EAST WEST UNIVERSITY

CSE479
Web Programming

Nishat Tasnim Niloy

Lecturer

Department of Computer Science and Engineering

Faculty of Science and Engineering

Topic 12

PHP and OOP

Object Orientated Concept

- **Classes**, which are the "blueprints" for an object and are the actual code that defines the properties and methods.
- **Objects**, which are running instances of a class and contain all the internal data and state information needed for your application to function.
- **Encapsulation**, which is the capability of an object to protect access to its internal data
- **Inheritance**, which is the ability to define a class of one kind as being a sub-type of a different kind of class (much the same way a square is a kind of rectangle).

What is a Class?

Classes

- **Sophisticated** **'variable types'**
- Data variables (**data members**) and functions (**methods**) are wrapped up in a class. Collectively, data members and methods are referred to as **class members**.

An **instance of a class** is known as an **object**.

```
<?php              // save the file named "test.php"
        class Demo
        {
                public $name;

                function SayHello($n)
                {
                        echo "Hello: ". $n;
                }

        }
?>
```

Instantiating a class using new

- Once a class has been created, **any number** of object instances of that class can be created.
- $objDemo = new Demo();
- To access an object's property, you use the

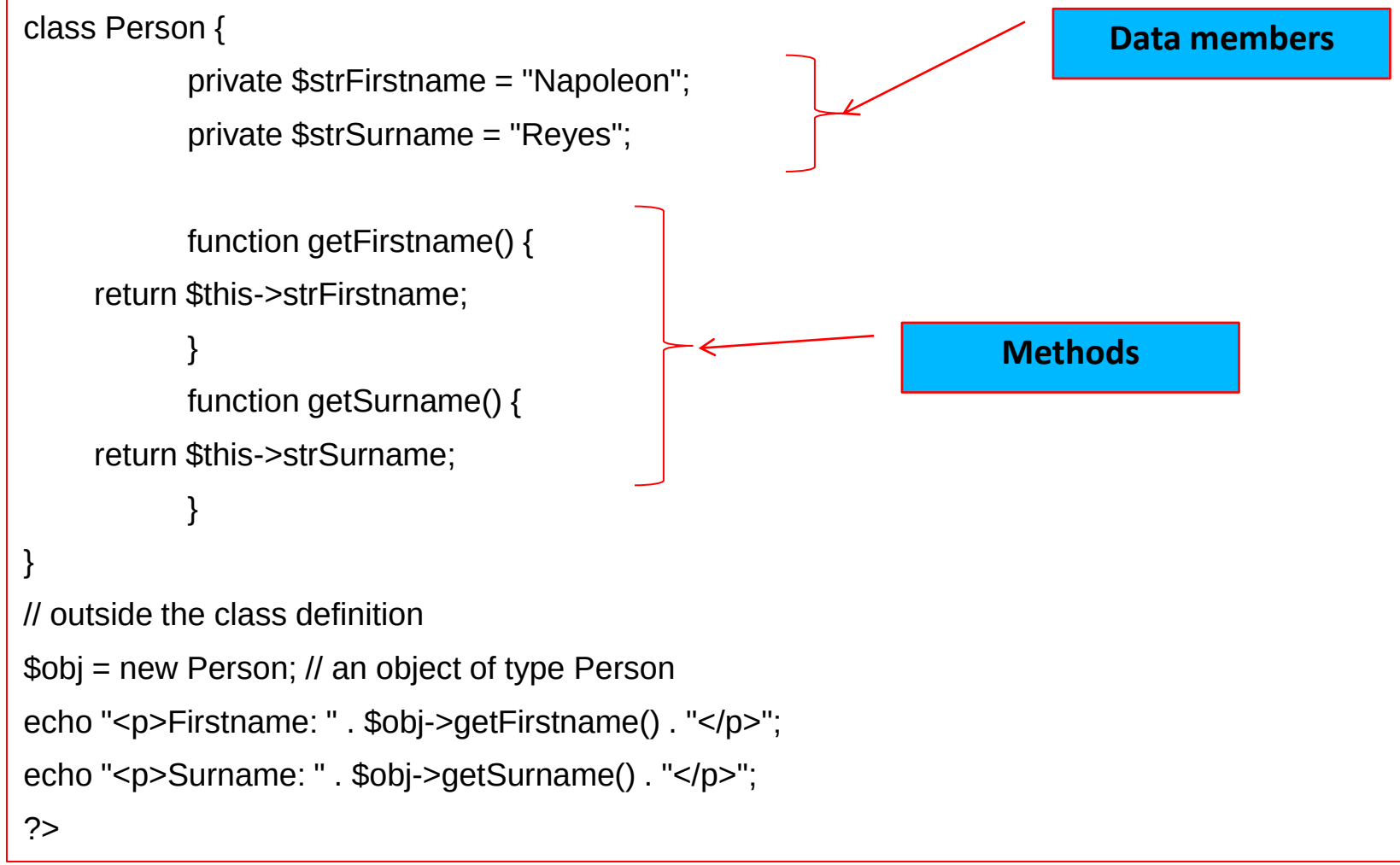
  -> operator

- Example: Invoke methods:
  - object->method()

```
<?php

  require_once('test.php');

  $objDemo = new Demo();

  $objDemo->name = "How are You All";

  $objDemo->SayHello($objDemo->name);

?>
```

Defining classes

```
<?php
class Person {
        private $strFirstname = "Napoleon";
        private $strSurname = "Reyes";

        function getFirstname() {
    return $this->strFirstname;
        }
        function getSurname() {
    return $this->strSurname;
        }
}
// outside the class definition
$obj = new Person; // an object of type Person
echo "<p>Firstname: " . $obj->getFirstname() . "</p>";
echo "<p>Surname: " . $obj->getSurname() . "</p>";
