MODULE 4

Chapter 1: PHP Arrays and Superglobals

1. Arrays
2. $_GET and $_POST Superglobal Arrays
3. $_SERVER Array
4. $_Files Array
5. Reading/Writing Files

1.1 Arrays

- An array is a data structure that allows the programmer to collect a number of related elements together in a single variable.
- Unlike most other programming languages, in PHP an array is actually an **ordered map**, which associates each value in the array with a key.
- This flexibility allows us to use arrays in PHP in a manner similar to other languages' arrays, but we can also use them like other languages' collection classes.

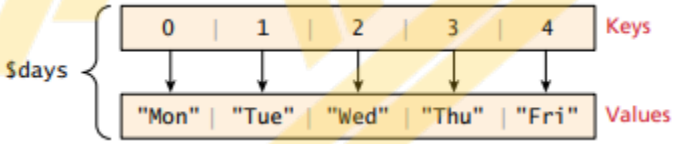

Figure 1.1: Visualization of a key-value array

- **Array keys**
  - In most programming languages array keys are limited to integers, i.e., start at 0,and go up by 1.
  - In PHP, keys *must* be either integers or strings and need not be sequential.
  - This means we cannot use an array or object as a key (doing so will generate an error).
  - One should be especially careful about mixing the types of the keys for an array since PHP performs cast operations on the keys that are not integers or strings.
- **Array values**
  - Unlike keys, are not restricted to integers and strings.

- They can be any object, type, or primitive supported in PHP.
- We can even have objects of our own types, so long as the keys in the array are integers and strings.

Defining and Accessing an Array

- Let us begin by considering the simplest array, which associates each value inside of it with an integer index (starting at 0).
- The following declares an empty array named days:

```
$days = array();
```

- To define the contents of an array as strings for the days of the week as shown in Figure 1.1, we declare it with a comma-delimited list of values inside the ( )braces using either of two following syntaxes:

```
$days = array("Mon","Tue","Wed","Thu","Fri");
$days = ["Mon","Tue","Wed","Thu","Fri"];     // alternate syntax
```

- In the above example, because no keys are explicitly defined for the array, the default key values are 0, 1, 2, . . . , n.
- Notice that we do not have to provide a size for the array: arrays are dynamically sized as elements are added to them.
- Elements within a PHP array are accessed in a manner similar to other programming languages, that is, using the familiar square bracket notation.
- The code example 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
```

- We could also define the array elements individually using the same 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";
```

- In PHP, we are also able to explicitly define the keys in addition to the values.
- This allows us to use keys other than the classic 0, 1, 2, . . , n to define the indexes of an array.

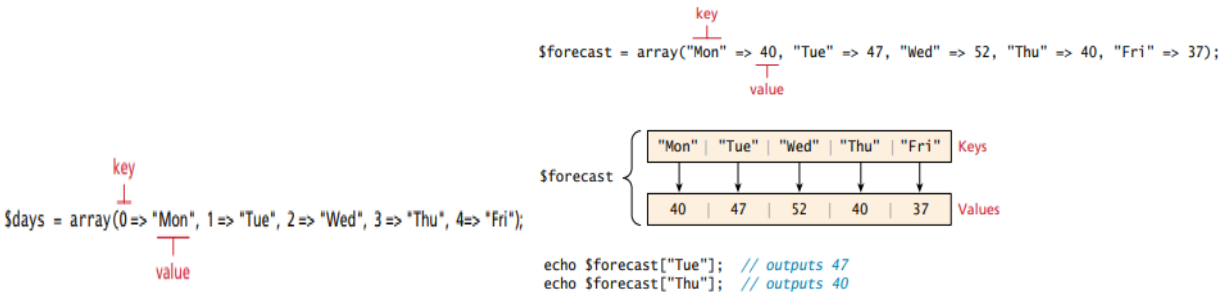

Figure 1.2: Assigning keys to values (left) and Array with strings as keys and integers as values (left)

- Explicit control of the keys and values opens the door to keys that do not start at 0, are not sequential, and that are not even integers (but rather strings).
- This is why we can also consider an array to be a dictionary or hash map.
- 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. e
- To access an element in an associative array, we simply use the key value rather than an index. `echo $forecast["Wed"]; // this will output 52`

Multidimensional Arrays

- PHP also supports multidimensional arrays.
- The values for an array can be any PHP object, which includes other arrays.

```
$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
```

```
$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
```

Figure 1.3: Illustrates the structure of these two multidimensional arrays.

Iterating through an Array

- One of the most common programming tasks that we will perform with an array is to iterate through its contents.

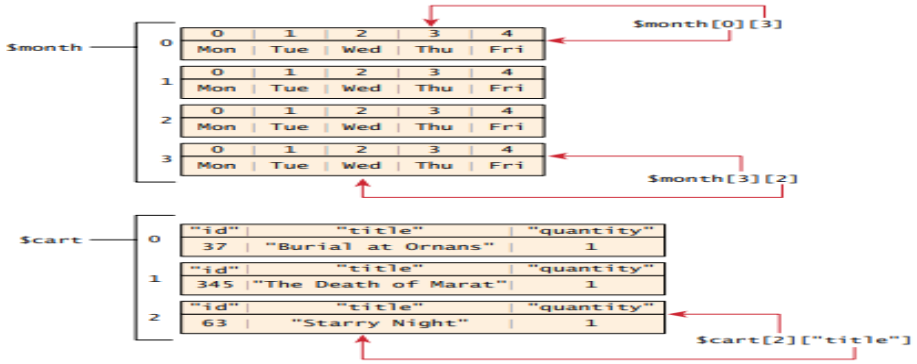

Figure 1.4: Visualizing multidimensional arrays

- Listing 1.1 illustrates how to iterate and output the content of the $days array using the built-in function count() along with examples using while, do while, and for loops.

```
// 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>";
}
```

Listing 1.1: Iterating an array using while, do while, and for loops

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

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

Listing 1.2: Iterating through an associative array using a foreach loop

- The challenge of using the classic loop structures is that when we have nonsequential integer keys (i.e., an associative array), we can't write a simple loop that uses the $i++ construct.
- To overcome this challenge we need to use iterator to move through an associative array as shown in Listing 1.2.

