Department of Computer Science and Engineering

THIRD INTERNAL QUESTION BANK– 6th Semester

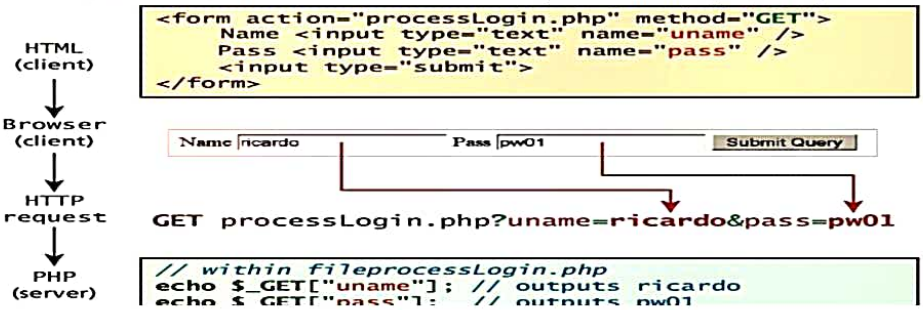
Subject Code : 18CS63 Subject Name: Web Technology and It's Applications

1. Explain $_GET and $_POST superglobal arrays with appropriate diagrams.
2. Explain the use of $_FILES array.
3. How do you achieve data encapsulation in PHP? Give example.
4. How to access methods and properties of class in PHP. Explain the use of static members
5. Explain the support for inheritance in PHP with UML class diagram
6. How is array defined in PHP? Explain the array operations of PHP.
7. Explain procedural error handling and object oriented exception handing with suitable code segments.
8. How do you read or write a file on the server from PHP? Give example.
9. Explain three approaches to restrict the file size in file upload with suitable code example.
10. Define constructor and discuss the concept of inheritance, polymorphism and object interface with respect to OOP.
11. Explain the $_SERVER array
12. Explain Serialization? What are its applications?
13. Explain the working of cookie with diagram
14. What are cookies? What is the purpose of it? Demonstrate cookies with PHP program
15. With suitable example explain AJAX GET and POST request.
16. Why is state is a problem for web applications? Explain.
17. Write short notes on i) XML ii) XPath
18. Discuss jQuery selectors in detail.
19. Discuss the following:- i) Session Cookies ii) Persistent Cookies iii) session state
20. What is session storage. how is it configured?

1) $_GET AND $_POST SUPERGLOBAL ARRAYS

$_GET and $_POST

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.

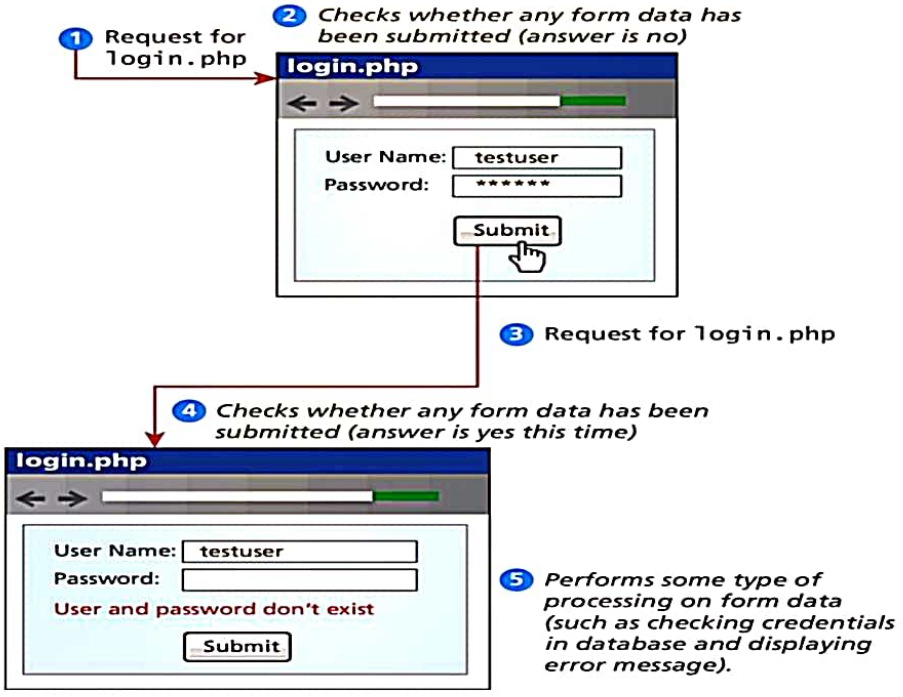


- Get requests parse query strings into the $_GETarray
- Post requests are parsed into the $POSTarray

