

PHP

Arrays and Superglobals

Chapter 9

Objectives

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

Arrays

Section 1 of 5

Arrays

Background

An array is a data structure that

- Collects a number of related elements together in a single variable.
- Allows the set to be iterated
- Allows access of any element

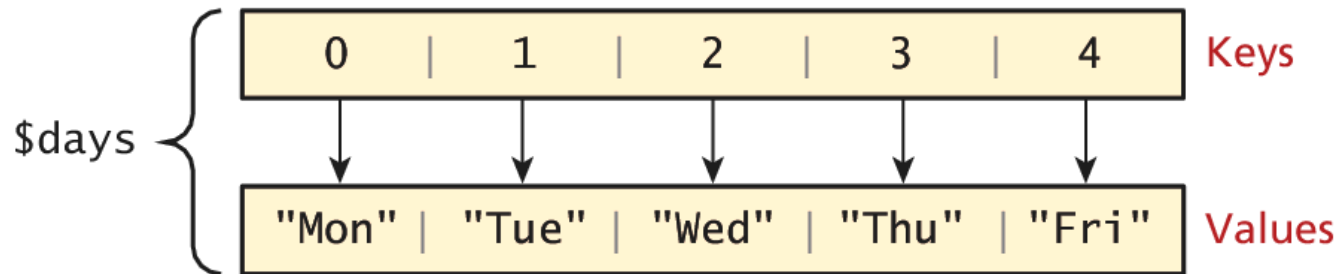
Since PHP implements an array as a dynamic structure:

- Add to the array
- Remove from the array

Arrays

Key Value

In PHP an array is actually an **ordered map**, which associates each value in the array with a key.



Arrays

Keys

Array keys are the means by which you refer to a single element in the array.

In most programming languages array keys are limited to integers, start at 0, and go up by 1.

In PHP, array keys *must* be either integers or strings and need not be sequential.

- Don't mix key types i.e. "1" vs 1
- If you don't explicitly define them they are 0,1,...

Arrays

Values

Array values, unlike keys, are not restricted to integers and strings.

They can be any object, type, or primitive supported in PHP.

You can even have objects of your own types, so long as the keys in the array are integers and strings.

Arrays

Defining an array

The following declares an empty array named days:

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

You can also initialize 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
```


Arrays

Defining an array

You can also declare each subsequent element in the array individually:

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

```
$days[0] = "Mon"; //set 0th key's value to "Mon"
```

```
$days[1] = "Tue";
```

// also alternate approach

```
$daysB = array();
```

```
$daysB[] = "Mon"; //set the next sequential value to "Mon"
```

```
$daysB[] = "Tue";
```

Arrays

Access values

To access values in an array you refer to their key using the square bracket notation.

```
echo "Value at index 1 is ". $days[1];
```

Keys and Values

In PHP, you are also able to explicitly define the keys in addition to the values.

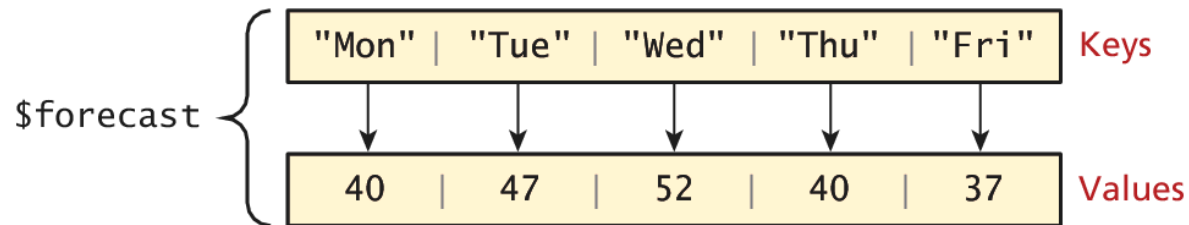
This allows you to use keys other than the classic 0, 1, 2, . . . , n to define the indexes of an array.

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

Super Explicit

Array declaration with string keys, integer values

```
$forecast = array(key"Mon" => value40, "Tue" => 47, "Wed" => 52, "Thu" => 40, "Fri" => 37);
```



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

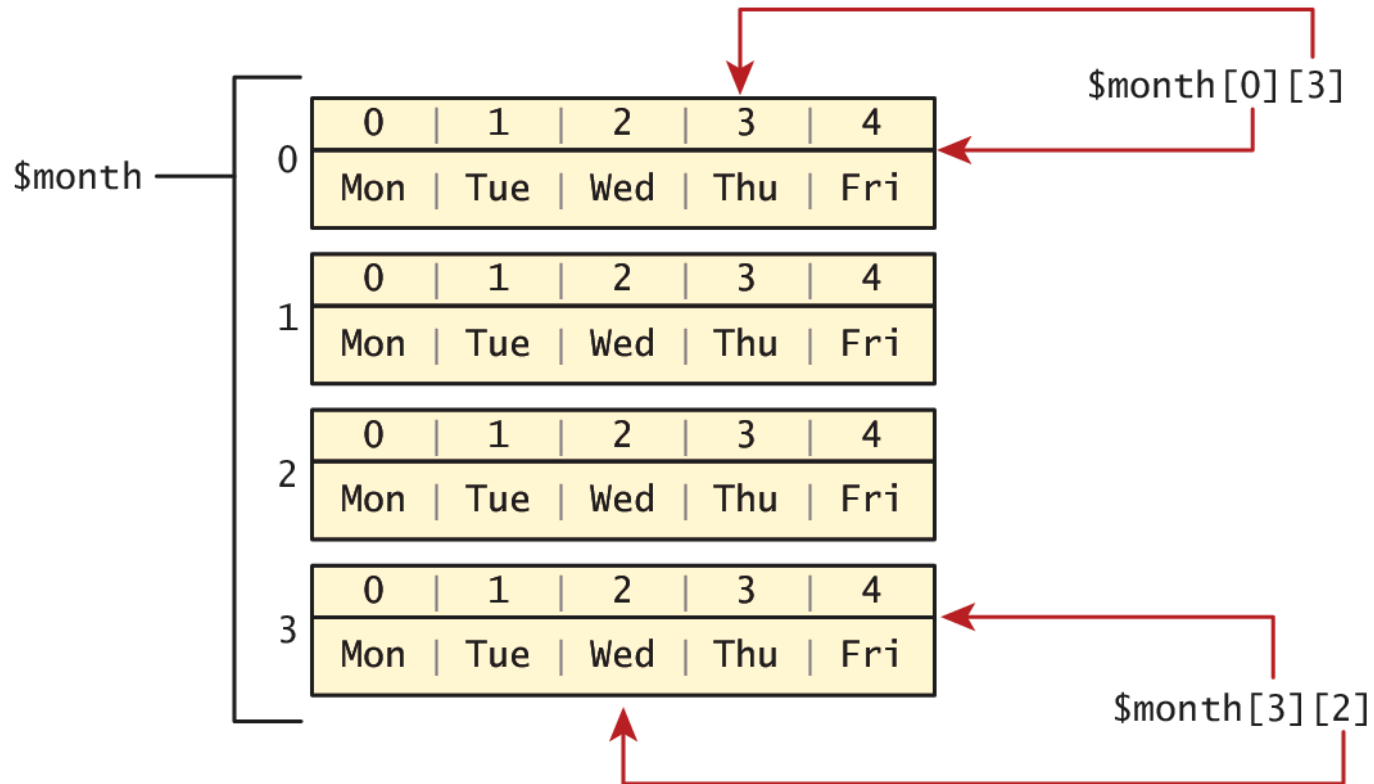
Multidimensional Arrays

Creation

```
$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
```

Multidimensional Arrays

Access



Multidimensional Arrays

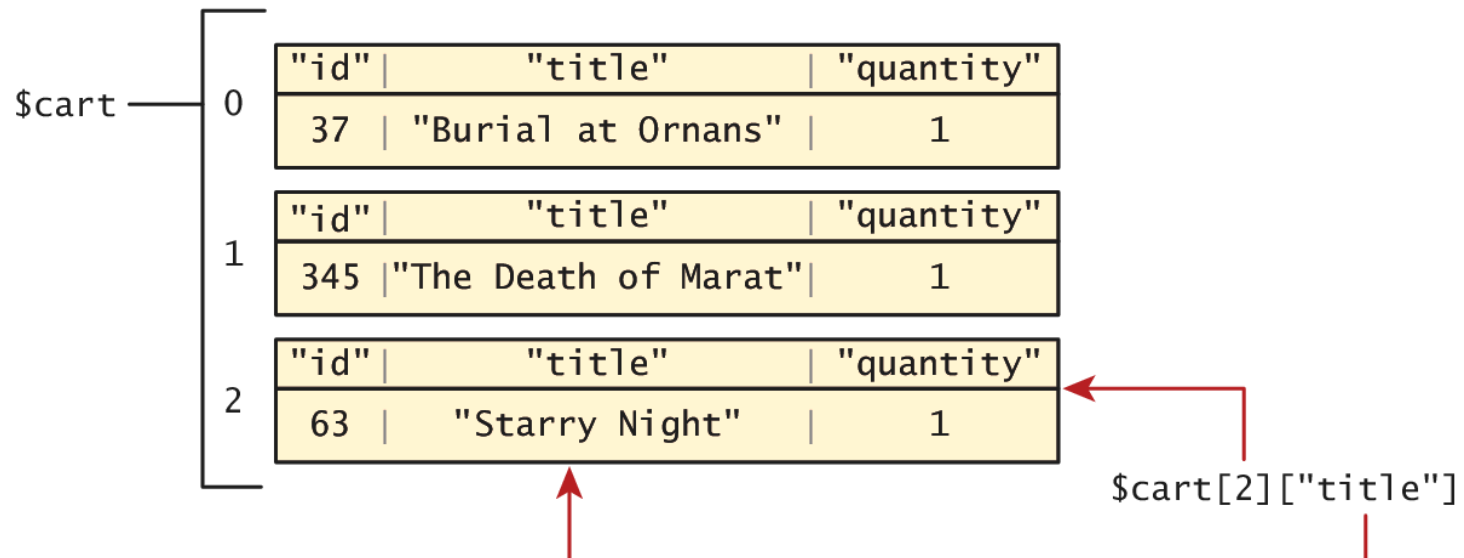
Another example

```
$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);
```



Iterating through an array

```
// 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 9.2 Iterating through an array using while, do while, and for loops

Iterating through an array

Foreach loop is pretty nice

The challenge of using the classic loop structures is that when you have nonsequential integer keys (i.e., an associative array), you can't write a simple loop that uses the `$i++` construct. To address the dynamic nature of such arrays, you have to use iterators to move through such an array.

```
// 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 9.3 Iterating through an associative array using a foreach loop

Adding to an array

To an array

An element can be added to an array simply by using a key/index that hasn't been used

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

A new element can be added to the end of any array

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

Adding to an array

And quickly printing

PHP is more than happy to let you “skip” an index

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

```
$days[7] = "Sat";
```

```
print_r($days);
```

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

If we try referencing \$days[6], it will return a **NULL** value

Deleting from an array

You can explicitly delete array elements 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 )  
  
$days = array_values($days);  
print_r($days); // outputs: Array ( [0] => Mon [1] => Tue [2] => Fri )
```

LISTING 9.4 Deleting elements

Deleting from an array

You can explicitly delete array elements using the `unset()` function.

