods and properties. Just as in Java, a PHP class is defined as a subclass by using the extends keyword.

class Painting extends Art { . . . }

Inheritance in a UML class diagram -



<?php

```
Class Student
{
        private $name;
        public $USN;
        public $address;
        public $avg;

function __construct($n,$usn,$add,$avg)
{
        $this->name = $n;
        $this->USN= $usn;
        $this->address = $add;
        $this->avg = $avg;
}
protected function display()
{
    echo $this->name;
}
}

class FeeDetails extends Student
{
        private $sports;
        private $academic;

        function __construct($n,$usn,$add,$avg,$s,$ac)
        {
        parent::__construct($n,$usn,$add,$avg);
        $this->sports= $s;
        $this->academic= $ac;
        }

        public function total()
        {
                $sum= ($this->sports)+($this->academic);
                echo "total fee is $sum";
        }
        public function display()
        {
                parent::display();
                $this->total();
        }
}

$f1 = new FeeDetails("Sajeev", "1VA15CS013","Bangalore",66,25000,3000);
$f1->display();
?>
```

Referencing Base Class Members
A subclass inherits the public and protected members of the base class. In PHP any reference to a member in the base class requires the addition of the **parent::** prefix instead of the $this-> prefix. So within the FeeDetails class, a reference to the parent constructor or display() method would be done by using parent:: prefix.
It is important to note that private members in the base class are **not** available to its subclasses. Thus, within the FeeDetails class, a reference like the following would **not** work.
$abc = parent::name; *// would not work within the* FeeDetails *class*

If a member has to be available to subclasses but not anywhere else, it can be done by using the protected access modifier. 'name' member of base class can be accessed in base class only as it is private. It is displayed using '$this->name' in base class.

Inheriting Methods
Every method defined in the base/parent class can be overridden when extending a class, by declaring a function with the same name. A simple example of overriding is done in the previous program, the display( ) method of subclass FeeDetails overrides the display( ) method of parent class.
To access a public or protected method or property defined within a base class from within a subclass, use the member name with parent:: prefix.

Parent Constructors
To invoke a parent constructor from the derived class's constructor, use the parent:: syntax and call the constructor on the first line parent:: _ _construct(). This is similar to calling other parent methods, except that to use it we *must* call it at the beginning of constructor.

Polymorphism
Polymorphism is the OO concept, where the object can act differently at different times. Even if the same function name is used.

Overriding of methods in two or more derived classes takes place, the actual method called will depend on the type of the object. In the below example program, the display ( ) method is defined in both the derived classes. The method accessed depends on the object used to access the function.

```
<?php

Class Student
{
        private $name;
        public $USN;
        public $address;
        public $avg;

function __construct($n,$usn,$add,$avg)
{
        $this->name = $n;
        $this->USN= $usn;
        $this->address = $add;
        $this->avg = $avg;
}
protected function display()
{
    echo $this->name;
}
}

class FeeDetails extends Student
{
        private $sports;
        private $academic;

        function __construct($n,$usn,$add,$avg,$s,$ac)
        {
        parent::__construct($n,$usn,$add,$avg);
        $this->sports= $s;
        $this->academic= $ac;
        }

        public function total()
        {
                $sum= ($this->sports)+($this->academic);
                echo "total fee is $sum";
        }
        public function display()
        {
                parent::display();
                $this->total();
        }
}

class Scholarship extends Student
{

        function __construct($n,$usn,$add,$avg)
        {
        parent::__construct($n,$usn,$add,$avg);
        }

        public function display()
        {
                parent::display();
                echo "No fees";
        }
}

$f1 = new FeeDetails("Sajeev", "1VA15CS013","Bangalore",66,25000,3000);
$s1=new Scholarship("Amith", "1VA15CS013","Bangalore",66);
$f1->display();
$s1->display();

?>
```

OUTPUT:

```
Sajeev total fee is 28000 Amith No fees
```

The same display( ) method is invoked by two different objects. First the display( ) function of FeeDetails class is invoked and then the display( ) function of Scholarship class is invoked.

The run time decision of determining the function to be invoked at run time, is called **dynamic dispatching**. Just as each object can maintain its own properties, each object also manages its own table of methods. This means that two objects of the same type can have different implementations with the same name.

Object Interfaces

An object **interface** is a way of declaring a formal list of methods(only) without specifying their implementation. The class that uses the interface **must** implement the declared functions.

Interfaces provide a mechanism for defining what a class can do without specifying how it does it. Interfaces are defined using the interface keyword. It looks similar to PHP classes, except an interface contains no properties and its methods do not have method bodies defined.