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

```
<!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

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

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

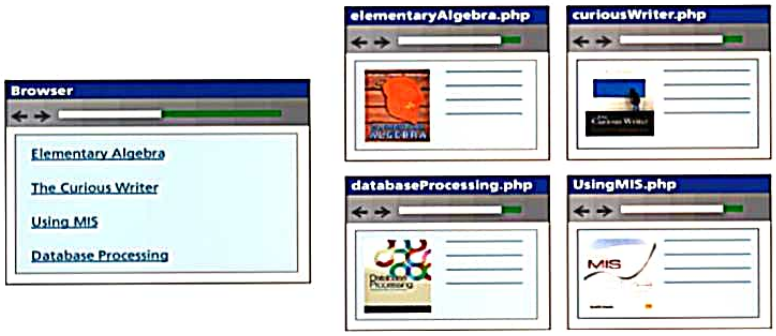
After making this change in the HTML, the corresponding variable $_GET ['day'] will now have a value that is of type array.

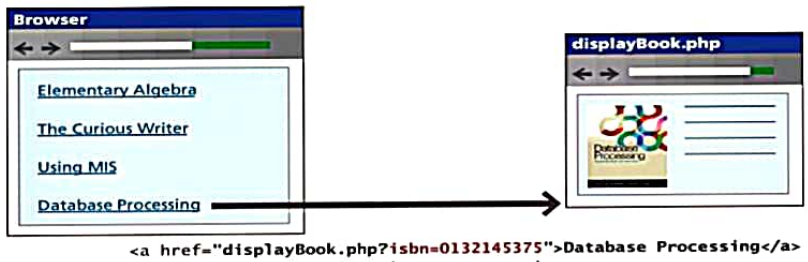
```
<?php

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

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.





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.

```
// 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

$_Files Array

2)

- The $_FILES associative array contains items that have been uploaded to the current script. <input type ="file"> Creates a user interface for uploading a file from the client to server.
- A server Script must process the upload files in some way ($_FILES array helps in this process)
- To allow users to upload files, there are some specific things you must do,
- First, you must ensure that the HTML form uses the HTTP post method, since transmitting a file through the URL is not possible.
- Second, You 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 (HTTP post body, the HTTP file attachment itself)

- Final you must include an input type of file in your 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 9.12 HTML for a form that allows an upload

- 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.

- **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 bein temporarily stored. The file will cease to exist upon termination of the scrip 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.

- A proper file upload script will therefore check each uploaded file by checking the various error codes as below,

```
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

3)

Data Encapsulation

- 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).

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

- getters
- setters

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

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

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

```
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;
}
}
```

4) Properties

- 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 (->) which is constructed from the dash and greater than symbols.

```
$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

Methods

- Objects only really become useful when you define behaviour 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.
- For our artist example one could write a method to convert the artist's details into a string of formatted HTML.
- Such a method is defined in Listing 10.4.

```
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

- To output the artist, you can use the reference and method name as follows:

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

- The UML class diagram in Figure 10.2 can now be modified to include the newly defined outputAsTable() method as well as the constructor and is shown in Figure 10.5.
- Notice that two versions of the class are shown in Figure 10.5, to illustrate that there are different ways to indicate a PHP constructor in UML.