`array_values()` reindexes the array numerically

```
$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 9.4 Deleting elements

Checking for a value

Since array keys need not be sequential, and need not be integers, you may run into a scenario where you want to check if a value has been set for a particular key.

To check if a value exists for a key, you can therefore use the `isset()` function, which returns true if a value has been set, and false otherwise

```
$oddKeys = array (1 => "hello", 3 => "world", 5 => "!");  
if (isset($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 9.5 Illustrating nonsequential keys and usage of `isset()`

Array Sorting

Sort it out

There are many built-in sort functions, which sort by key or by value. To sort the `$days` array by its values you 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)
```

A better sort, one that would have kept keys and values associated together, is:

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

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

More array operations

Too many to go over in depth here...

- `array_keys($someArray)`
- `array_values($someArray)`
- `array_rand($someArray, $num=1)`
- `array_reverse($someArray)`
- `array_walk($someArray, $callback, optionalParam)`
- `in_array($needle, $haystack)`
- `shuffle($someArray)`
- ...

\$_GET and \$_POST

Superglobal arrays

Section 2 of 5

Superglobal Arrays

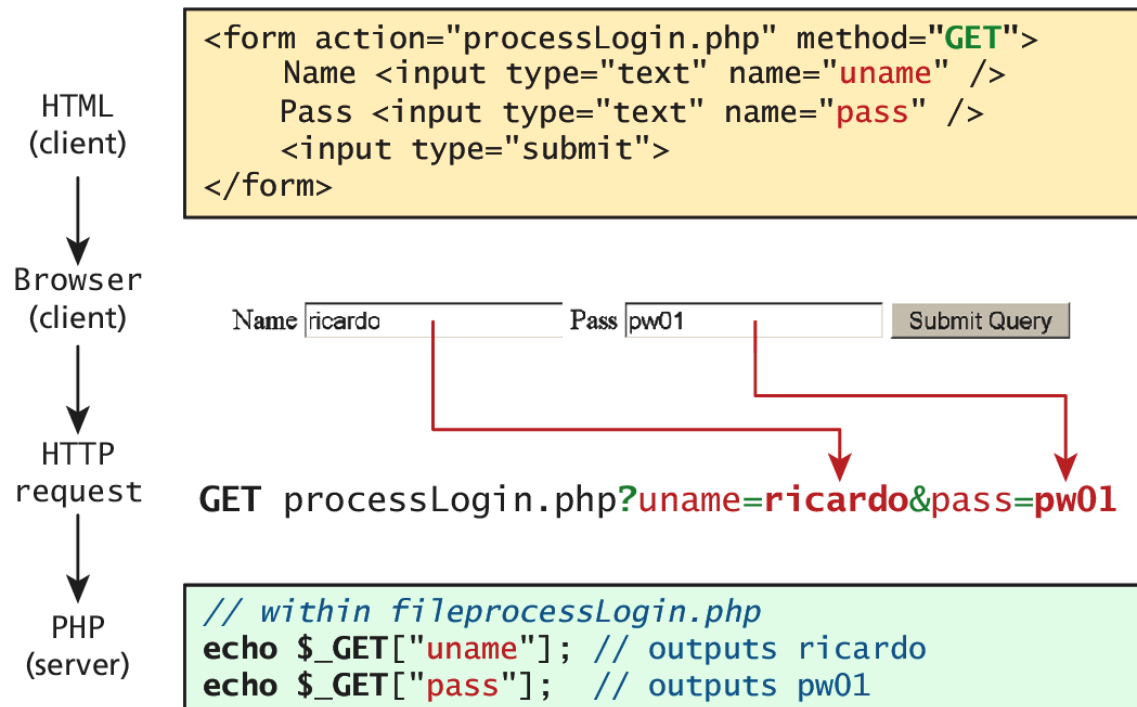
PHP uses special predefined associative arrays called **superglobal variables** that allow the programmer to easily access HTTP headers, query string parameters, and other commonly needed information.

They are called superglobal because they are always in scope, and always defined.

\$_GET and \$_POST

Sound familiar?

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.



`$_GET` and `$_POST`

Sound familiar?

- Get requests parse query strings into the `$_GET` array
- Post requests are parsed into the `$_POST` array

This mechanism greatly simplifies accessing the data posted by the user, since you need not parse the query string or the POST request headers!

Determine if any data sent

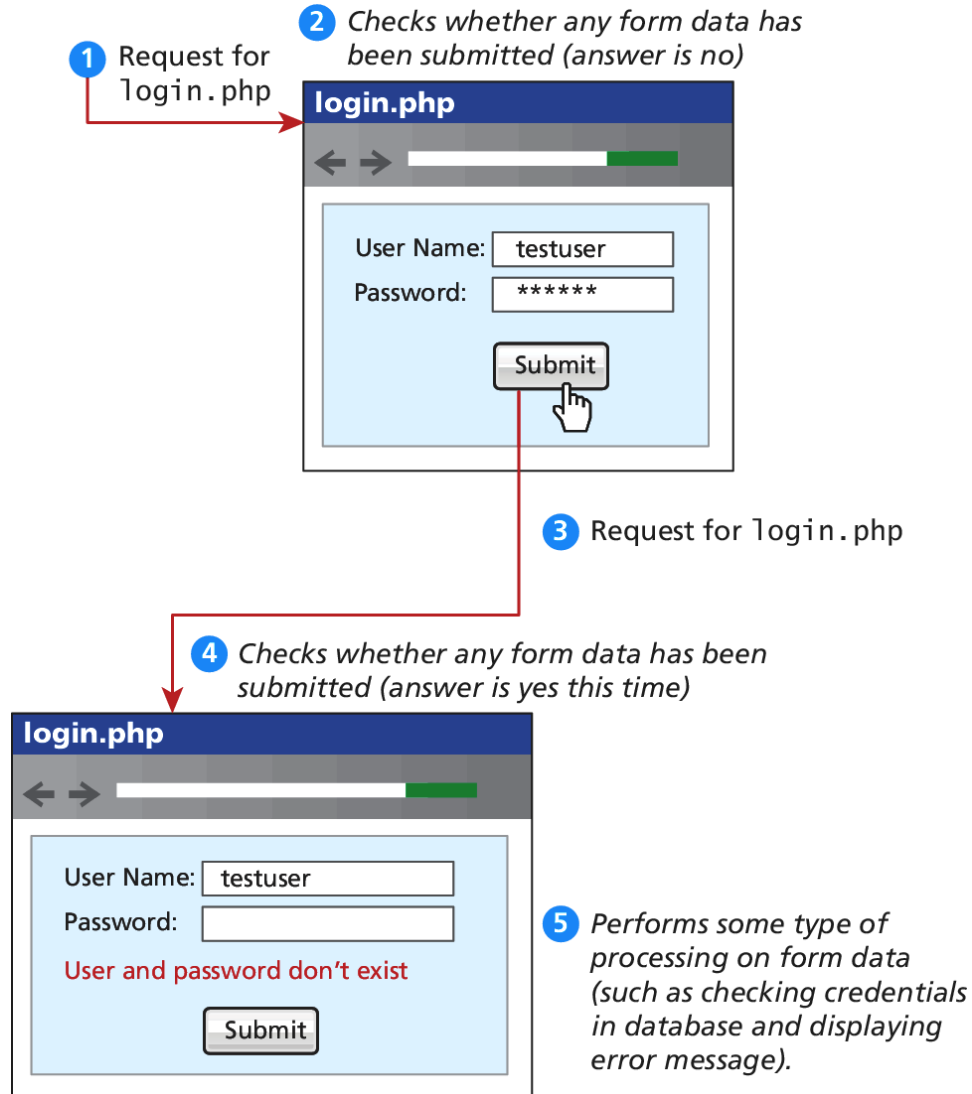
Isset()

```
<!DOCTYPE html>
<html>
<body>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if ( isset($_POST["uname"]) && isset($_POST["pass"])) ) {
        // handle the posted data.
        echo "handling user login now ...";
        echo "... here we could redirect or authenticate ";
        echo " and hide login form or something else";
    }
}

?>
<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>
```

LISTING 9.6 Using isset() to check query string data

Determine if any data sent



Determine if any data sent

Isset()

```
<!DOCTYPE html>
<html>
<body>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if ( isset($_POST["uname"]) && isset($_POST["pass"])) ) {
        // handle the posted data.
        echo "handling user login now ...";
        echo "... here we could redirect or authenticate ";
        echo " and hide login form or something else";
    }
}

?>
<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>
```

LISTING 9.6 Using isset() to check query string data

Accessing Form Array Data

Sometimes in HTML forms you might have multiple values associated with a single name;

```
<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 9.7 HTML that enables multiple values for one name

Accessing Form Array Data

HTML tweaks for arrays of data

Unfortunately, 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 name attribute for each checkbox from `day` to `day[]`.