Eg -

```
interface Viewable {
public function getmessage();
public function getData();
}
```

Notice that an interface contains only public methods, and instead of having a method body, each method is terminated with a semicolon.

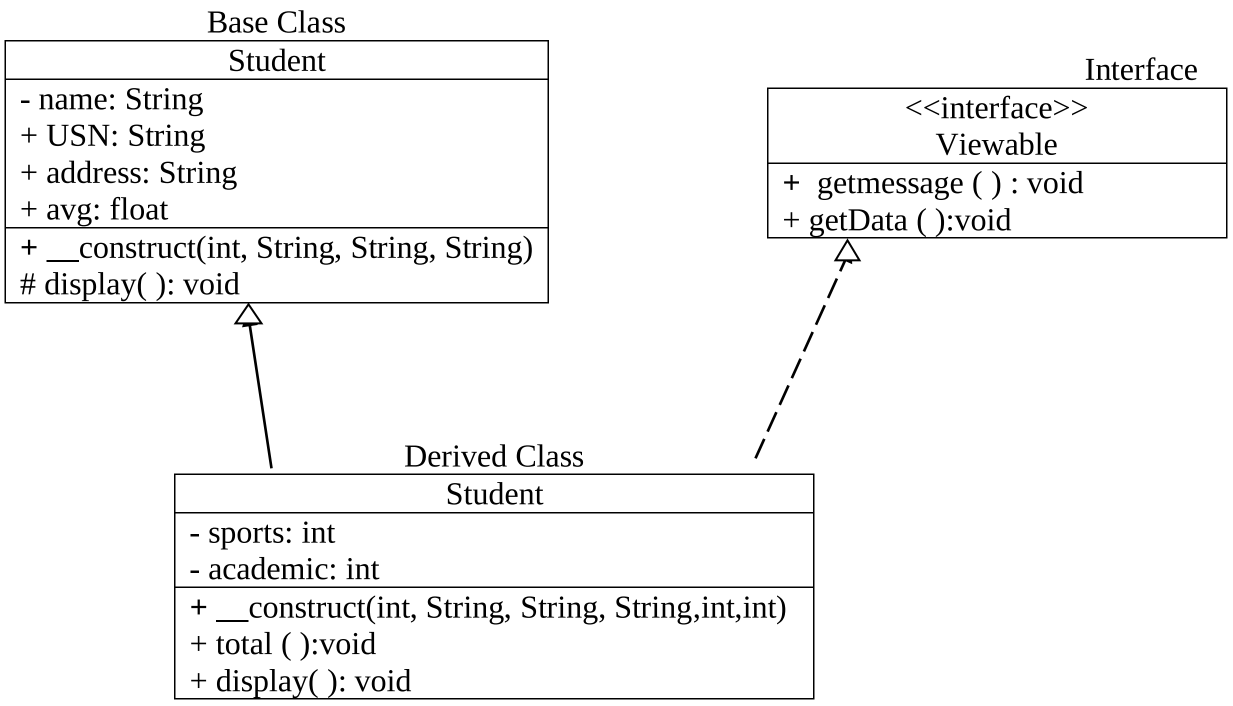
In PHP, a class can be said to *implement* an interface, using the **implements** keyword:

```
class FeeDetails extends Student implements Viewable
{ ... }
```

This means then that the class 'FeeDetails' must provide implementations for the getmessage() and getData() methods.

Implementing an interface allows a class to become more formal about the behavior it promises to provide. The same interface can be inherited by more than one class. It helps in the implementation of multiple inheritance.

PHP allows implementing two or more interfaces. The UML diagram to denote the interfaces include the <<interface>> stereotype. Classes that implement an interface are shown to implement using the same hollow triangles as inheritance but with dotted lines.



Runtime Class and Interface Determination
During the implementation, it might be required to know the class of a particular object or the interfaces used in the object and so on, this is possible in PHP using some of the built-in functions.

To display the class name of an object $x use the get_class() function:
echo get_class($x);

Similarly we can access the parent class with:
echo get_parent_class($x);

To determine what interfaces this class has implemented, use the function class_implements(), which returns an array of all the interfaces implemented by this class or its parents.
$allInterfaces = class_implements($x);

Error Handling and Validation

4.9 What Are Errors and Exceptions?

Even the best-written web application can suffer from runtime errors. Most complex web applications must interact with external systems such as databases, web services, email servers, file system, and other externalities that are beyond the developer's control. A failure in any one of these systems will mean that the web application will no longer run successfully. It is vitally important that web applications gracefully handle such problems.