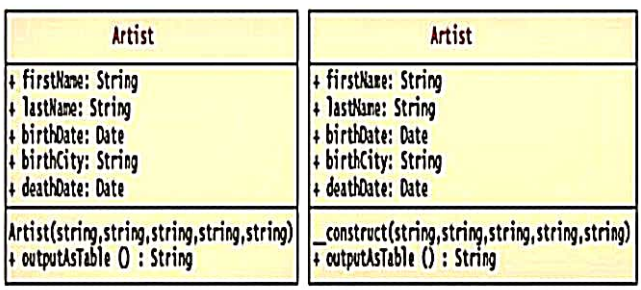


FIGURE 10.5 Updated class diagram

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.
- 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.
- For illustrative purposes we will also modify our constructor, so that it increments this value, as shown in Listing 10.5.

```
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 10.5 Class definition modified with static members

- Notice that you 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.
- To illustrate the impact of these changes look at Figure 10.7, where the shared property is underlined (UML notation) to indicate its static nature and the shared reference between multiple instances is illustrated with arrows, including one reference without any instance.

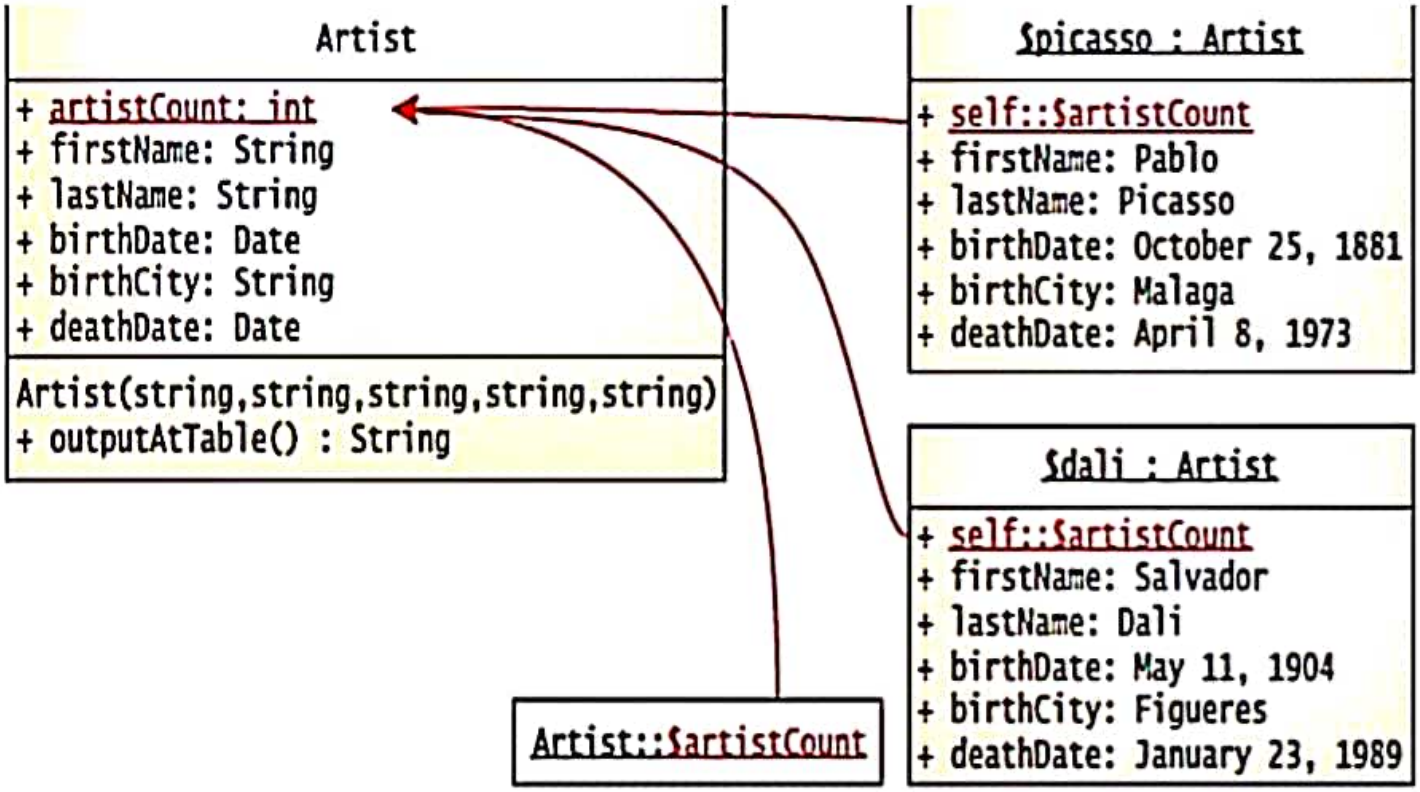


FIGURE 10.7 A static property

- 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.

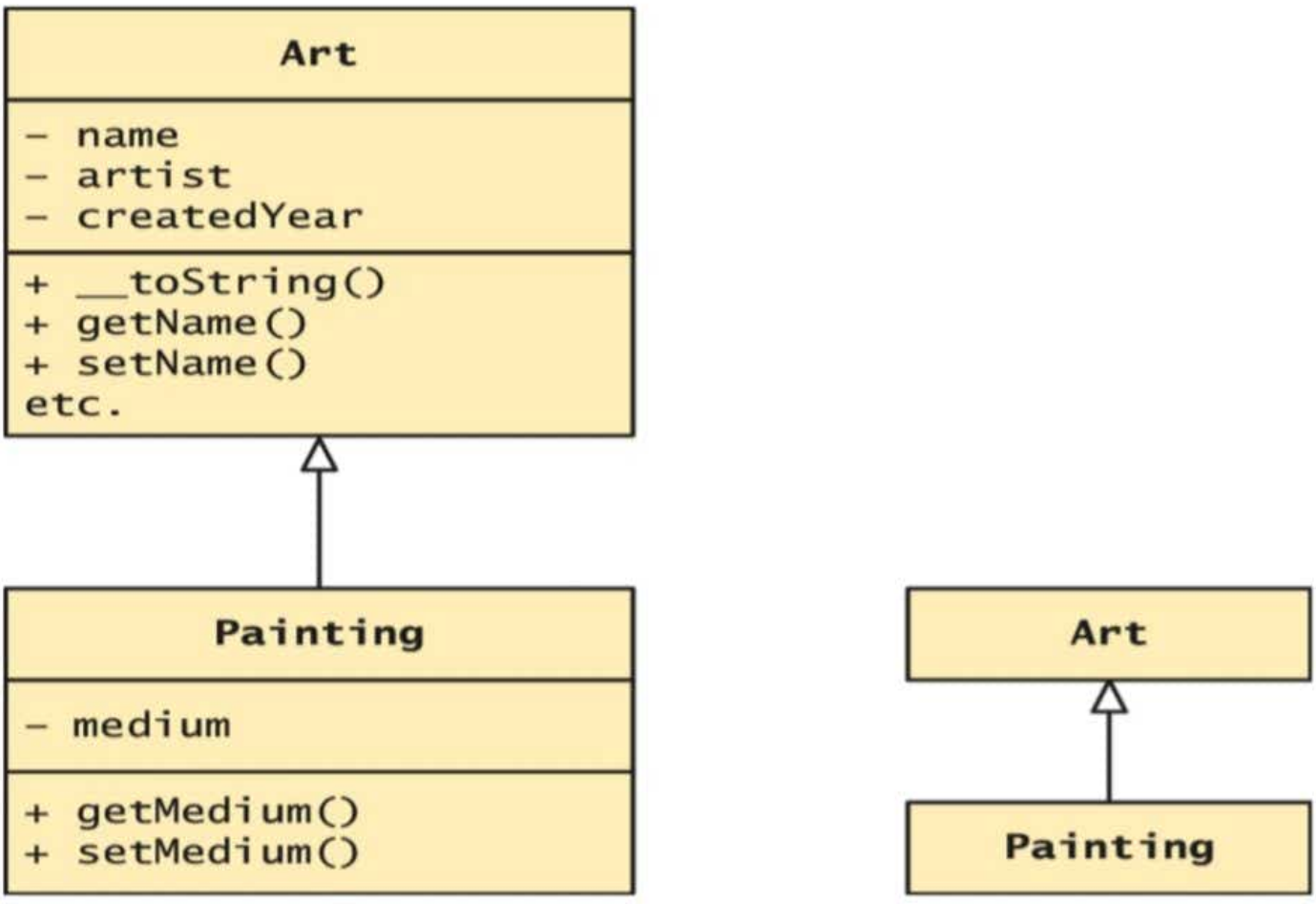
5) Inheritance

Inheritance enables you to create new PHP classes that reuse, extend, and modify the behaviour 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 super class or a base class A

PHP class is defined as a subclass by using the extends keyword. Class Painting extends Art {. . .}