Monday ☐

Tuesday ☐

Accessing Form Array Data

Meanwhile on the server

After making this change in the HTML, the corresponding variable `$_GET['day']` will now have a value that is of type array.

```
<?php

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

?>
```

LISTING 9.8 PHP code to display an array of checkbox variables

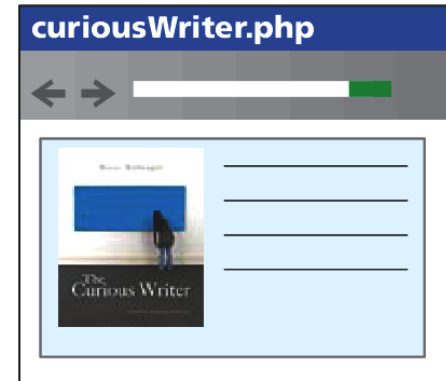
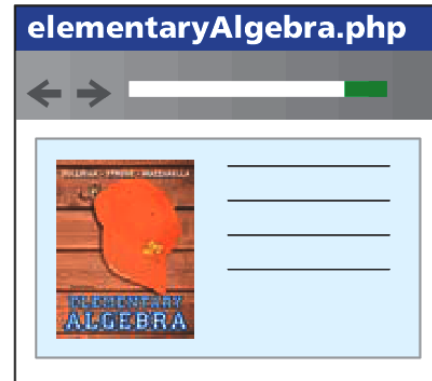
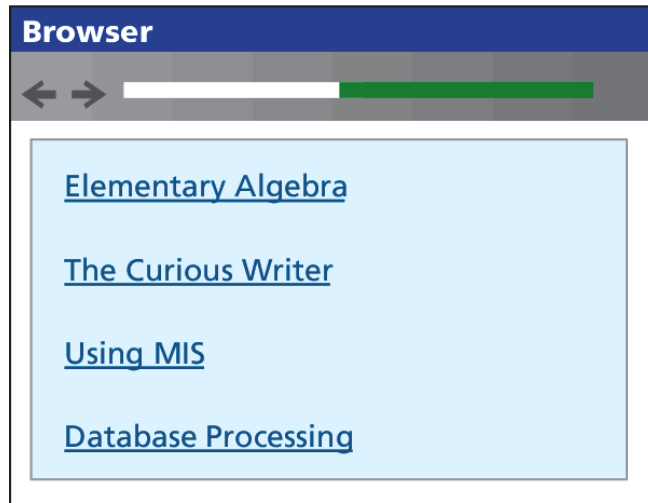
Using Query String in Links

Design idea

Imagine a web page in which we are displaying a list of book links. One approach would be to have a separate page for each book.

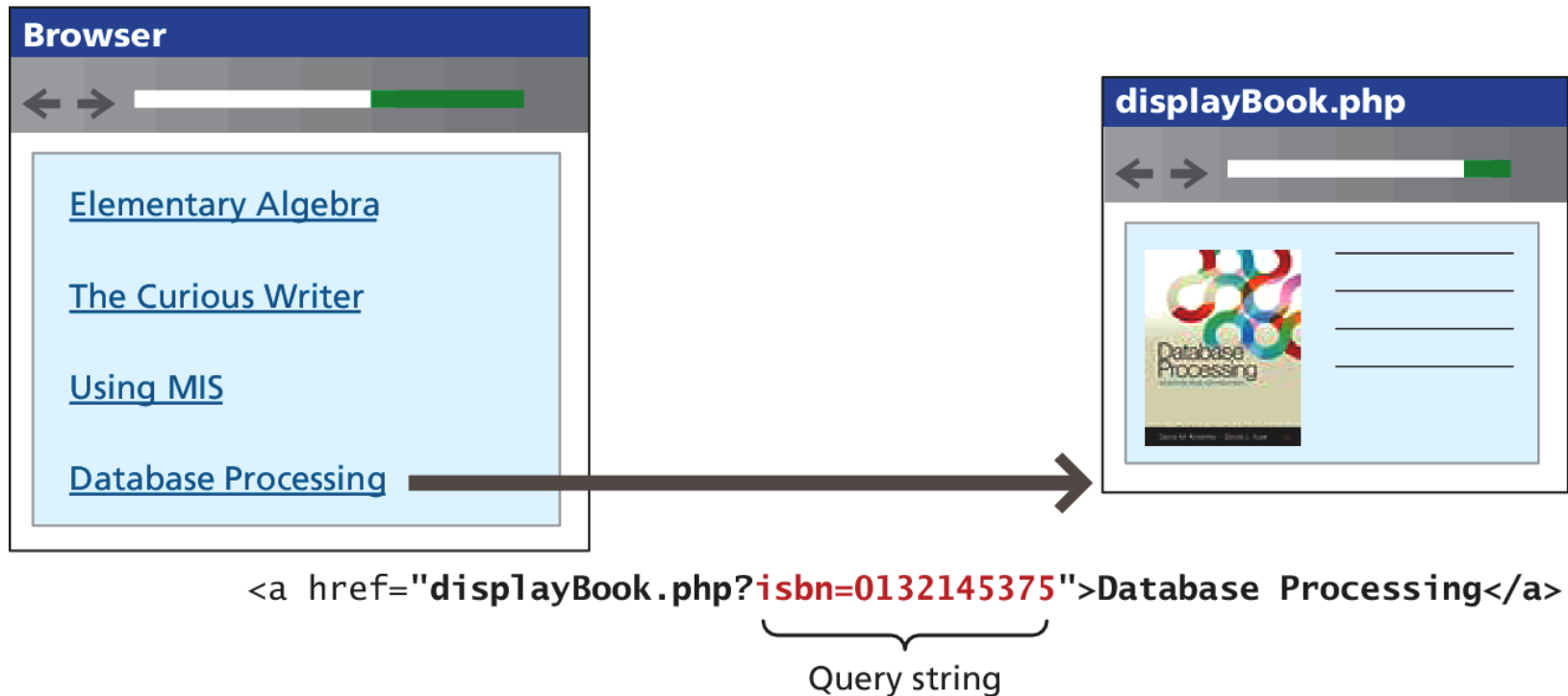
Using Query Strings in links

Not a great setup



Using Query Strings in links

Use the query string to reduce code duplication



Sanitizing Query Strings

Just because you are expecting a proper query string, doesn't mean that you are going to get a properly constructed query string.

- **distrust all user input**

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

Learn more about this in Chapter 11/12.

Sanitation

Don't forget trim()

```
// 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 9.9 Simple sanitization of query string values

`$_SERVER` array

Section 3 of 5

\$_SERVER

The \$_SERVER associative array contains

- HTTP request headers (send by client)
- configuration options for PHP

To use the \$_SERVER array, you simply refer to the relevant case-sensitive keyname:

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

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

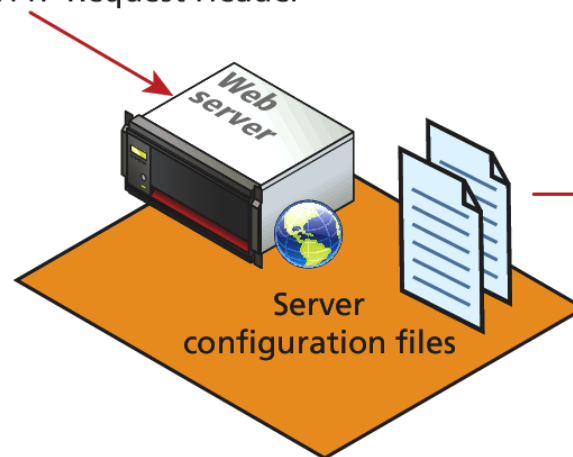
```
echo $_SERVER["REMOTE_ADDR"] . "<br/>";
```

\$_SERVER

\$_SERVER['REQUEST_METHOD'] \$_SERVER['SERVER_PROTOCOL']

```
$_SERVER['REQUEST_TIME'] — POST /page.php http/1.1
$_SERVER['HTTP_USER_AGENT'] — Date: Sun, 20 May 2012 23:59:59 GMT
$_SERVER['HTTP_CONNECTION'] — Host: www.mysite.com
                             User-Agent: Mozilla/4.0
                             Accept-Encoding: gzip
                             Connection: Keep-Alive
                             ...
                             <html> ...
```

HTTP Request Header



\$_SERVER['SERVER_NAME']
\$_SERVER['SERVER_ADDR']
\$_SERVER['SERVER_PORT']
...

SERVER INFORMATION KEYS

- `SERVER_NAME` contains the name of the site that was requested
- `SERVER_ADDR` tells us the IP of the server
- `DOCUMENT_ROOT` tells us the location from which you are currently running your script
- `SCRIPT_NAME` key that identifies the actual script being executed

Request Header Keys

- REQUEST_METHOD returns the request method that was used to access the page: that is, GET, HEAD, POST, PUT
- REMOTE_ADDR key returns the IP address of the requestor
- HTTP_USER_AGENT contains the operating system and browser that the client is using
- HTTP_REFERER contains the address of the page that referred us to this one (if any) through a link

Header Access Examples

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

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

LISTING 9.10 Accessing the user-agent string in the HTTP headers

```
$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 9.11 Using the HTTP_REFERER header to provide context-dependent output

Security

Headers can be forged

All headers can be forged!

- The HTTP_REFERER header can lie about where the referral came from
- The USER_AGENT can lie about the operating system and browser the client is using.

`$_FILES` array

Section 4 of 5

`$_FILES` Array

The **`$_FILES`** associative array contains items that have been uploaded in the current request.

A server script must process the upload file(s) in some way; the **`$_FILES`** array helps in this process.

the **`$_FILES`** array will contain a key=value pair for each file uploaded in the post

First a refresher on HTML forms for uploading files...