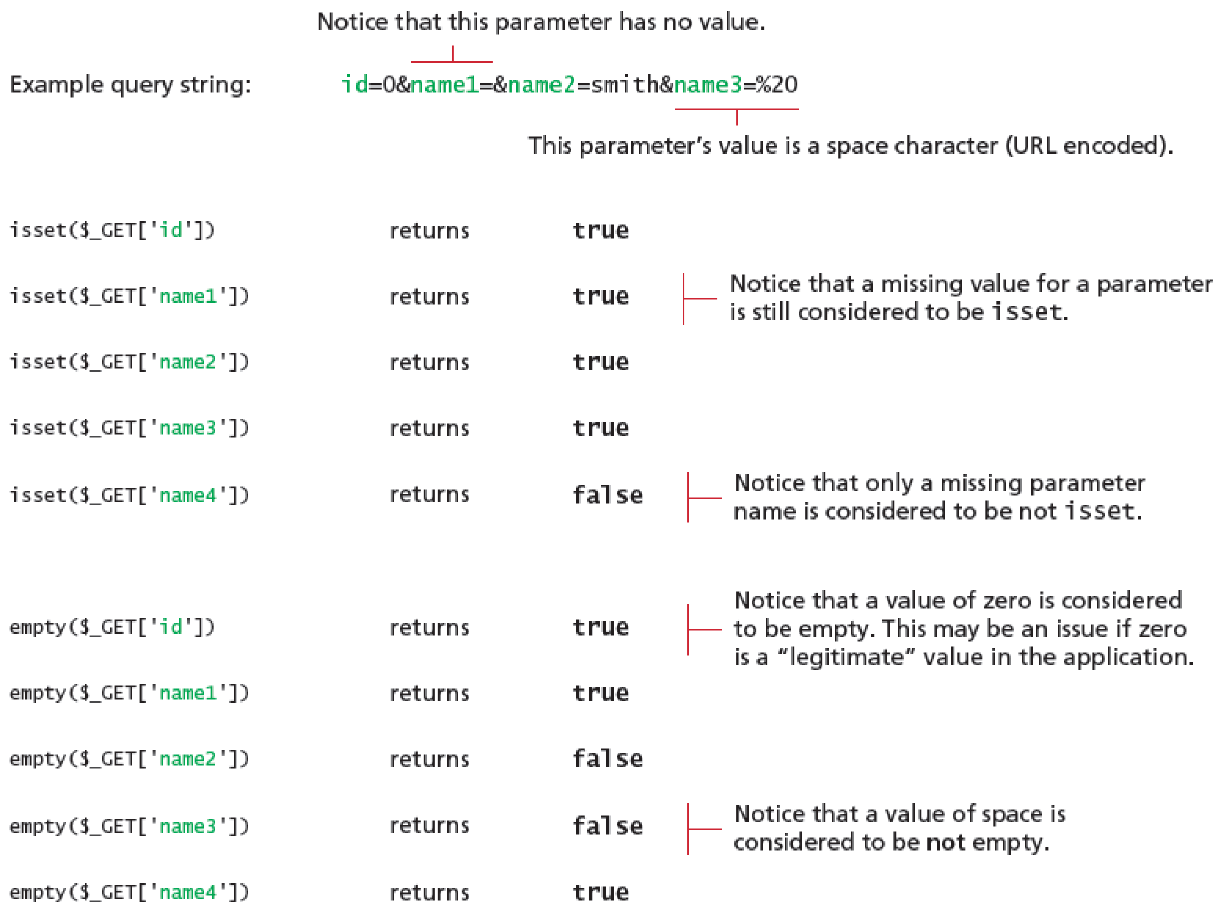
Types of Errors
There are three different types of website problems:

- Expected errors
- Warnings
- Fatal errors

An **expected error** is an error that routinely occurs during an application. The most common example of this type of error is as a result of user inputs, ie. , entering letters when numbers were expected. The web application should be developed such that, we expect the user to not always enter expected values. Users will leave fields blank, enter text when numbers were expected, type in too much or too little text, forget to click certain things, and click things they should not etc.

Not every expected error is the result of user input. Web applications that rely on connections to externalities such as database management systems, legacy software systems, or web services is expected to occasionally fail to connect. So there should be some type of logic that verifies the errors in code, check the user inputs and check if it contains the expected values.

PHP provides two functions for testing the value of a variable.
isset() function - which returns true if a variable is not null.
empty() function - which returns true if a variable is null, false, zero, or an empty string.



Notice that this parameter has no value.