```
$p = new Painting();
. . .
echo $p->getName(); // defined inbase class echo
$p->getMedium(); // defined in subclass
```

PHP Arrays and Superglobals

6)

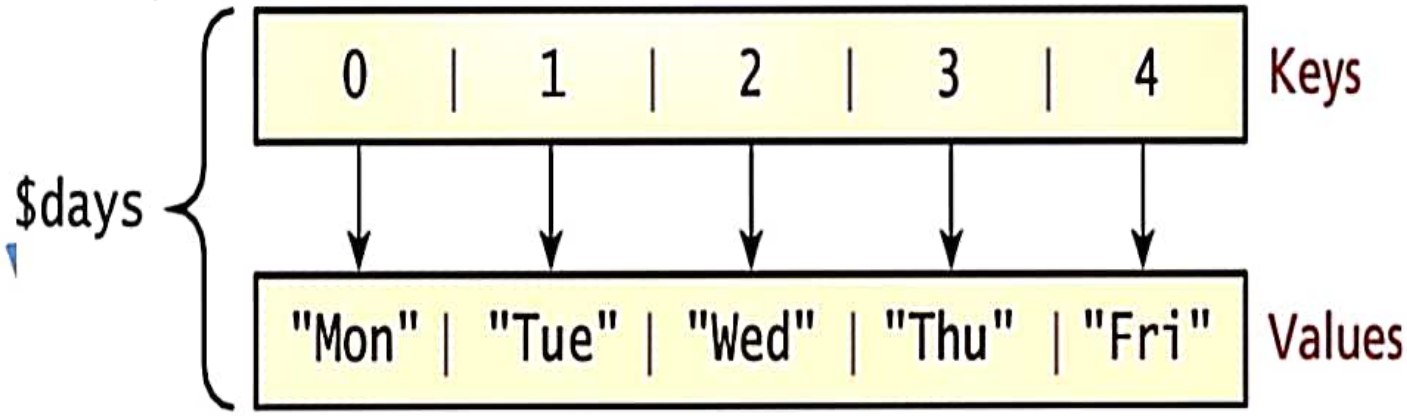
Arrays

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

- Add to the array
- Remove from the array

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



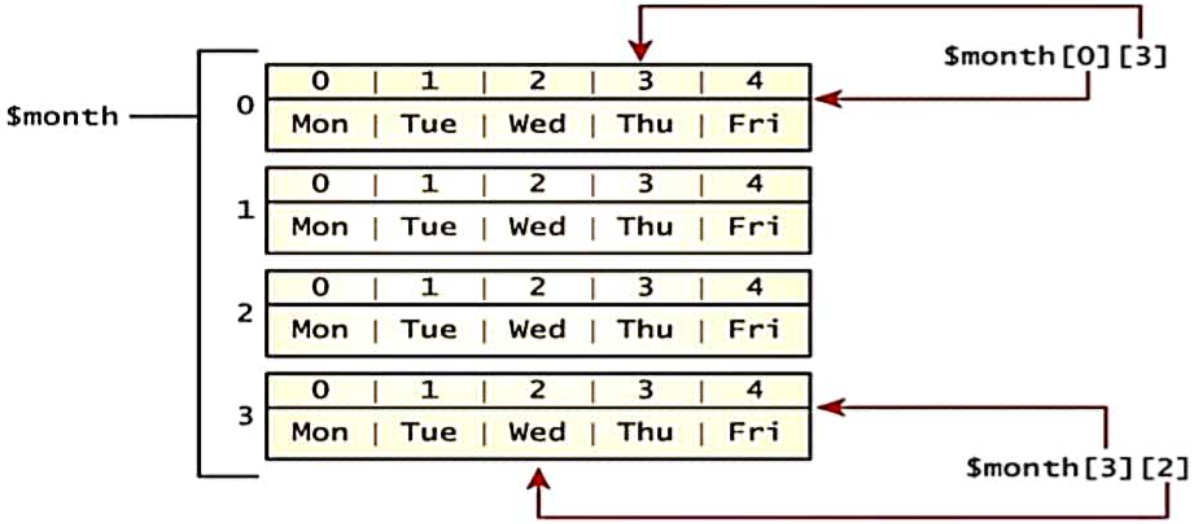
Array keys are the means by which you refer to single element in the array.
In most programming languages array keys are limited to integers, start at 0, and go up by 1.
In PHP, array keys *must* be either integers or strings and need not be sequential.

- Don't mix key types i.e. "1" vs1
- If you don't explicitly define them they are 0,1,…

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.
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*
To access values in an array you refer to their key using the square bracket notation. echo "Value at index 1 is ". $days[1];

Multidimensional 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
```