Adding and Deleting Elements

- In PHP, arrays are dynamic, i.e., they 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, as shown below:

```
$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 same style of assignment replaces the value at that key.

- As an alternative to specifying the index, a new element can be added to the end of any array using the following technique:

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

- The advantage to this approach is that we don't have to worry about skipping an index key. PHP is more than happy to let us "skip" an index, as shown in the following example.

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

- Output for the above snippet is:

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

  - That is, there is now a "gap" in our array that will cause problems if we try iterating through it using the techniques.
  - If we try referencing $days[6], for instance, it will return a **NULL** value, which is a special PHP value that represents a variable with no value.
- We can also create "gaps" by explicitly deleting array elements using the unset() function as shown in Listing 1.3.

```
$days = array("Mon","Tue","Wed","Thu","Fri");

unset($days[2]);
unset($days[3]);

print_r($days); // outputs: Array ( [0] => Mon [1] => Tue [4] => Fri )

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

Listing 1.3: Deleting elements

```
$oddKeys = array (1 => "hello", 3 => "world", 5 => "!");
if (isset($oddKeys[0])) {
   // The code below will never be reached since $oddKeys[0] is not set!
   echo "there is something set for key 0";
}
if (isset($oddKeys[1])) {
   // This code will run since a key/value pair was defined for key 1
   echo "there is something set for key 1, namely ". $oddKeys[1];
}
```

Listing 1.4: nonsequential keys and usage of isset( )

- **Checking If a Value Exists:**
  - Since array keys need not be sequential, and need not be integers, we may run into a scenario where we want to check if a value has been set for a particular key.
  - As with undefined null variables, values for keys that do not exist are also undefined.
  - To check if a value exists for a key, we can therefore use the isset() function, which returns true if a value has been set, and false otherwise.
  - Listing 1.4 defines an array with noninteger indexes, and shows the result of asking isset() on several indexes.

Array Sorting

- There are many built-in sort functions, which sort by key or by value. To sort the $days array by its values we would simply use:

```
sort($days);
```

  - As the values are all strings, the resulting array would be:

```
Array ([0] => Fri [1] => Mon [2] => Sat [3] => Sun [4] => Thu
       [5] => Tue [6] => Wed)
```

- However, the above sort loses the association between the values and the keys!
- A better sort, one that would have kept keys and values associated together, is:

```
asort($days);
```

  - The resulting array in this case is:

```
Array ([4] => Fri [0] => Mon [5] => Sat [6] => Sun [3] => Thu
       [1] => Tue [2] => Wed)
```

More Array Operations

In addition to the powerful sort functions, there are other convenient functions you can use on arrays.

- **array_keys($someArray)**: This method returns an indexed array with the values being the keys of $someArray.
  - For example, print_r(array_values($days)) outputs

```
Array ( [0] => 0 [1] => 1 [2] => 2 [3] => 3 [4] => 4 )
```

- **array_values($someArray):** This function returns an indexed array with the values being the values of $someArray.
  - For example, print_r(array_values($days)) outputs

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

- **array_rand($someArray, $num=1):** This function returns as many random keys as are requested. If we only want one, the key itself is returned; otherwise, an array of keys is returned.
  - For example, print_r(array_rand($days,2)) might output: `Array (3, 0)`
- **array_reverse($someArray):** This method returns $someArray in reverse order. The passed $someArray is left untouched.

- For example, print_r(array_reverse($days)) outputs:

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

- **array_walk($someArray, $callback, $optionalParam):** It allows us to call a method ($callback), for each value in $someArray. The $callback function typically takes two parameters, the value first, and the key second.
  - An example that simply 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;
}
```

- **in_array($needle, $haystack):** This method lets us to search array $haystack for a value ($needle). It returns true if it is found, and false otherwise.
- **shuffle($someArray):** This method shuffles $someArray. Any existing keys are removed and $someArray is now an indexed array if it wasn't already.

Superglobal Arrays

- PHP uses special predefined associative arrays called superglobal variables that allow the programmer to easily access HTTP headers, query string parameters, and other commonly needed information.
- They are called superglobal because these arrays are always in scope and always exist, ready for the programmer to access or modify them without having to use the global keyword.

Name	Description
$GLOBALS	Array for storing 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

Table 1.1: Superglobal variables

1.2 $_GET and $_POST Superglobal Arrays

- The $_GET and $_POST arrays are the most important superglobal variables in PHP since they allow the programmer to access data sent by the client in a query string.
- An HTML form (or an HTML link) allows a client to send data to the server.
- That data is formatted such that each value is associated with a name defined in the form.
- If the form was submitted using an HTTP GET request, then the resulting URL will contain the data in the query string.
- PHP will populate the superglobal $_GET array using the contents of this query string in the URL.

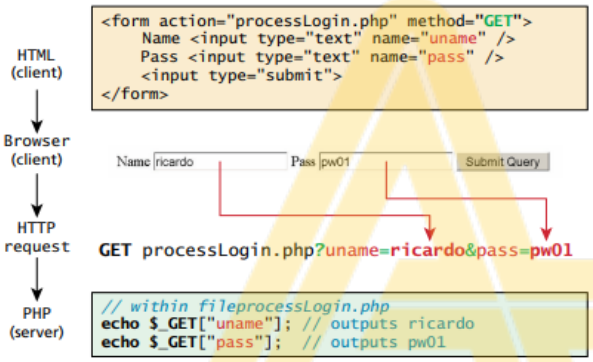

Figure1.5: HTTP request to $_GET array

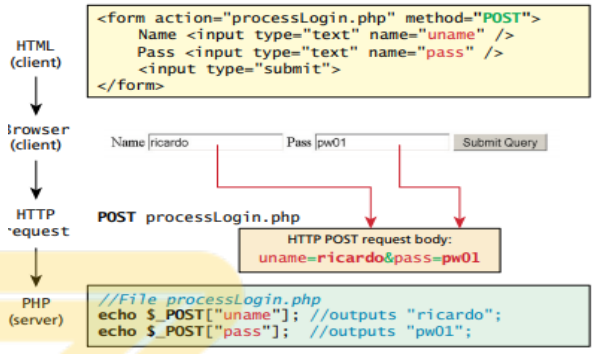

Figure1.6: HTTP request to $_POST array

- If the form was sent using HTTP POST, then the values would not be visible in the URL, but will be sent through HTTP POST request body.
- From the PHP programmer's perspective, almost nothing changes from a GET data post except that those values and keys are now stored in the $_POST array.
- This mechanism greatly simplifies accessing the data posted by the user, since we need not parse the query string or the POST request headers.

Determining If Any Data Sent

- PHP can use the same file to handle both the display of a form as well as the form input.
  - For **example**, a single file is often used to display a login form to the user, and that same file also handles the processing of the submitted form data, as shown in Figure 1.7.
  - In such cases we may want to know whether any form data was submitted at all using either POST or GET.

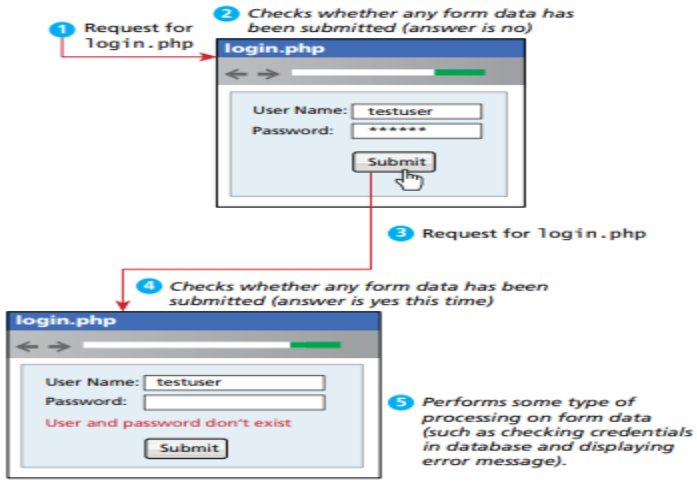

Figure 1.7: Form display and processing by the same PHP page

Accessing Form Array Data

- Sometimes in HTML forms you might have multiple values associated with a single name;
- Listing 1.5 provides another example. Notice that each checkbox has the same name value (name="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>
```