?>
```



Example: Class Multiple Instance.php

```
<?php
 class MyClass
{  public $prop1 = "I'm a class property!";
   public function setProperty($newval)
 {
     $this->prop1 = $newval;
 }
 public function getProperty()
 {
     return $this->prop1 . "<br />";
 }
}
// Create two objects
$obj = new MyClass;
$obj2 = new MyClass;
// Get the value of $prop1 from both objects
echo $obj->getProperty();
echo $obj2->getProperty();
// Set new values for both objects
$obj->setProperty("I'm a new property value!");
$obj2->setProperty("I belong to the second instance!");
 // Output both objects' $prop1 value
echo $obj->getProperty();
echo $obj2->getProperty();
 ?>
```

Output

I'm a class property!
I'm a class property!
I'm a new property value!
I belong to the second instance!

Adding methods to a class

- Methods are simply functions that are part of a class.
- So it probably comes as no surprise that you create a method much like any other function — by using the function keyword. The only difference is that you should also add public, private or protected to the method definition, much as you do with properties:

Example: Method Example.php

```
<?php
class Member
{
public $username = "";
private $loggedIn = false;
public function login() {
$this->loggedIn = true;
}
public function logout() {
$this->loggedIn = false;
}
public function isLoggedIn() {
return $this->loggedIn;
}
}
```

```
$member = new Member();

$member->username = "Fred";

echo $member->username . " is " . (  $member->isLoggedIn()
    ? "logged in" : "logged out" ) . "<br>";

$member->login();

echo $member->username . " is " . ( $member->isLoggedIn()
    ? "logged in" : "logged out" ) . "<br>";

$member->logout();

echo $member->username . " is " . ( $member->isLoggedIn()
    ? "logged in" : "logged out" ) . "<br>";

?>
```

Output

Fred is logged out
Fred is logged in
Fred is logged out

Example: Another Method Example.php

```
<?php
class Dog
{  public $hungry = 'I am  hungry.';
        function eat($food)
        {
        $this->hungry = 'not so much.';
        }
    .
```

```
$dog = new Dog;
echo $dog->hungry."<br>";
$dog->eat('cookie');
echo $dog->hungry;
    ?>
```

Output
I am Hungry.
not so much.