HTML Required for File Uploads

1. You must ensure that the HTML form uses the HTTP POST **method**, since transmitting a file through the URL is not possible.
2. You must add the **enctype="multipart/form-data"** attribute to the HTML form that is performing the upload so that the HTTP request can
3. You must include an input type of file in your form.

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

LISTING 9.12 HTML for a form that allows an upload

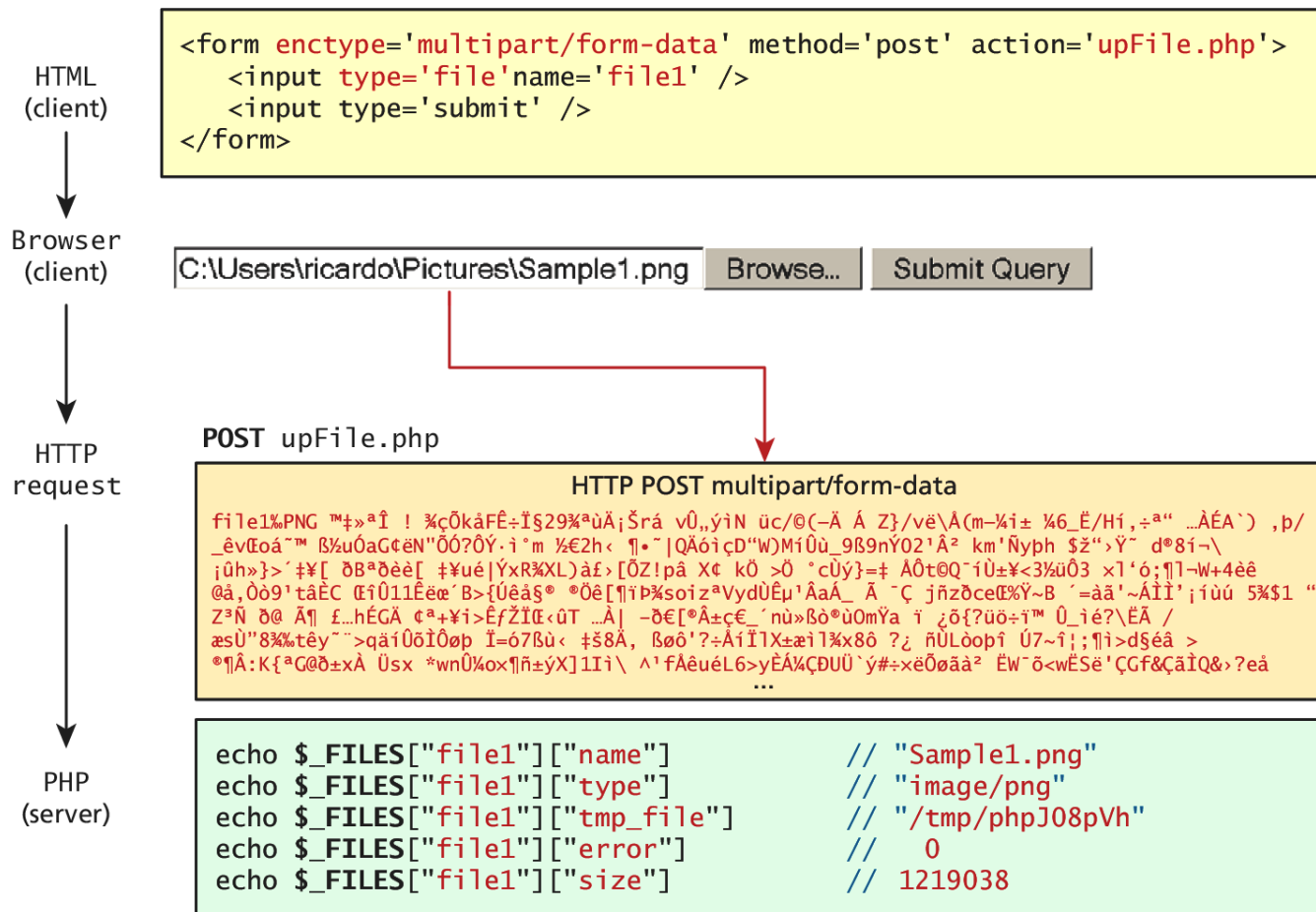
Handling File upload in PHP

The `$_FILES` 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 .

Handling File upload in PHP



Handling File upload in PHP

Keys. We still have to do something with this data

- **name** is a string containing the full file name used on the client machine, 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.
- **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.

Check for errors

For every uploaded file, there is an error value associated with it in the `$_FILES` array.

The value for a successful upload is **UPLOAD_ERR_OK**, and should be looked for before proceeding any further.

```
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 9.13 Checking each file uploaded for errors

File Size Restrictions

There are three main mechanisms for maintaining uploaded file size restrictions:

- HTML in the input form
- via JavaScript in the input form
- via PHP coding.

HTML in the input form

Add an hidden input field before any other input fields in your HTML form with a name of MAX_FILE_SIZE

The file uploading must be complete before an error message can be received.

```
<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 9.14 Limiting upload file size via HTML

via JavaScript

Allows a client side check to happen before any data transmitted.
(Easily overridden).

```
<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 9.15 Limiting upload file size via JavaScript

via PHP

The only one you HAVE to do.

The third mechanism for limiting the uploaded file size is to add a simple check on the server side (just in case JavaScript was turned off or the user modified the MAX_FILE_SIZE hidden field).

```
$max_file_size = 10000000;  
foreach($_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 9.16 Limiting upload file size via PHP

Limiting the type of File Upload

I won't allow .abc, .def now let me be

What if you wanted the user to upload an image and they uploaded a Microsoft Word document?

You might also want to limit the uploaded image to certain image types, such as jpg and png, while disallowing bmp and others.

- examine the file extension
- and the type field

Limiting the type of File Upload

Example code

```
$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 9.17 PHP code to look for valid mime types and file extensions

Moving the File

Finally!

You must move the temporary file to a permanent location on your server.

move_uploaded_file() takes in the temporary file location and the file's final destination.

```
$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 9.18 Using move_uploaded_file() function

READING/WRITING Files

Section 5 of 5

Reading/Writing

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. While not appropriate for large files, it does make processing of the file extremely easy.

Stream Access

C style

C style file access. More difficult, but more memory efficient.

The function `fopen()` takes a file location or URL and access mode as parameters. The returned value is a **stream resource**, which you can then read sequentially.

Use `fread()` or `fgets()` to read ahead in the file. `Fclose()` is invoked when you are done.

Writing done much the same with `fwrite()`.

Stream Access

Just show me the code

```
$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

In-Memory File Access

Easy as pie

- **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

In-Memory File Access

To read an entire file into a variable you simply use:

```
$fileAsString = file_get_contents(FILENAME);
```

To write the contents of a string \$writeme to a file:

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

In-Memory File Access

Consider a realistic example

imagine we have a comma-delimited text file that contains information about paintings, where each line in the file corresponds to a different painting:

01070,Picasso,The Actor,1904

01080,Picasso,Family of Saltimbanques,1905

02070,Matisse,The Red Madras Headdress,1907

05010,David,The Oath of the Horatii,1784

In-Memory File Access

Parsing our file

```
// read the file into memory; if there is an error then stop processing
$paintings = file($filename) or die('ERROR: Cannot find file');

// our data is comma-delimited
$delimiter = ',';

// loop through each line of the file
foreach ($paintings as $painting) {

    // returns an array of strings where each element in the array
    // corresponds to each substring between the delimiters
    $paintingFields = explode($delimiter, $painting);

    $id= $paintingFields[0];
    $artist = $paintingFields[1];
    $title = $paintingFields[2];
    $year = $paintingFields[3];