Listing 1.5: HTML that enables multiple values for one name

```
<?php

echo "You submitted " . count($_GET['day']) . "values";
foreach ($_GET['day'] as $d) {
   echo $d . ", ";
}

?>
```

Listing 1.6: PHP code to display an array of checkbox variables

- If the user selects more than one day and submits the form, the $_GET['day'] value in the superglobal array will only contain the last value from the list that was selected.
- To overcome this limitation, you must change the HTML in the form. In particular, you will have to change the name attribute for each checkbox from day to day[].

```
Monday <input type="checkbox" name="day[]" value="Monday" />
Tuesday <input type="checkbox" name="day[]" value="Tuesday" />
...
```

- Listing 1.6 to echo the number of days selected and their values.

Using Query Strings in Hyperlinks

- WKT, form information packaged in a query string is transported to the server in one of two locations depending on whether the form method is GET or POST.
- It is also important to realize that making use of query strings is not limited to only data entry forms.
- It is possible to combine query strings with anchor tags . . Anchor tags (i.e., hyperlinks) also use the HTTP GET method
  - **EX:** Our database may have hundreds or thousands of books in it: surely it would be too much work to create a separate page for each book!
  - It would make a lot more sense to have a single Display Book page that receives as input a query string that specifies which book to display, as shown in Figure 1.8.

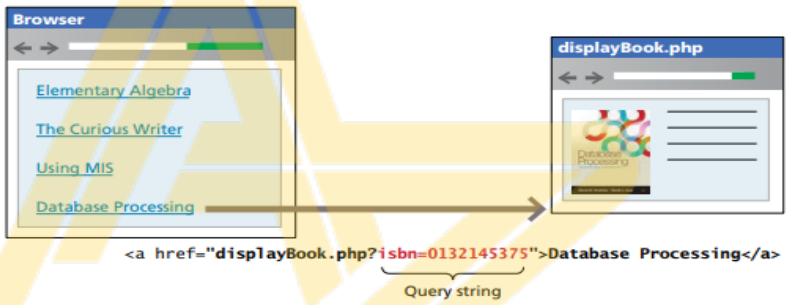

Figure 1.8: Sensible approach to displaying individual items using query strings

Sanitizing Query Strings

- The process of checking user input for incorrect or missing information is sometimes referred to as the process of sanitizing user inputs.

```
// This uses a database API . . . we will learn about it in Chapter 11
$pid = mysqli_real_escape_string($link, $_GET['id']);

if ( is_int($pid) ) {
    // Continue processing as normal
}
else {
    // Error detected. Possibly a malicious user
}
```

Listing 1.7: Simple sanitization of query string values

- Our program must be able to handle the following cases for every query string or form value.
  - If query string parameter doesn't exist.
  - If query string parameter doesn't contain a value.

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

1.3 $_SERVER Array

- The $_SERVER associative array contains a variety of information as follows.
  - It contains some of the information contained within HTTP request headers sent by the client.
  - It also contains many configuration options for PHP itself as shown in figure 1.9

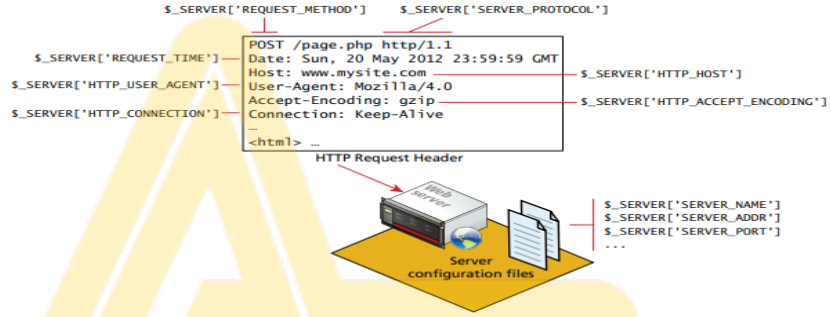

Figure 1.9: Relationship between request headers, the server, and the $_SERVER array

- To use the $_SERVER array, we simply refer to the relevant case-sensitive key name:

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

- It is worth noting that because the entries in this array are created by the webserver, not every key listed in the PHP documentation will necessarily be available.
- A complete list of keys contained within this array is listed in the online PHP documentation, but we will cover some of the critical ones here.