Encapsulation

- **Data members** are normally set inaccessible from outside the class (as well as certain types of **methods**) **protecting them** from the rest of the script and other classes. This protection of class members is known as **Encapsulation**.
- The visibility of class members, (properties, methods), relates to how that member may be manipulated within, or from outside the class.
- There are three different levels of visibility that a member variable or method can have :
  - Public
    - members are accessible to any and all code
  - Private
    - members are only accessible to the class itself
  - Protected
    - members are available to the class itself, and to classes that inherit from it

Example: Public.php

```
<?php
class Mathematics
{
public $num;
    public function addTwo()
    {
    return $this->num+2;
    }
}
$math = new Mathematics;
$math->num = 2;
echo $math->addTwo();
?>
```

Output : 4

Example: Private.php

```
<?php
class mathematics{
    private $num;
            public function setNum($num)
            {    $this->num = $num;
            }
            public function addTwo()
            {            return $this->num+2;
            }
}
    $math = new mathematics;
    $math->setNum(2);
    echo $math->addTwo();
?>
```

Output : 4

Example: Private another Example.php

```
<?php

class Member {

private $username;

private $location;

private $homepage;

public function __construct( $username, $location,
    $homepage ) {

$this->username = $username;

$this->location = $location;

$this->homepage = $homepage;

}

public function showProfile() {

echo "";

echo "Username:$this->username". "</br>";

echo "Location:$this->location". "</br>";

echo "Homepage:<a href= $this->homepage> $this->homepage </a>".
      "</br>";

echo "";

}

}

$aMember = new Member( “mdh", “dhaka",
      "http://tinyurl.com/sadfsaasd" );

$aMember->showProfile();

?>
```

Output :

Username:mdh
Location:dhaka
Homepage: http://tinyurl.com/ sadfsaasd

Example: Protected Properties and Methods

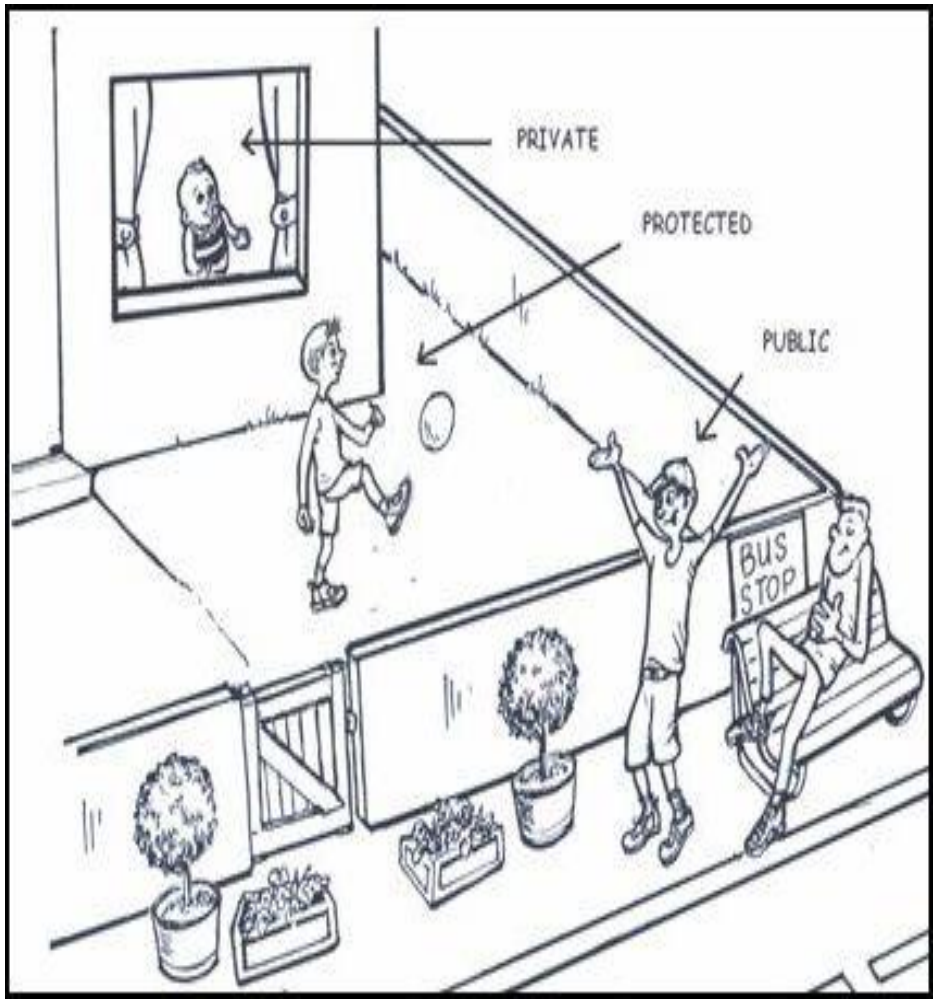
- When a property or method is declared protected, **it can only be accessed within the class itself or in descendant classes** (classes that extend the class containing the protected method). Attempt to call a protected method get an error echo $newobj->getProperty(); . create a new method in MyOtherClass to call the getProperty() method:

```
<?php
 class MyClass
{  public $prop1 = "I'm a class property!";
 public function __construct()
 { echo 'The class "', __CLASS__, '" was initiated!<br />';
 }
 public function __destruct()
 {echo 'The class "', __CLASS__, '" was destroyed.<br />';
 }
public function __toString()
 {  echo "Using the toString method: ";
    return $this->getProperty();
 }
 public function setProperty($newval)
 {  $this->prop1 = $newval;
 }
protected function getProperty()
{ return $this->prop1 . "<br />";    }
}
class MyOtherClass extends MyClass
{  public function __construct()
 { parent::__construct();
echo "A new constructor in " . __CLASS__ . ".<br />";
 }
 public function newMethod()
 {echo "From a new method in " . __CLASS__ . ".<br />";
 }
 public function callProtected()
 { return $this->getProperty();
 }
}
      $newobj = new MyOtherClass;
// Call the protected method from within a public method
echo $newobj->callProtected();
?>
```