    // do something with this data
    . . .
}
```

LISTING 9.20 Processing a comma-delimited file

What You've Learned

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

PHP Classes and Objects

Chapter 10

Objectives

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

Object-Oriented Overview

Section 1 of 3

Overview

Object-Oriented Overview

PHP is a full-fledged object-oriented language with many of the syntactic constructs popularized in languages like Java and C++.

Earlier versions of PHP do not support all of these object-oriented features,

- PHP versions after 5.0 do

Terminology

Object-Oriented Terminology

The notion of programming with objects allows the developer to think about an item with particular **properties** (also called attributes or **data members**) and methods (functions).

The structure of these **objects** is 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 (largely) independently from the class once created.

Relationship between Class and Objects



Book class

Defines properties such as:
title, author, and number of pages



Objects (or instances of the Book class)

Each instance has its own title,
author, and number of pages
property values

UML

The Unified Modelling Language

The standard diagramming notation for object-oriented design is **UML (Unified Modeling Language)**.

Class diagrams and object diagrams, in particular, are useful to us when describing the properties, methods, and relationships between classes and objects.

For a complete definition of UML modeling syntax, look at the [Object Modeling Group's living specification](#)

UML Class diagram

By example

Every Artist has a

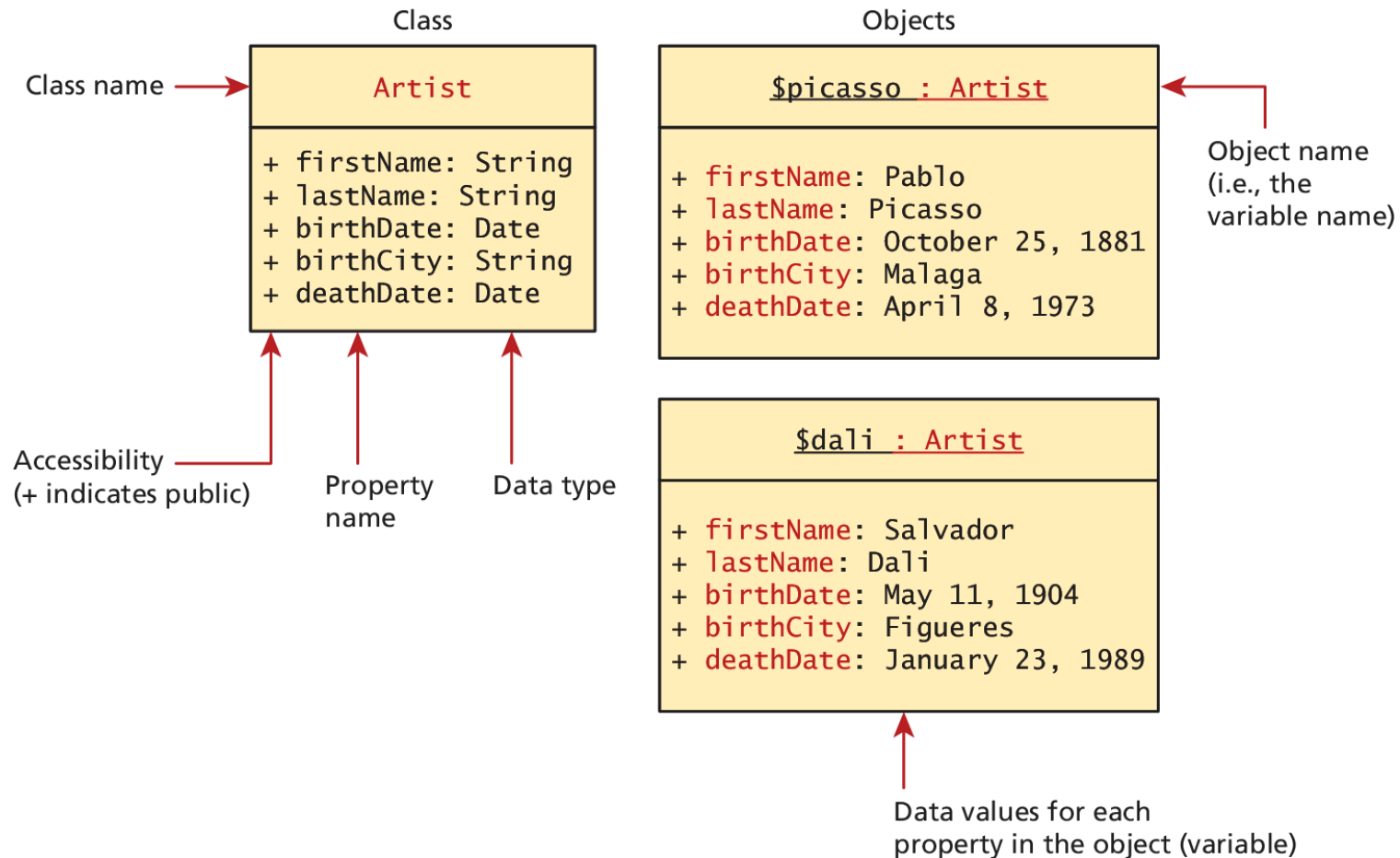
- first name,
- last name,
- birth date,
- birth city, and
- death date.

Using objects we can encapsulate those properties together into a class definition for an Artist.

UML articulates that design

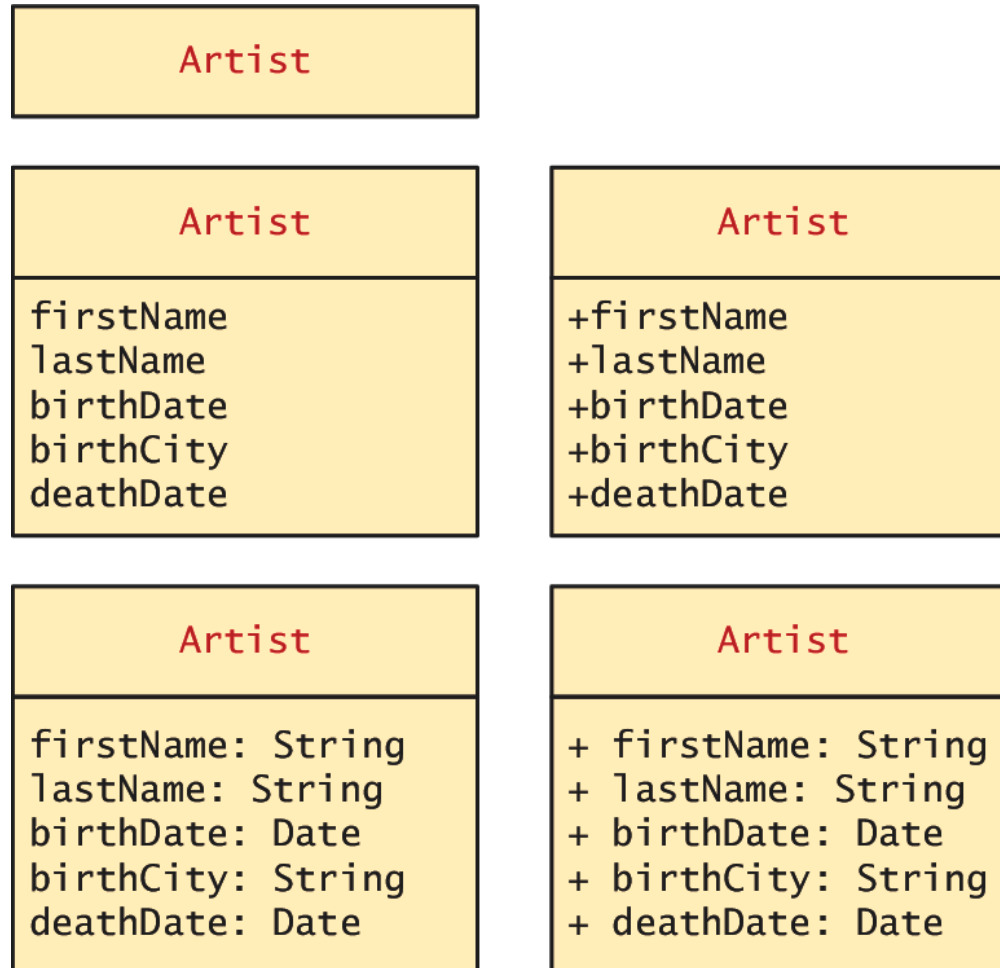
UML Class diagram

Class and a couple of objects



UML Class diagram

Different levels of detail



Server and Desktop Objects

Not the same

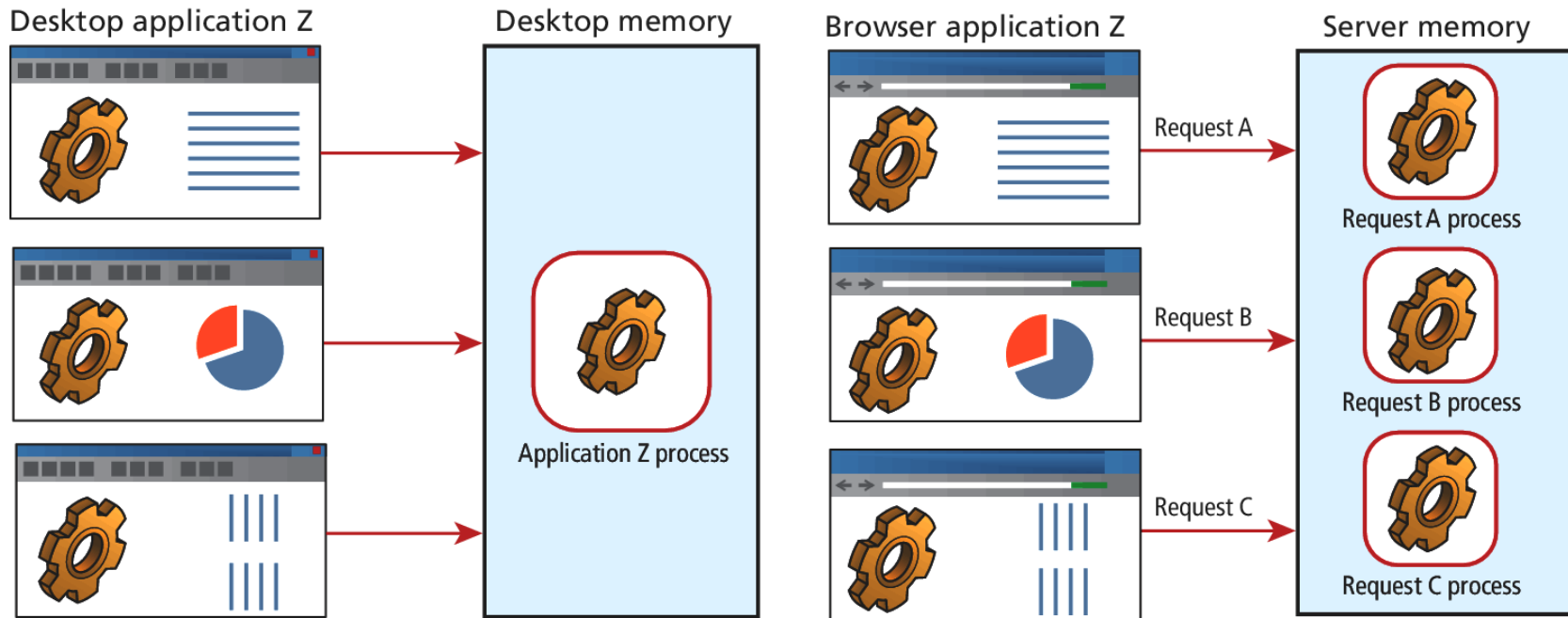
While desktop software can load an object into memory and make use of it for several user interactions, a PHP object is loaded into memory only for the life of that HTTP request.

We must use classes differently than in the desktop world, since the object must be recreated and loaded into memory

Unlike a desktop, there are potentially many thousands of users making requests at once, so not only are objects destroyed upon responding to each request, but memory must be shared between many simultaneous requests, each of which may load objects into memory or each request that requires it

Server and Desktop Objects

Not the same



Objects and Classes in PHP

Section 2 of 3

Defining Classes

In PHP

The PHP syntax for defining a class uses the class keyword followed by the class name and { } braces

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

LISTING 10.1 A simple Artist class

Instantiating Objects

In PHP

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.

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

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

Properties

The things in the objects

Once you have instances of an object, you can access and modify the properties of each one separately using the variable name and an arrow (->).