Server Information Keys

- SERVER_NAME is a key in the $_SERVER array that contains the name of the site that was requested. If we are running multiple hosts on the same code base, this can be a useful piece of information.
- SERVER_ADDR is a complementary key telling us the IP of the server. Either of these keys can be used in a conditional to output extra HTML to identify a development server.

- DOCUMENT_ROOT tells us the file location from which we are currently running our script.
- Since we are often moving code from development to production, this key can be used to great effect to create scripts that do not rely on a particular location to run correctly.
- This key complements the SCRIPT_NAME key that identifies the actual script being executed.

Request Header Information Keys

- Recall that the web server responds to HTTP requests, and that each request contains a request header.
- These keys provide programmatic access to the data in the request header.
- The REQUEST_METHOD key returns the request method that was used to access the page: that is, GET, HEAD, POST, PUT.
- The REMOTE_ADDR key returns the IP address of the requestor, which can be a useful value to use in your web applications.
- In real-world sites these IP addresses are often stored to provide an audit trail of which IP made which requests, especially on sensitive matters like finance and personal information.

```php
<?php
echo $_SERVER['HTTP_USER_AGENT'];

$browser = get_browser($_SERVER['HTTP_USER_AGENT'], true);
print_r($browser);
?>
```

Listing 1.8: Accessing the user-agent string in the HTTP headers

```php
$previousPage = $_SERVER['HTTP_REFERER'];
// Check to see if referer was our search page
if (strpos("search.php",$previousPage) != 0) {
   echo "<a href='search.php'>Back to search</a>";
}
// Rest of HTML output
```

Listing 1.9: Using the HTTP_REFERER header to provide context-dependent output

- One of the most commonly used request headers is the user-agent header, which contains the operating system and browser that the client is using. This header value can be accessed using the key HTTP_USER_AGENT as shown in Listing 1.8.
- HTTP_REFERER is an especially useful header. Its value contains the address of the page that referred us to this one (if any) through a link. It is commonly used in analytics to determine which pages are linking to our site as shown in Listing 1.9.

1.4 $_FILES Array

- The $_FILES associative array contains items that have been uploaded 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.

HTML Required for File Uploads

- To allow users to upload files, there are some specific things you must do:
- First, we must ensure that the HTML form uses the HTTP POST method, since transmitting a file through the URL is not possible.
- Second, we must add the enctype="multipart/form-data" attribute to the HTML form that is performing the upload so that the HTTP request can submit multiple pieces of data.
- Finally we must include an input type of file in our form. This will show up with a browse button beside it so the user can select a file from their computer to be uploaded.

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

Listing 1.10: HTML for a form that allows an upload

Handling the File Upload in PHP

- The corresponding PHP file responsible for handling the upload will utilize the superglobal $_FILES array.
- 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.

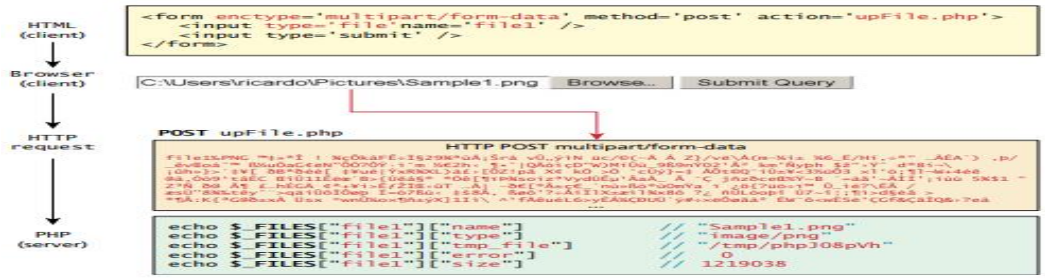

Figure 1.10: Data flow from HTML form through POST to PHP $_FILES array

- Figure 1.10 illustrates the process of uploading a file to the server and how the corresponding upload information is contained in the $_FILES array.
- The values for each of the keys, in general, are described below.
  - **Name** is a string containing the full file name used on the client machine, including any file extension. It does not include the file path on the client's machine.
  - **Type** defines the MIME type of the file. This value is provided by the client browser and is therefore not a reliable field.
  - **tmp_name** is the full path to the location on your server where the file is being temporarily stored. The file will cease to exist upon termination of the script, so it should be copied to another location if storage 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.

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, and should be looked for before proceeding any further.
- The full list of errors is provided in Table 1.2 and shows that there are many causes for bad file uploads.

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.

Table 1.2: Error Codes in PHP for File Upload

```
foreach ($_FILES as $fileKey => $fileArray) {
   if ($fileArray["error"] != UPLOAD_ERR_OK) { // error
      echo "Error: " . $fileKey . " has error" . $fileArray["error"]
          . "<br>";
   }
   else {   // no error
      echo $fileKey . "Uploaded successfully ";
   }
}
```

Listing 1.11: Checking each file uploaded for errors

- A proper file upload script will therefore check each uploaded file by checking the various error codes as shown in listing 1.11.