Example query string: `id=0&name1=&name2=smith&name3=%20`

This parameter's value is a space character (URL encoded).

`isset($_GET['id'])`	returns	`true`	
`isset($_GET['name1'])`	returns	`true`	Notice that a missing value for a parameter is still considered to be `isset`.
`isset($_GET['name2'])`	returns	`true`	
`isset($_GET['name3'])`	returns	`true`	
`isset($_GET['name4'])`	returns	`false`	Notice that only a missing parameter name is considered to be not `isset`.
`empty($_GET['id'])`	returns	`true`	Notice that a value of zero is considered to be empty. This may be an issue if zero is a "legitimate" value in the application.
`empty($_GET['name1'])`	returns	`true`	
`empty($_GET['name2'])`	returns	`false`	
`empty($_GET['name3'])`	returns	`false`	Notice that a value of space is considered to be not empty.
`empty($_GET['name4'])`	returns	`true`	

To check a numeric value, use the is_numeric() function, as shown

```
$id = $_GET['id'];
if (!empty($id) && is_numeric($id) ) {
// use the query string since it exists and is a numeric value
...
}
```

Another type of error is **warnings**, which are problems that generate a PHP warning message (which may or may not be displayed) but will not halt the execution of the page. For instance, calling a function without a required parameter will generate a warning message but not stop execution. While not as serious as expected errors, these types of incidental errors should be eliminated by the programmer.

The **fatal errors**, which are serious in that the execution of the page will terminate unless handled in some way. These types of errors are exceptional and unexpected, such as a required input file being missing or a database table or field disappearing. These types of errors need to be reported so that the developer can try to fix the problem, and also the page needs to recover gracefully from the error so that the user is not excessively puzzled or frustrated.

Exceptions

In the context of PHP, error and exception is not the same. An **error** is some type of problem that generates a nonfatal warning message or that generates an error message that terminates the program's execution. An **exception** refers to objects that are of type Exception and which are used in conjunction with the object-oriented try . . . catch language construct for dealing with runtime errors.

4.10 PHP Error and Exception Handling

When a fatal PHP error occurs, program execution will eventually terminate unless it is handled. The PHP documentation provides two mechanisms for handling runtime errors: procedural error handling and the more object-oriented exception handling.

Procedural Error Handling

In the procedural approach to error handling, the programmer needs to explicitly test for error conditions after performing a task that might generate an error. Eg - While connecting to the database, we may need to test for and deal with errors after each operation that might generate an error state -

```
$connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME);
$error = mysqli_connect_error();
if ($error != null) {
// handle the error
...
}
```

This type of coding requires the programmer to know ahead of time what code is going to generate an error condition and also result in a great deal of code duplication. The advantage of the try . . . catch mechanism is that it allows the developer to handle a wider variety of exceptions in a single catch block.

Object-Oriented Exception Handling

When a runtime error occurs, PHP *throws* an *exception*. This exception can be *caught* and handled either by the function, class, or page that generated the exception or by the code that called the function or class. If an exception is not caught, then eventually the PHP environment will handle it by terminating execution with an "Uncaught Exception" message.

PHP also uses the try . . . catch programming construct to programmatically deal with exceptions at runtime. The catch construct expects some type of parameter of type Exception. The Exception built-in class provides methods for accessing not only the exception message, but also the line number of the code that generated the exception and the stack trace, both of which can be helpful for understanding where and when the exception occurred.

```
// Exception throwing function
function throwException($message = null,$code = null)
{
        throw new Exception($message,$code);
}
try {
        // PHP code here
        $connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME)
        or throwException("error");
        //...
}
catch (Exception $e) {
        echo ' Caught exception: ' . $e->getMessage();
        echo ' On Line : ' . $e->getLine();
        echo ' Stack Trace: '; print_r($e->getTrace());
}

finally {
// PHP code here that will be executed after try or after catch
}
```

The finally block is optional. Any code within it will always be executed *after* the code in the try or in the catch blocks, even if that code contains a return statement. It is typically used if the developer wants certain things to do regardless of whether an exception occurred, such as closing a connection or removing temporary files.