```
$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 10.2 Instantiating two Artist objects and setting one of those object's properties

Constructors

A Better way to build

Constructors let you specify parameters during instantiation to initialize the properties within a class right away.

In PHP, constructors are defined as functions (as you shall see, all methods use the function keyword) with the name **__construct()**.

Notice that in the constructor each parameter is assigned to an internal class variable using the `$this->` syntax. you **must** always use the `$this` syntax to reference all properties and methods associated with this particular instance of a class.

Constructors

An Example

```
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 10.3 A constructor added to the class definition

Constructors

Using the constructor

```
$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

Functions 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.

```
$picasso = new Artist( . . . )
```

```
echo $picasso->outputAsTable();
```

Methods

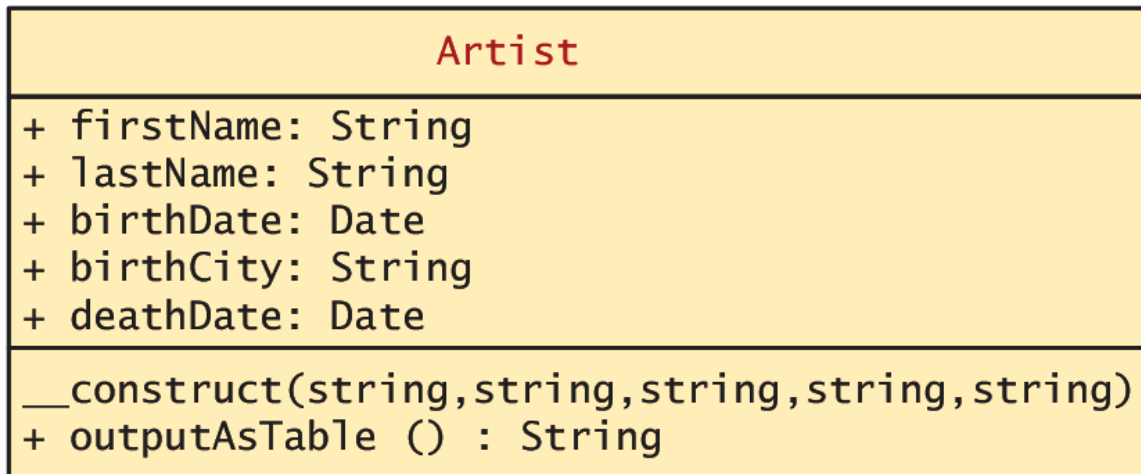
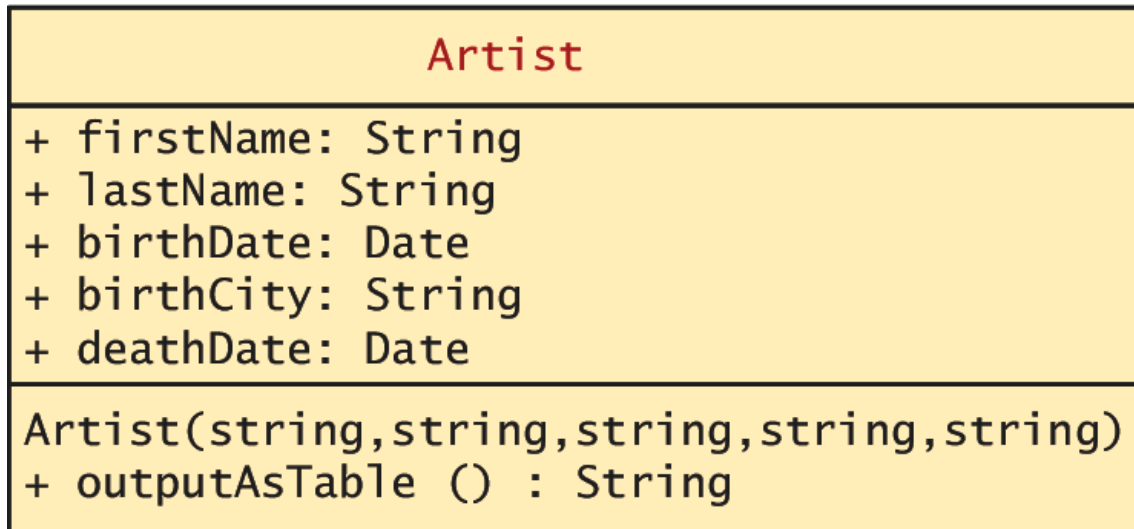
The example definition

```
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 10.4 Method definition

Methods

UML class diagrams adding the method



Visibility

Or accessibility

The **visibility** of a property or method determines the accessibility of a **class member** and can be set to:

- **Public** the property or method is accessible to any code that has a reference to the object
- **Private** sets a method or variable to only be accessible from within the class
- **Protected** is related to inheritance...

Visibility

Or accessibility

*// within some PHP page
// or within some other class*

```
$p1 = new Painting();
```

```
$x = $p1->title; ✓ allowed
```

```
$y = $p1->profit; ✗ not allowed
```

```
$p1->doThis(); ✓ allowed
```

```
$p1->doSecretThat(); ✗ not allowed
```

Painting
+ title - profit
+ doThis() - doSecretThat()

```
class Painting {
```

```
    public $title;
```

```
    private $profit;
```

```
    public function doThis()  
    {
```

```
        $a = $this->profit; ✓
```

```
        $b = $this->title; ✓
```

```
        $c = $this->doSecretThat(); ✓
```

```
        ...
```

```
    }
```

```
    private function doSecretThat()  
    {
```

```
        $a = $this->profit;
```

```
        $b = $this->title;
```

```
        ...
```

```
    }
```

```
}
```

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.

Static members use the `self::` syntax and are not associated with one object

They can be accessed without any instance of an Artist object by using the class name, that is, via **`Artist::$artistCount`**.

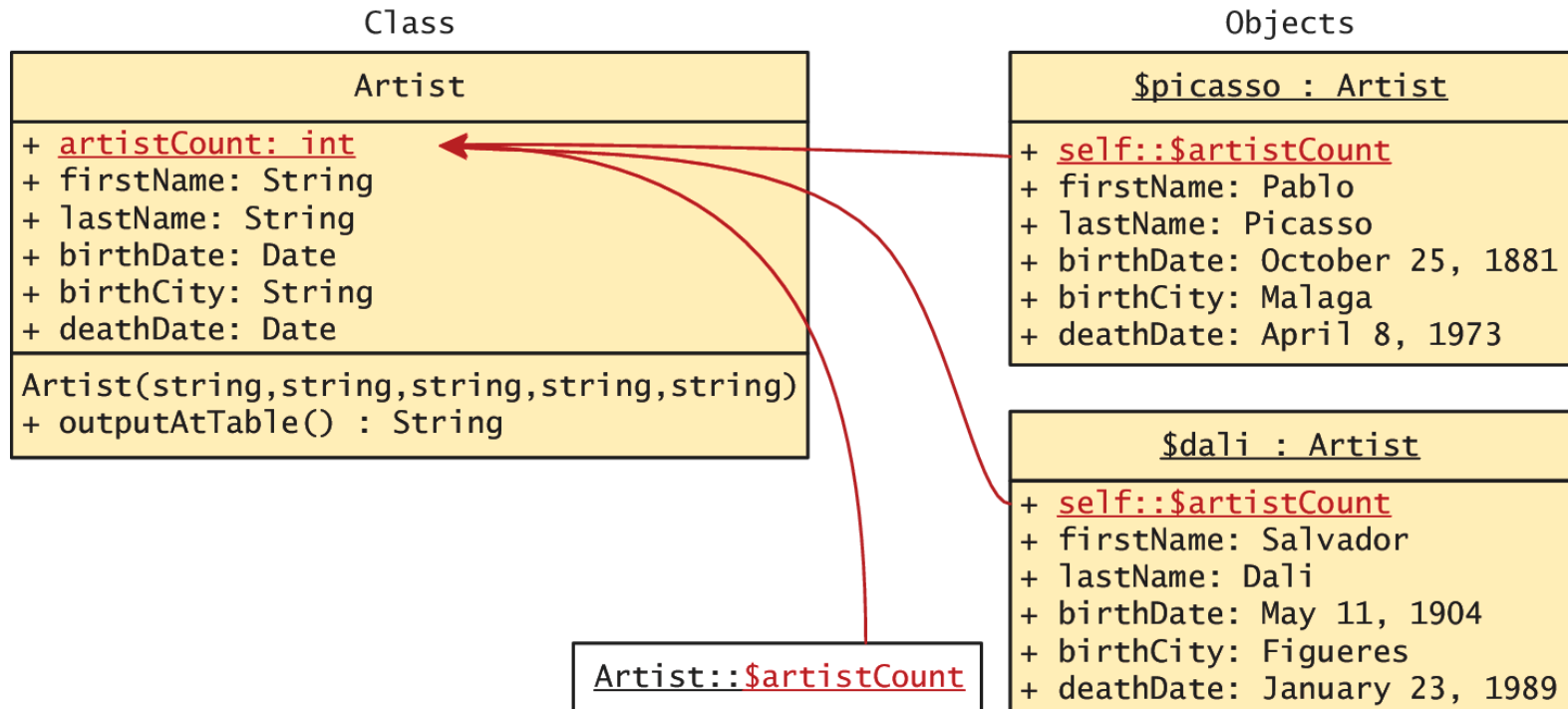
Static Members

```
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 10.5 Class definition modified with static members

Static Members

Uml again



Class constants

Never changes

Constant values can be stored more efficiently as class constants so long as they are not calculated or updated

They are added to a class using the **const** keyword.

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

Unlike all other variables, constants don't use the \$ symbol when declaring or using them.

Accessed both inside and outside the class using

- **self::EARLIEST_DATE** in the class and
- **classReference::EARLIEST_DATE** outside.