File Size Restrictions

- Some scripts limit the file size of each upload.
- There are many reasons to do so, and ideally we would prevent the file from even being transmitted in the first place if it is too large.
- There are three main mechanisms for maintaining uploaded file size restrictions:
    - **via HTML in the input form**,
        - This technique allows your php.ini maximum file size to be large, while letting some forms override that large limit with a smaller one

```
<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>
```

Listing 1.12: Limiting upload file size via HTML

    - **via JavaScript in the input form, and**
        - As intuitive as it is, this hidden field can easily be overridden by the client, and is therefore unacceptable as the only means of limiting size.

```
<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>
```

Listing 1.13: Limiting upload file size via JavaScript

    - **via PHP coding.**
        - 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);
}
```

Listing 1.14: Limiting upload file size via PHP

Limiting the Type of File Upload

- Even if the upload was successful and the size was within the appropriate limits, we may still have a problem.
- What if we wanted the user to upload an image and they uploaded a Microsoft Word document? We might also want to limit the uploaded image to certain image types, such as jpg and png, while disallowing bmp and others.
- To accomplish this type of checking you typically examine the file extension and the type field and also to compare the type to valid image types.
- Listing 1.15 shows sample code to check the file extension of a file, and also to compare the type to valid image types.

```
$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";
   }
}
```

Listing 1.15: PHP code to look for valid mime types and file extensions

```
$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";
}
```

Listing 1.16: Using move_uploaded_file() function

Moving the File

- With all of our checking completed, we may now finally want to move the temporary file to a permanent location on your server.
- Typically, we make use of the PHP function move_uploaded_file(), which takes in the temporary file location and 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, and a warning may be output.
- Listing 1.16 illustrates a simple use of the function.

1.5 Reading/Writing Files

- Before the age of the ubiquitous database, software relied on storing and accessing data in files.
- In web development, the ability to read and write to text files remains an important technical competency.

- Even if our site uses a database for storing its information, the fact that the PHP file functions can read/write from a file or from an external website (i.e., from a URL) means that file system functions still have relevance even in the age of database-driven websites.
- There are two basic techniques for read/writing files in PHP:
  - **Stream access.** In this technique, our code will read just a small portion of the file at a time. While this does require more careful programming, it is the most memory-efficient approach when reading very large files.
  - **All-In-Memory access.** In this technique, we can read the entire file 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

- In the C-style file access you separate the acts of opening, reading, and closing a file.
- The function fopen() takes a file location or URL and access mode as parameters.
- Some of the common modes are "r" for read, "rw" for read and write, and "c," which creates a new file for writing.
- Once the file is opened, we can read from it in several ways.
  - To read a single line, use the fgets() function, which will return false if there is no more data,
  - To read an arbitrary amount of data (typically for binary files), use fread() and for reading a single character use fgetsc().
  - Finally, when finished processing the file you must close it using fclose().
- Listing 1.17 illustrates a script using fopen(), fgets(), and fclose() to read a file and echo it out.
- To write data to a file, we can employ the fwrite() function in much the same way as fgets(), passing the file handle and the string to write.

```
$f = fopen("sample.txt", "r");
$ln = 0;
while ($line = fgets($f)) {
   $ln++;
   printf("%2d: ", $ln);
   echo $line . "<br>";
}
fclose($f);
```

Listing 9.19 Opening, reading lines, and closing a file

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

Table 1.3: In-Memory File Functions

All-In-Memory access

- While the previous approach to reading/writing files gives you complete control, the programming requires more care in dealing with the streams, file handles, and other low-level issues.
- The alternative simpler approach is much easier to use, at the cost of relinquishing fine-grained control as shown in table 1.3.
- 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, you use

```
file_put_contents(FILENAME, $writeme);
```

- These functions are especially convenient when used in conjunction with PHP's many powerful string-processing functions.

Chapter 2: PHP Classes and Objects

1. Object-Oriented Overview
2. Classes and Objects in PHP
3. Object Oriented Design

2.1 Object-Oriented Overview

Terminology

- The notion of programming with objects allows the developer to think about an item with particular properties (attributes /data members) and methods (functions).
- The structure of these objects are defined by classes, which outline the properties and methods like a blueprint.
- Each variable created from a class is called an object or instance, and each object maintains its own set of variables, and behaves independently from the class once created.

- Figure 2.1 illustrates the differences between a class, which defines an object's properties and methods, and the objects or instances of that class.

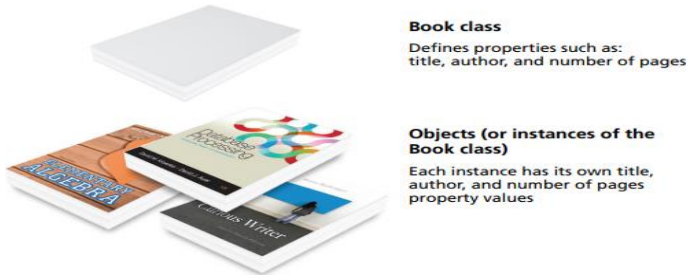

Figure 2.1: Relationship between a class and its objects

The Unified Modeling Language

- The standard diagramming notation for object-oriented design is UML (Unified Modeling Language).
- UML is a succinct set of graphical techniques to describe software design which gives the visualization / representation of objects and classes.
- Several types of UML diagram are defined. Class diagrams and object diagrams, in particular, are useful to us when describing the properties, methods, and relationships between classes and objects.
- **Ex:** To illustrate classes and objects in UML, consider the artist we have looked at in the Art Case Study. Every artist has a first name, last name, birth date, birth city, and death date. Using objects we can encapsulate those properties together into a class definition for an Artist.

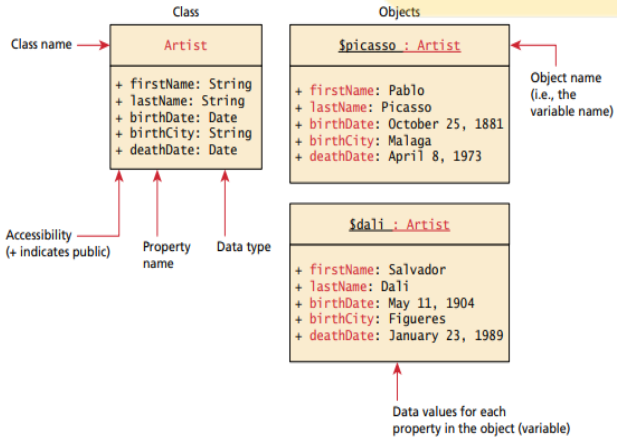

Figure 2.2: Relationship between a class and its objects in UML

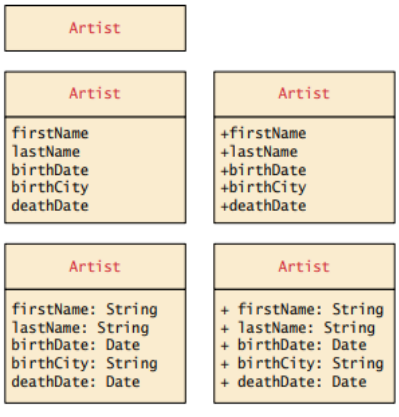

Figure 2.3: Different levels of UML detail

Differences between Server and Desktop Objects

- One important distinction between web programming and desktop application programming is that the objects you create (normally) only exist until a web script is terminated.
- While desktop software can load an object into memory and make use of it for several user interactions, a PHP object is loaded into memory only for the life of that HTTP request.
- Figure 10.4 shows an illustration of the lifetimes of objects in memory between a desktop and a browser application.
- For this reason, we must use classes differently than in the desktop world, since the object must be recreated and loaded into memory for each request that requires it.
- Object-oriented web applications can see significant performance degradation compared to their functional counterparts if objects are not utilized correctly.
- Unlike a desktop, there are potentially many thousands of users making requests at once, so not only objects are destroyed upon responding to each request, but memory must be shared between many simultaneous requests, each of which may load objects into memory.

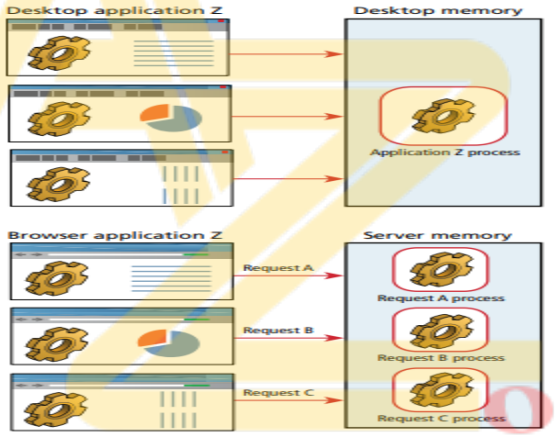

Figure 2.4: Lifetime of objects in memory in web versus desktop applications

2.2 Classes and Objects in PHP

- Classes should be defined in their own files so they can be imported into multiple scripts.
- Any PHP script can make use of an external class by using one of the include statements or functions, that is, include, include_once, require, or require_once.
- Once a class has been defined, we can create as many instances of that object as memory will allow using the new keyword.

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.
- The Artist class with the properties illustrated in Figure 2.2 is defined using PHP in Listing 2.1.
- Each property in the class is declared using one of the keywords public, protected, or private followed by the property or variable name.