Iterating through an array

```php
// 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

The challenge of using the classic loop structures is that when you have non sequential 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.

```php
// 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

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

used $days[5] = "Sat";

Anew element can be added to the end of any array

$days[ ] ="Sun";

PHP is more than happy to let you "skip" an index

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

$days[7] = "Sat";

print_r($days);

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

```php
$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

You can explicitly delete array elements using the unset () function. array_values() reindexes the array numerically


```php
$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

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

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.

7) 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.
- 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.
- As well, it might result in a great deal of code duplication.
- The advantage of the try . . . catch mechanism is that it allows the developer to handle a wider variety of exceptions in a single catch block.

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

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

LISTING 12.2 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" message.
- Like other object-oriented programming languages, PHP uses the try . . . Catch programming construct to programmatically deal with exceptions at runtime.

```
// 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 12.3 Example of try . . . catch block

8) 9.5 Reading/Writing Files

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

1.Stream Access : In this technique, our code will read just a small portion of the file at a time.

2. All – In – Memory access: In this technique, we can read the entire file into memory (I.e., into PHP variable).

9.5.1 Stream Access

- 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.**
- 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, you 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, and if it reads a line it will advance the stream forward to the next one so you can use the `===` check to see if you have reached the end of the file. 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 9.19 illustrates a script using `fopen()`, `fgets()`, and `fclose()` to read a file and echo it out (replacing new lines with
 tags).

```
$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

9.5.2 In-Memory File Access

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 9.3 In-Memory File Functions

To Read and Write an entire file into variable

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

To write the contents of a string $writeme to a file, you use

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

- For instance, let us 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

- To read and then parse this text file as shown below in listing 9.20.

```
// 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

9) 9.4.4 File Size Restrictions

- Some scripts limit the file size of each upload.
- There are many reasons to do so, and ideally you 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

– Via JavaScript in the input form

– Via PHP coding.

- The first of these mechanisms is to add a hidden input field before any other input fields in your HTML form with a name of MAX_FILE_SIZE.
- 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 9.14 Limiting upload file size via HTML

- The more complete client-side mechanism to prevent a file from uploading if it is too big is to pre validate the form using JavaScript.

```
<script>
var file = document.getElementById('file1');
var max_size = document.getElementById("max_file_size").value;
if (file.files && file.files.length ==1){
   if (file.files[0].size > max_size) {
      alert("The file must be less than " + (max_size/1024) + "KB");
      e.preventDefault();
   }
}
</script>
```

LISTING 9.15 Limiting upload file size via JavaScript

- The third (and essential) 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).
- 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 9.16 Limiting upload file size via PHP

Constructors

10)

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.