Object Oriented Design

Section 3 of 3

Data Encapsulation

What is it?

Perhaps 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).

Data Encapsulation

Getters and setters

If a properly encapsulated class makes its properties private, then how do you access them?

- **getters**
- **setters**

Data Encapsulation

Getters

A getter to return a variable's value is often very straightforward and should not modify the property.

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

Data Encapsulation

Setters

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

```
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;  
    }  
}
```

Data Encapsulation

UML

Artist
<ul style="list-style-type: none">- <u>artistCount</u>: int- firstName: String- lastName: String- birthDate: Date- deathDate: Date- birthCity: String
Artist(string,string,string,string,string) + outputAsTable () : String + getFirstName() : String + getLastName() : String + getBirthCity() : String + getDeathCity() : String + getBirthDate() : Date + getDeathDate() : Date + getEarliestAllowedDate() : Date + <u>getArtistCount()</u> : int + setLastName(\$lastname) : void + setFirstName(\$firstname) : void + setBirthCity(\$birthCity) : void + setBirthDate(\$deathdate) : void + setDeathDate(\$deathdate) : void

Artist
<ul style="list-style-type: none">- artistCount: Date- firstName: String- lastName: String- birthDate: Date- deathDate: Date- birthCity: String
Artist(string,string,string,string,string) + outputAsTable () : String + getEarliestAllowedDate() : Date

Data Encapsulation

Using an encapsulated class

```
<html>
<body>
<h2>Tester for Artist class</h2>

<?php
// first must include the class definition
include 'Artist.class.php';

// now create one instance of the Artist class
$picasso = new Artist("Pablo","Picasso","Malaga","Oct 25,1881",
    "Apr 8,1973");

// output some of its fields to test the getters
echo $picasso->getLastName() . ': ';
echo date_format($picasso->getBirthDate(),'d M Y') . ' to ';
echo date_format($picasso->getDeathDate(),'d M Y') . '<hr>';

// create another instance and test it
$dali = new Artist("Salvador","Dali","Figures","May 11,1904",
    "January 23,1989");

echo $dali->getLastName() . ': ';
echo date_format($dali->getBirthDate(),'d M Y') . ' to ';
echo date_format($dali->getDeathDate(),'d M Y') . '<hr>';

// test the output method
echo $picasso->outputAsTable();

// finally test the static method: notice its syntax
echo '<hr>';
echo 'Number of Instantiated artists: ' . Artist::getArtistCount();

?>
</body>
</html>
```

LISTING 10.7 Using the encapsulated class

Inheritance

Inheritance enables you 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**

A PHP class is defined as a subclass by using the ***extends*** keyword.

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

Referencing the Base Class

Inheritance enables you to create new PHP classes that reuse, extend, and modify the behavior that is defined in another PHP class.

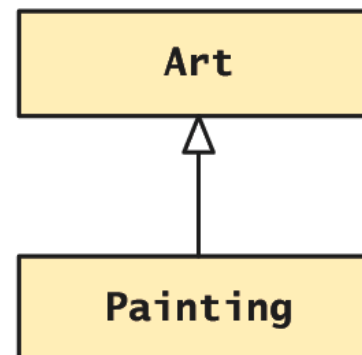
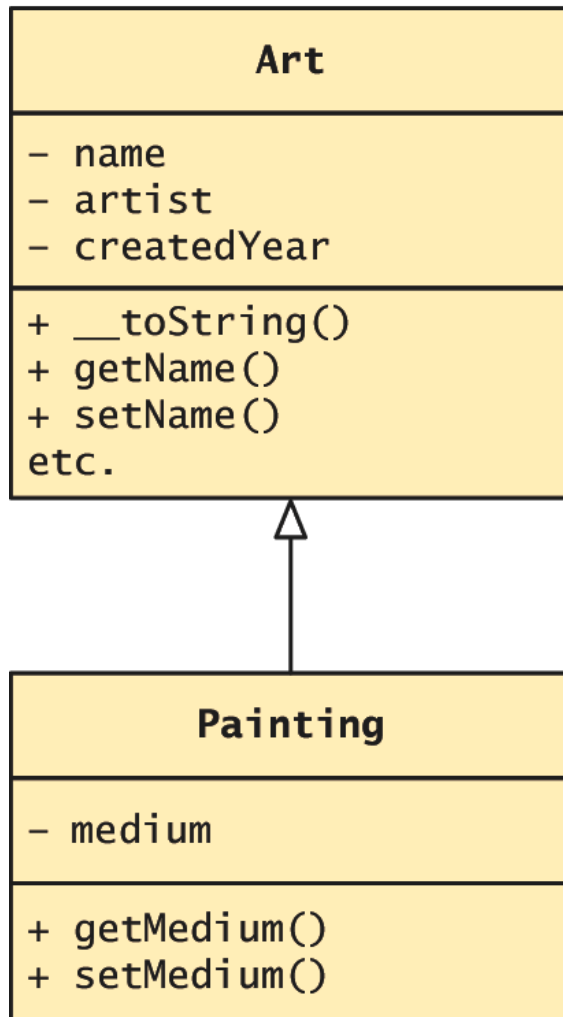
- PHP only allows you to inherit from one class at a time
- A class that is inheriting from another class is said to be a **subclass** or a **derived class**
- The class that is being inherited from is typically called a **superclass** or a **base class**

A PHP class is defined as a subclass by using the ***extends*** keyword.

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

Inheritance

There's UML for that too



Example usage

```
$p = new Painting();
```

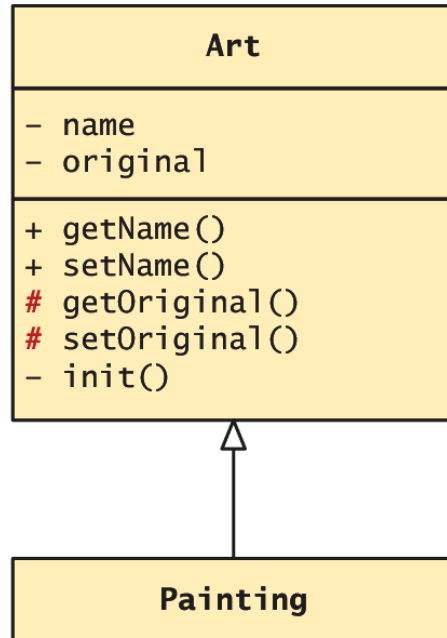
```
...
```

```
echo $p->getName(); // defined in base class
```

```
echo $p->getMedium(); // defined in subclass
```

Protected access modifier

Remember Protected?



```
class Painting extends Art {
    ...
    private function foo() {
        ...
        // these are allowed
        ✓ $w = parent::getName();
        ✓ $x = parent::getOriginal();

        // this is not allowed
        ✗ $y = parent::init();
    }
}
```

// in some page or other class

`$p = new Painting();`

`$a = new Art();`

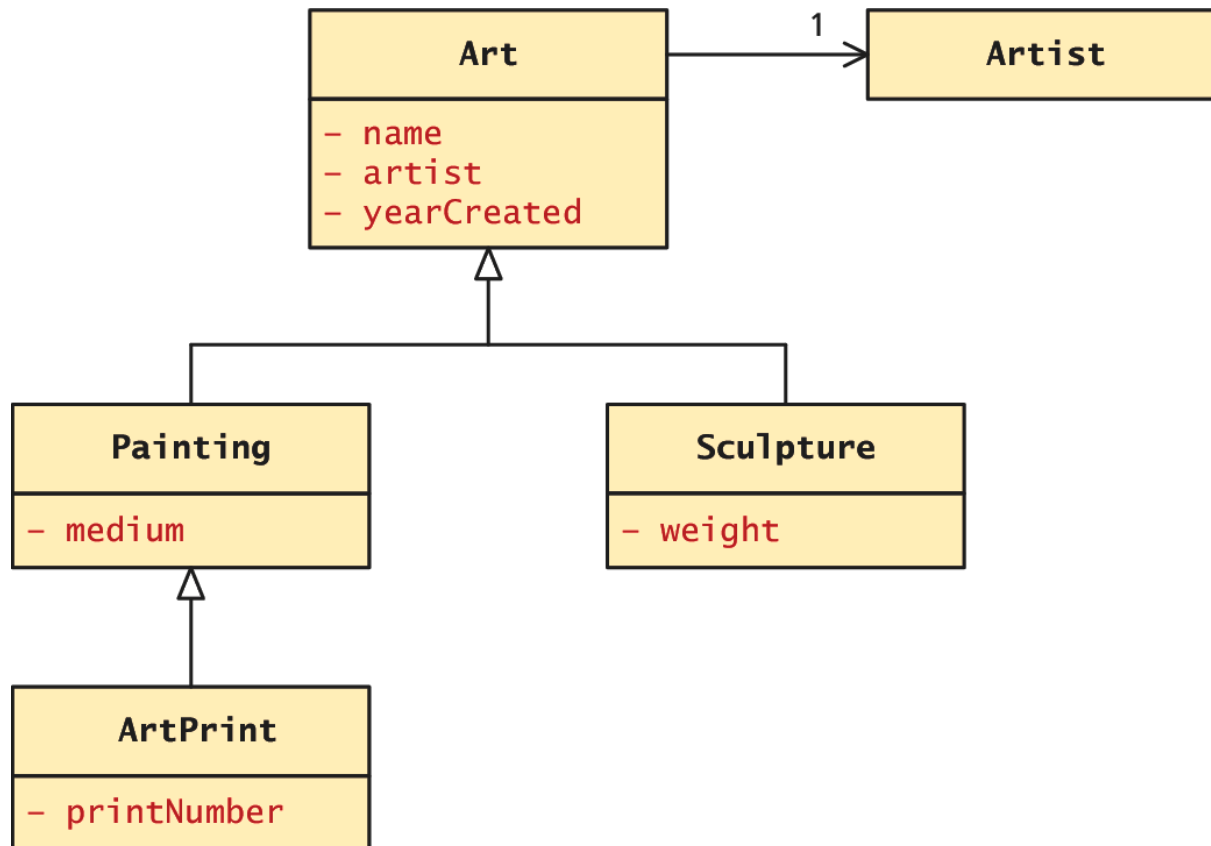
// neither of these references are allowed