```
class Artist {
    public   $firstName;
    public   $lastName;
    public   $birthDate;
    public   $birthCity;
    public   $deathDate;
}
```

```
$picasso = new Artist();
$dali = new Artist();
$picasso->firstName = "Pablo";
$picasso->lastName = "Picasso";
$picasso->birthCity = "Malaga";
$picasso->birthDate = "October 25 1881";
$picasso->deathDate = "April 8 1973";
```

Listing 2.1: Artist class **Listing 2.2** Instantiating two Artist objects and setting its object's properties

Instantiating Objects

- It's important to note that defining a class is not the same as using it.
- To make use of a class, one must **instantiate** (create) objects from its definition using the new keyword.
- To create two new instances of the Artist class called $picasso and $dali, we instantiate two new objects using the new keyword as follows:

```
$picasso = new Artist();
$dali = new Artist();
```

Properties

- Once we have instances of an object, we can access and modify the properties of each one separately using the variable name and an arrow (→), which is constructed from the dash and greater than symbols.
- Listing 2.2 Shows code that defines the two Artist objects and then sets all the properties for the $picasso object.

Constructors

- Listing 2.2 takes multiple lines and every line of code introduces potential maintainability problems, especially when we define more artists.
- Inside of a class definition, we should therefore define constructors.
- In PHP, constructors are defined as functions with the name __construct().

- Listing 2.3 shows an updated Artist class definition that now includes a constructor.
- Inside of a class you must always use the $this syntax to reference all properties and methods associated with this particular instance of a class.

```
class Artist {
   // variables from previous listing still go here
   ...

   function __construct($firstName, $lastName, $city, $birth,
                        $death=null) {
      $this->firstName = $firstName;
      $this->lastName = $lastName;
      $this->birthCity = $city;
      $this->birthDate = $birth;
      $this->deathDate = $death;
   }
}
```

Listing 2.3: A constructor added to the class definition

```
class Artist {
   ...
   public function outputAsTable() {
      $table = "<table>";
      $table .= "<tr><th colspan='2'>";
      $table .= $this->firstName . " " . $this->lastName;
      $table .= "</th></tr>";
      $table .= "<tr><td>Birth:</td>";
      $table .= "<td>" . $this->birthDate;
      $table .= "(" . $this->birthCity . ")</td></tr>";
      $table .= "<tr><td>Death:</td>";
      $table .= "<td>" . $this->deathDate . "</td></tr>";
      $table .= "</table>";
      return $table;
   }
}
```

Listing 2.4: Method definition

- This new constructor can then be used when instantiating as shown below

```
$picasso = new Artist("Pablo","Picasso","Malaga","Oct 25,1881",
                      "Apr 8,1973");
$dali = new Artist("Salvador","Dali","Figures","May 11 1904",
                   "Jan 23 1989");
```

Methods

- Objects are useful when we define behavior or operations that they can perform.
- In object-oriented lingo these operations are called methods and are like functions, except they are associated with a class as shown in listing 2.4.
- They define the tasks each instance of a class can perform and are useful since they associate behavior with objects.
- To output the artist, you can use the reference and method name as follows:

```
$picasso = new Artist( . . . )
echo $picasso->outputAsTable();
```

- Updated UML diagram is shown below.

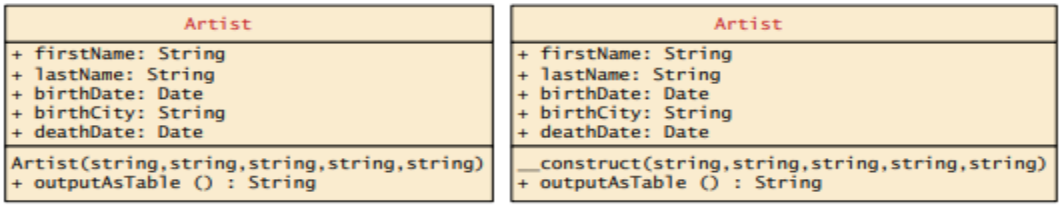

Figure 2.5: Updated class diagram

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.

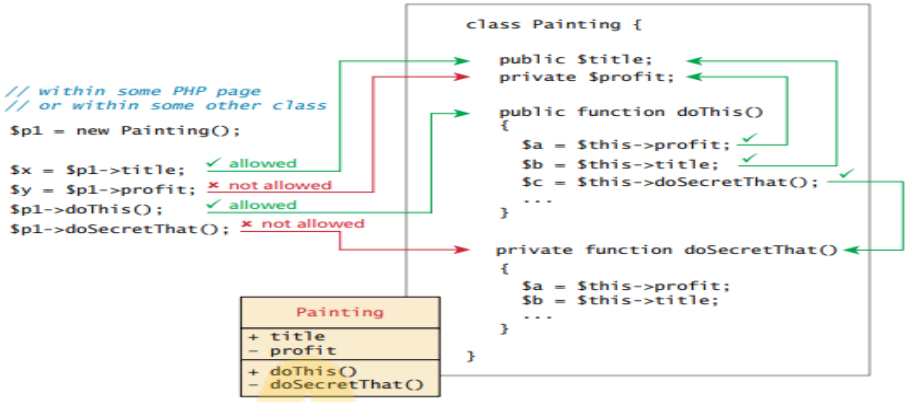

Figure 2.6: Visibility of class members

- **Public**
  - The public keyword means that the property or method is accessible to any code that has a reference to the object.
- **Private**
  - 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 we have a reference to it as shown in Figure 2.6.
- **Protected**
  - Protected data members and methods are only accessible by the classes of the same package and the subclasses present in any package.
- In UML, the "+" symbol is used to denote public properties and methods, the "–" symbol for private ones, and the "#" symbol for protected ones.

Static Members

- A static member is a property or method that all instances of a class share and there is only one value for a class's static property.
- To illustrate how a static member is shared between instances of a class, we will add the static property artistCount to our Artist class, and use it to keep a count of how many Artist objects are currently instantiated.

- This variable is declared static by including the static keyword in the declaration as shown in listing 2.5

```
class Artist {
    public static $artistCount = 0;
    public   $firstName;
    public   $lastName;
    public   $birthDate;
    public   $birthCity;
    public   $deathDate;

    function __construct($firstName, $lastName, $city, $birth,
                         $death=null) {
       $this->firstName = $firstName;
       $this->lastName = $lastName;
       $this->birthCity = $city;
       $this->birthDate = $birth;
       $this->deathDate = $death;
       self::$artistCount++;
    }
 }
```

Listing 2.5: Class definition modified with static members

- Notice that we do not reference a static property using the $this-> syntax, but rather it has its own self:: syntax.
- The rationale behind this change is to force the programmer to understand that the variable is static and not associated with an instance ($this).
- This static variable can also be accessed without any instance of an Artist object by using the class name, that is, via Artist::$artistCount.
- Static methods are similar to static properties in that they are globally accessible (if public) and are not associated with particular objects.
- It should be noted that static methods cannot access instance members.
- Static methods are called using the same double colon syntax as static properties.

Class Constants

- To add a property to a class that is constant as same as static members but with a const keyword.

```
const EARLIEST_DATE = 'January 1, 1200';
```

- However, constant values can be stored more efficiently as class constants so long as they are not calculated or updated.

- 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.

2.3 Object-Oriented Design

- With the basic understanding of how to define and use classes and objects, we can start to get the benefits of software engineering patterns, which encourage understandable and less error-prone code.
- The object-oriented design of software offers many benefits in terms of modularity, testability, and reusability.

Data Encapsulation

- The most important advantage to object-oriented design is the possibility of **encapsulation**, which generally refers to restricting access to an object's internal components.
- Another way of understanding encapsulation is: it is the hiding of an object's implementation details.
- A properly encapsulated class will define an interface to the world in the form of its public methods, and leave its data, that is, its properties, hidden (that is, private).
- If a properly encapsulated class makes its properties private, then the typical approach is to write methods for accessing and modifying properties rather than allowing them to be accessed directly.
- These methods are commonly called **getters and setters** (or accessors and mutators).
- A **getter** to return a variable's value is often very straightforward and should not modify the property.
- It is normally called without parameters, and returns the property from within the class.
- **Setter** methods modify properties, and allow extra logic to be added to prevent properties from being set to strange values.

- **Ex:**

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

getter function : get()

```
public function setBirthDate($birthdate){
    // set variable only if passed a valid date string
    $date = date_create($birthdate);