```
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

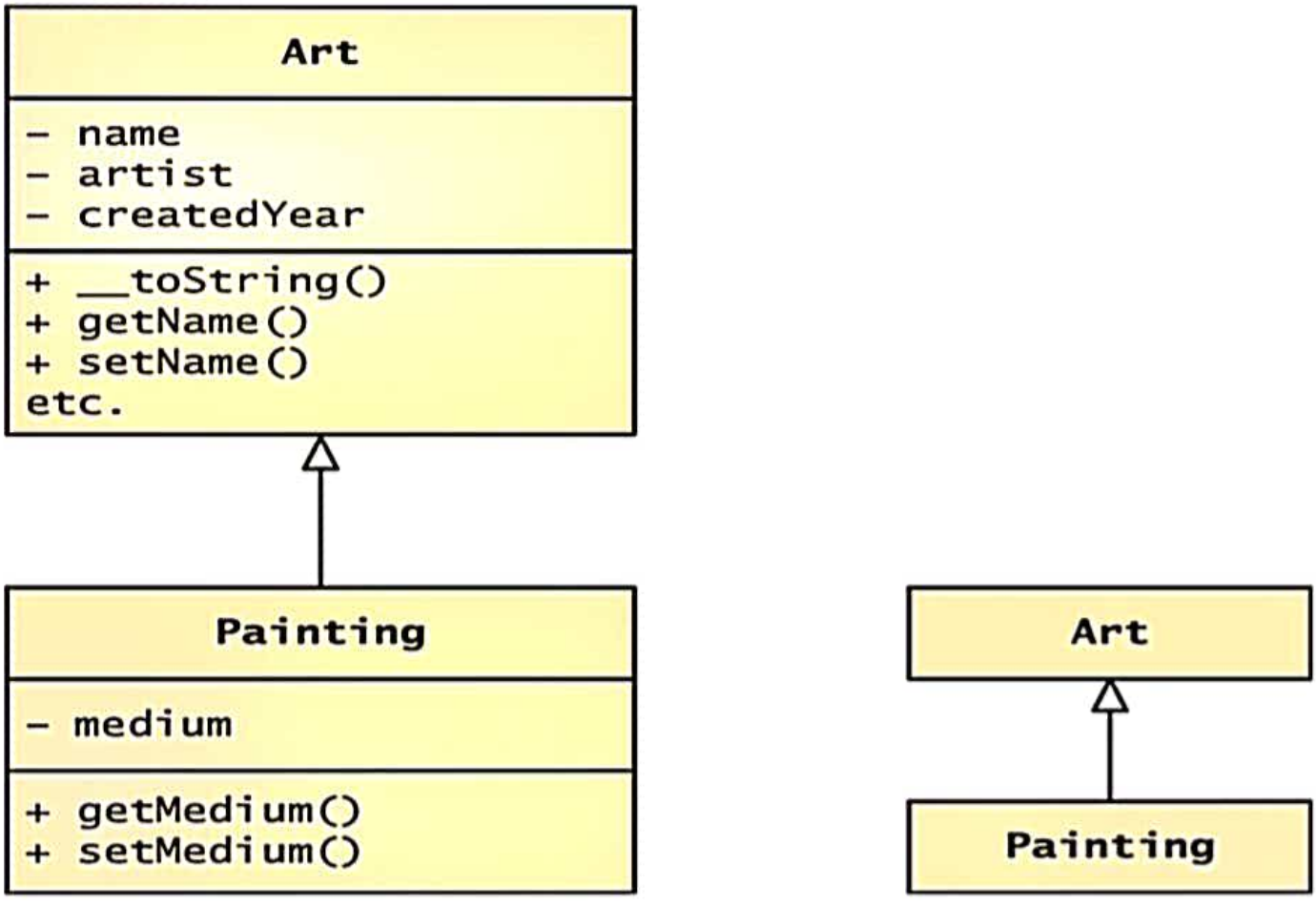
Inheritance

Inheritance enables you to create new PHP classes that reuse, extend, and modify the behaviour 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 super class or a base class A

PHP class is defined as a subclass by using the extends keyword. Class Painting extends Art {. . .}

```
$p = new Painting();
. . .
echo $p->getName(); // defined inbase class echo
$p->getMedium(); // defined in subclass
```

Polymorphism

- **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.
- Listing 10.10 illustrates polymorphism at work.

```
$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);
```

(continued)

```
// 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

- Due to overriding methods in child classes, the actual method called will depend on the type of the object!
- Using __toString() as an example, a Painting will output its name, date, and medium and a Sculpture will output its name, date, and weight.
- The code in Listing 10.10 calls echo on both a Painting and a Sculpture with different output for each shown below:

Date:1937, Name:Guernica, Medium: Oil on canvas

Date:1967, Name:Chicago, Weight: 454kg

- The formal notion of having a different method for a different class, all of which is determined at run time, is called **dynamic dispatching.**

Interfaces

- 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();
}
```



```
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