✗ `$w = $p->getOriginal();`

✗ `$y = $a->getOriginal();`

A More Complex Example

Using inheritance



Extended example

All art has certain properties

/ The abstract class that
contains functionality required by
all types of Art */*

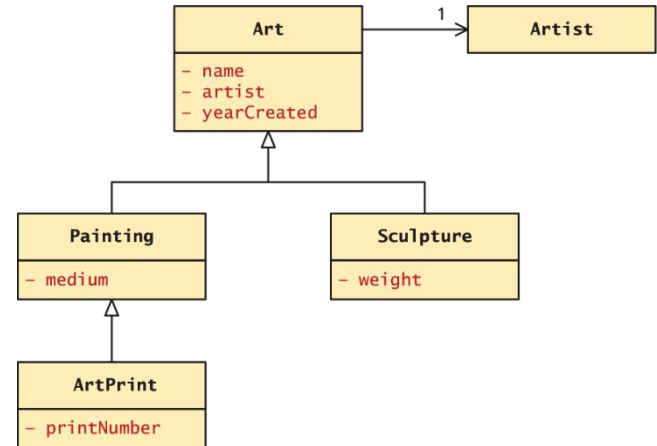
abstract class Art {

private \$name;

private \$artist;

private \$yearCreated;

//... constructor, getters, setters



Extended example

Painting require a “medium”

```
class Painting extends Art {
```

```
    private $medium;
```

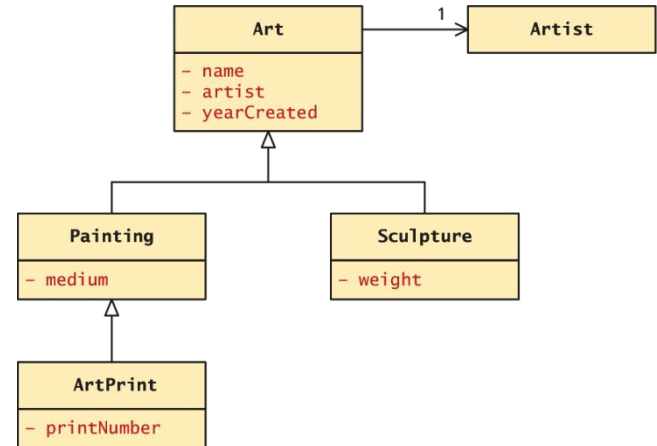
```
    //...constructor, getters, setters
```

```
    public function __toString() {
```

```
        return parent::__toString() . ", Medium: " .  
            $this->getMedium();
```

```
    }
```

```
}
```



Extended example

Sculptures have weight

```
class Sculpture extends Art {
```

```
    private $weight;
```

```
    //...constructor, getters, setters
```

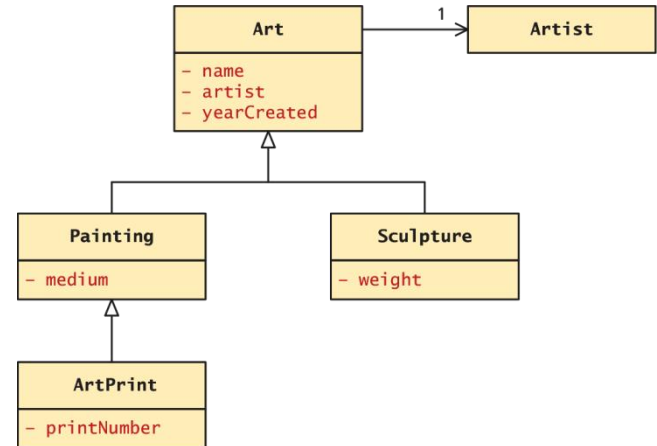
```
    public function __toString() {
```

```
        return parent::__toString() . ", Weight: " .
```

```
            $this->getWeight() . "kg";
```

```
    }
```

```
}
```



Extended example

Using the classes

...

```
$picasso = new Artist("Pablo","Picasso","Malaga","May 11,904","Apr 8, 1973");
```

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

```
$woman = new Sculpture("1909",$picasso,"Head of a Woman", 30.5);
```

```
?>
```

```
<h2>Paintings</h2>
```

```
<p><em>Use the __toString() methods </em></p>
```

```
<p><?php echo $guernica; ?></p>
```

```
<h2>Sculptures</h2>
```

```
<p> <?php echo $woman; ?></p>
```

Polymorphism

No thank you, I'll have water

Polymorphism is the notion that an object can in fact be multiple things at the same time.

Consider 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.

Polymorphism

```
$picasso = new Artist("Pablo","Picasso","Malaga","Oct 25, 1881",  
    "Apr 8, 1973");  
  
// create the paintings  
$guernica = new Painting("1937",$picasso,"Guernica","Oil on canvas");  
$chicago = new Sculpture("1967",$picasso,"Chicago", 454);  
  
// create an array of art  
$works = array();  
$works[0] = $guernica;  
$works[1] = $chicago;  
// to test polymorphism, loop through art array  
foreach ($works as $art)  
{  
    // the beauty of polymorphism:  
    // the appropriate __toString() method will be called!  
    echo $art;  
}  
  
// add works to artist ... any type of art class will work  
$picasso->addWork($guernica);  
$picasso->addWork($chicago);  
// do the same type of loop  
foreach ($picasso->getWorks() as $art) {  
    echo $art; // again polymorphism at work  
}
```

LISTING 10.10 Using polymorphism

Interfaces

Defining the interface

An object **interface** is a way of defining a formal list of methods that a class **must** implement without specifying their implementation.

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.

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

Interfaces

Defining the interface

An object **interface** is a way of defining a formal list of methods that a class **must** implement without specifying their implementation.

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.

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

Interfaces

Implementing the Interface

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 for the getSize() and getPNG() methods.

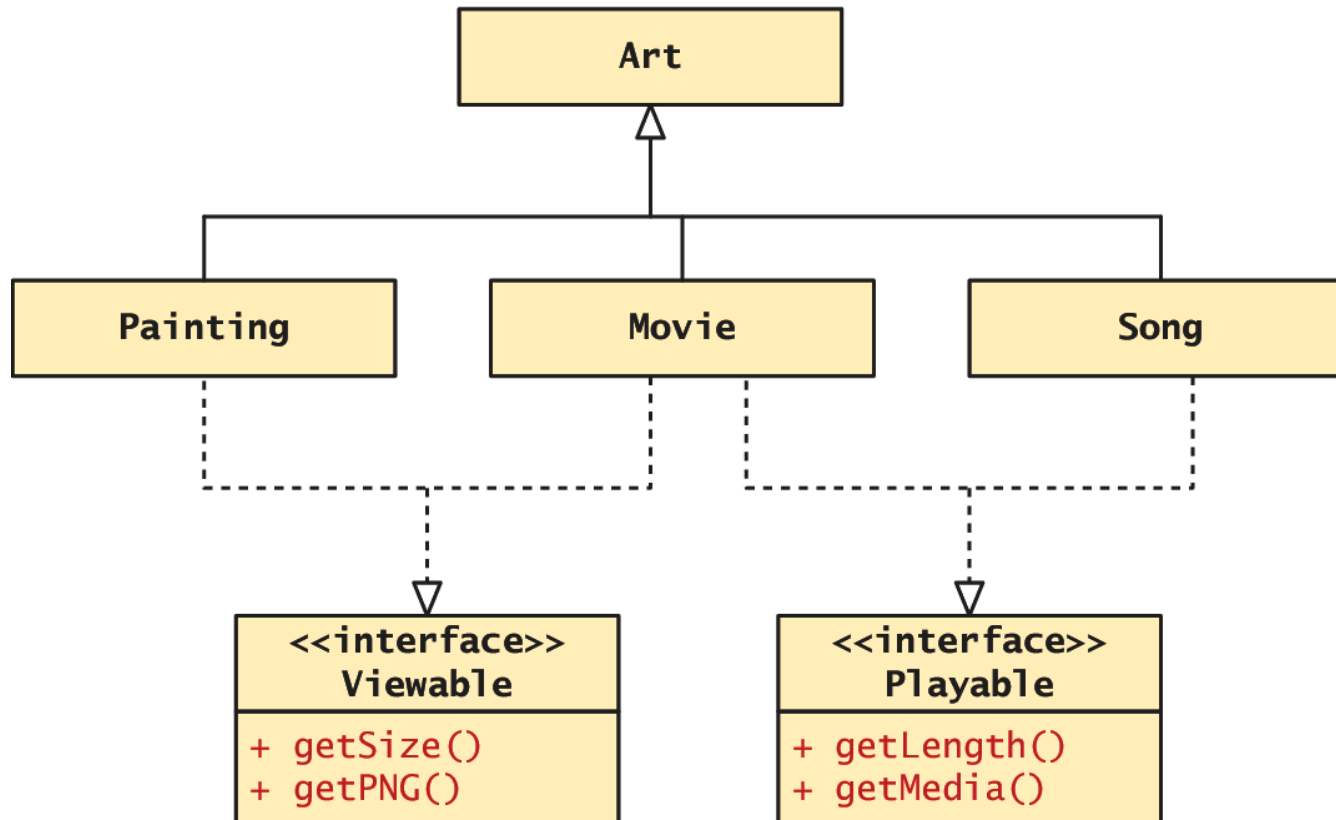
Interface Example

```
interface Viewable {  
    public function getSize();  
    public function getPNG();  
}  
  
class Painting extends Art implements Viewable {  
    ...  
    public function getPNG() {  
        //return image data would go here  
        ...  
    }  
    public function getSize() {  
        //return image size would go here  
        ...  
    }  
}
```

LISTING 10.11 Painting class implementing an interface

Interfaces

An Extended example



What You've Learned

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