    if ( ! $date ) {
        $this->birthDate = $this->getEarliestAllowedDate();
    }
    else {
        // if very early date then change it to
        // the earliest allowed date
        if ( $date < $this->getEarliestAllowedDate() ) {
            $date = $this->getEarliestAllowedDate();
        }
        $this->birthDate = $date;
    }
}
```

setter function: set()

- The key advantage of using getters and setters are:
  - The class can handle the responsibility of ensuring its own data validation.
  - And since the setter functions are performing validation, the constructor for the class should use the setter functions to set the values.

Inheritance

- Along with encapsulation, **inheritance** is one of the three key concepts in object oriented design and programming.
- Inheritance enables us to create new PHP classes that reuse, extend, and modify the behavior that is defined in another PHP class.
- PHP only allows you to inherit from one class at a time.
- A class that is inheriting from another class is said to be a **subclass** or a **derived class**.
- The class that is being inherited from is typically called a **superclass** or a **base class**.
- When a class inherits from another class, it inherits all of its public and protected methods and properties. Figure 2.7 illustrates how inheritance is shown in a UML class diagram.

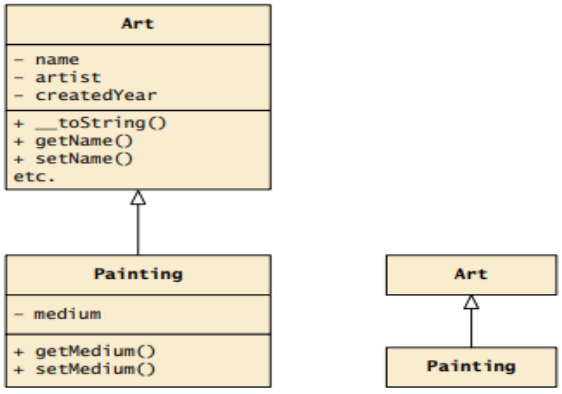

Figure 2.7: UML class diagram with inheritance

- Just as in Java, a PHP class is defined as a subclass by using the extends keyword.

```
class Painting extends Art { . . . }
```

Referencing Base Class Members

- A subclass inherits the public and protected members of the base class. Thus in the following code based on Figure 2.7, both of the references will work because it is as if the base class public members are defined within the subclass.

```
$p = new Painting();
. . .
// these references are ok
echo $p->getName();      // defined in base class
echo $p->getMedium();    // defined in subclass
```

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.
- To access a public or protected method or property defined within a base class from within a subclass, we do so by prefixing the member name with parent::. So to access the parent's __toString() method we would simply use parent::__toString().

Parent Constructors

- If we want to invoke a parent constructor in the derived class's constructor, we can 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 our constructor.

Polymorphism

- Polymorphism is the third key object-oriented concept (along with encapsulation and inheritance).
- Polymorphism is the notion that an object can in fact be multiple things at the same time.
- Let us begin with an instance of a Painting object named $guernica created as follows:

```
$guernica = new Painting("1937",$picasso,"Guernica","Oil on canvas");
```

- The variable $guernica is both a Painting object and an Art object due to its inheritance.
- The advantage of polymorphism is that we can manage a list of Art objects, and call the same overridden method on each.

Object Interfaces

- An object interface is a way of defining a formal list of methods that a class must implement without specifying their implementation.
- Interfaces provide a mechanism for defining what a class can do without specifying how it does it, which is often a very useful design technique.
- Interfaces are defined using the interface keyword, and look similar to standard PHP classes, except an interface contains no properties and its methods do not have method bodies defined.
- Ex:

```
interface Viewable {
    public function getSize();
    public function getPNG();
}
```

- 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 Painting extends Art implements Viewable { ... }
```

- This means then that the class Painting must provide implementations (i.e., normal method bodies) for the getSize() and getPNG() methods.

Chapter 3: Error Handling and Validation

1. What are Errors and Exceptions?
2. PHP Error Reporting
3. PHP Error and Exception Handling

3.1 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, RSS feeds, 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

- Not every problem is unexpected or catastrophic. One might say that 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. Perhaps the most common example of this type would be an error as a result of user inputs, for instance, entering letters when numbers were expected.
- 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.
- The final type of error is **fatal errors**, which are serious in that the execution of the page will terminate unless handled in some way. These should truly be exceptional and unexpected, such as a required input file being missing or a database table or field disappearing.

Exceptions

- Developers sometimes treat the words "error" and "exception" as synonyms. In the context of PHP, they do have different meanings.
- 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.

3.2 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.**
- There are three main error reporting flags:
  - error_reporting
  - display_errors
  - log_errors
- The meaning of each of these is important and should be learned by PHP developers.

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 lists some of the most common values. It is worth noting that in some PHP environments, the default setting is zero, that is, no reporting.

The display_errors Setting

- The display_error setting specifies whether error messages should or should not bedisplayed in the browser.2 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
```

3.3 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.
- In such a case we needed to test for and deal with errors after each operation that might generate an error state.
- While this approach might seem more straightforward, it does require the programmer to know ahead of time what code is going to generate an error condition.
- 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.
- Even with explicit testing for error conditions, there will still be situations when an unforeseen error occurs. In such a case, unless a custom error handler has been defined, PHP will terminate the execution of the application.

```php
$connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME);

$error = mysqli_connect_error();
if ($error != null) {
   // handle the error
   ...
}
```

Listing 3.1: Procedural approach to error handling

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".
- Like other object-oriented programming languages, PHP uses the try .. .catch programming construct to programmatically deal with exceptions at runtime.
- The Exception 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.

```php
// 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
}
```

Listing 3.2: Example of try . . . catch block

- 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.

Custom Error and Exception Handlers

- When a web application is in development, one can generally be content with displaying and/or logging error messages and then terminating the script.
- But for production applications, we will likely want to handle significant errors in a better way.

- It is possible to define our own handler for uncaught errors and exceptions; the mechanism for doing so varies depending upon whether you are using the procedural or object-oriented mechanism for responding to errors.
- If using the procedural approach (i.e., *not* using try . . . catch), we can define a custom *error*-handling function and then register it with the set_error_handler() function.
- If we are using the object-oriented exception approach with try . . . catch blocks, we can define a custom *exception*-handling function and then register it with the set_exception_handler() function.

```
function my_exception_handler($exception) {

  // put together a detailed exception message
  $msg = "<p>Exception Number " . $exception->getCode();
  $msg .= $exception->getMessage() . " occurred on line ";
  $msg .= "<strong>" . $exception->getLine() . "</strong>";
  $msg .= " and in the file: ";
  $msg .= "<strong>" . $exception->getFile() . "</strong> </p>";

  // email error message to someone who cares about such things
  error_log($msg, 1, 'support@domain.com',
            'From: reporting@domain.com');

  // if exception serious then stop execution and tell maintenance fib
  if ($exception->getCode() !== E_NOTICE) {
     die("Sorry the system is down for maintenance. Please try
          again soon");
  }
}
```

Listing 3.3: Custom exception handler