It is also possible in PHP to programmatically throw an exception via the throw keyword. An exception is thrown when an expected programming assumption is not met. It also possible to rethrow an exception within a catch block.

```
function processArray($array)
{
// make sure the passed parameter is an array with values
if ( empty($array) ) {
        throw new Exception('Array with values expected');
}
// process the array code
...
}
```

Rethrowing the exception

```
try {
// PHP code here
}
catch (Exception $e) {
        // do some application-specific exception handling here
        ...
        // now rethrow exception
        throw $e;   // same exception is rethrown
}
```

4.11 PHP Error Reporting

PHP has a flexible and customizable system for reporting warnings and errors that can be set programmatically at runtime or declaratively at design-time within the **php.ini** file. There are three main error reporting flags:

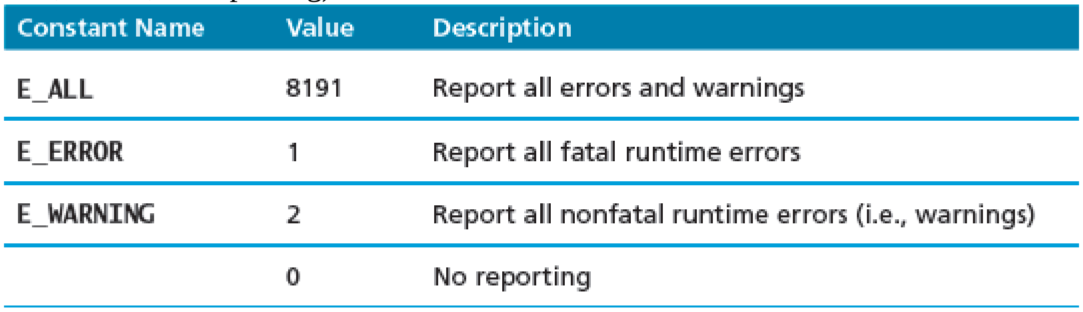
- error_reporting
- display_errors
- log_errors

Note: PHP.ini is a configuration file, that is used to customize behavior of PHP at runtime. It consists of settings for different values such as register global variables, display errors, log errors, max uploading size setting, maximum time to execute a script and other configurations. When PHP Server starts up it looks for PHP.ini file first to load various values for settings.

The error_reporting Setting
The error_reporting setting specifies which type of errors are to be reported. It can be set programmatically inside any PHP file by using the error_reporting() function:
error_reporting(E_ALL);

It can also be set within the **php.ini** file:
error_reporting = E_ALL

The possible levels for error_reporting are defined by predefined constants(the default setting is zero, that is, no reporting).



Constant Name	Value	Description
E_ALL	8191	Report all errors and warnings
E_ERROR	1	Report all fatal runtime errors
E_WARNING	2	Report all nonfatal runtime errors (i.e., warnings)
	0	No reporting

The display_errors Setting
The display_error setting specifies whether error messages should or should not be displayed in the browser. It can be set programmatically via the ini_set() function:
ini_set('display_errors','0');

It can also be set within the **php.ini** file:
display_errors = Off

The log_error Setting
The log_error setting specifies whether error messages should or should not be sent to the server error log. It can be set programmatically via the ini_set() function:
ini_set('log_errors','1');

It can also be set within the **php.ini** file:

```
log_errors = On
```

When logging is turned on, error reporting will be sent to either the operating system's error log file or to a specified file in the site's directory. If error messages is to be saved in a log file in the user's directory, the file name and path can be set via the error_log setting.

```
ini_set('error_log', '/restricted/my-errors.log');
```

It can also be set within the **php.ini** file:

```
error_log = /restricted/my-errors.log
```

You can also programmatically send messages to the error log at any time via the error_log()

```
$msg = 'Some horrible error has occurred!';
// send message to system error log (default)
error_log($msg,0);
// email message
error_log($msg,1,'support@abc.com','From: somepage.php@abc.com');
// send message to file
error_log($msg,3, '/folder/somefile.log');
```

Important questions –

1. How is array defined in PHP? Explain the array operations of PHP.
2. What are superglobal arrays? List them
3. Explain the process of populating $_GET superglobal array.
4. Explain the process of populating $_POST superglobal array.
5. Explain the $_SERVER array
6. Explain the use of $_FILES array.
7. Explain how the files are uploaded from the browser and values are accessed at the server
8. How is the size of file restricted in PHP
9. Explain the stream access of file in PHP.
10. Explain the All-In-Memory access of file in PHP.
11. How are the classes declared in PHP
12. How to access methods and properties of class in PHP
13. Explain the use of static members
14. How are constructors defined in PHP. Explain with example.
15. Explain encapsulation with example.
16. Explain inheritance with example.
17. Explain polymorphism with example.
18. Explain interface with example.
19. How is exception handled in PHP.
20. What are the different types of errors
21. Explain the different error reporting flags of PHP.
22. Wrie a PHP script to move the uploaded file to the desired location.
23. Wrie a PHP script to limit the type of file.