The class "MyClass" was initiated!
A new constructor in MyOtherClass.
I'm a class property!
The class "MyClass" was destroyed.

Public Private Protected Scope

- public scope to make that variable/function available from anywhere, other classes and instances of the object.
- private scope when you want your variable/function to be visible in its own class only.
- protected scope when you want to make your variable/function visible in all classes that extend current class including the parent class.



Example: Static Properties and Methods

- To declare a method or property as static, we must use the static keyword. When accessing properties from outside the class scope, we use the class name followed by two :: and then the property name.

```
<?php
class Foo {
   static public $Name = "Alex Roxon";
   static public function helloWorld() {
      print "Hello world from " . self::$Name . "<br>";
   }
    public function nonStatic() {
      return self::$Name . "<br>";
   }
}class Bar extends Foo
{
   public function notStatic() {
      print "Hi " . self::$Name . "\n";
   }
    static public function aStatic() {
      print "Hello all" . "<br>";
   }
}
```

```
print Foo::$Name . "<br>";
Foo::helloWorld() . "<br>";
$foo = new Foo();
print $foo->nonStatic() . "<br>";
$bar = new Bar();
print $bar->notStatic() . "<br>";
print $bar::helloWorld() . "<br>";
?>
```

Output

Alex Roxon
Hello world from Alex Roxon
Alex Roxon

Hi Alex Roxon
Hello world from Alex Roxon

Constructors and destructors

- When you create a new object, there are sometimes things that are good to do at the same time.
    - For example, you might want to set some or all of the object's properties to initial values, or you might want to load the object's data automatically from a database table.
- Likewise, when an object is removed from memory, you might want to do things like removing dependent objects, closing files, or closing database connections.
- PHP provides a special function called **__construct()** to define a constructor.
- Like a constructor function you can define a destructor function using function **__destruct()**.
- How do you remove an object?

PHP automatically removes an object from memory when there are no more variables left that reference the object. For example, if you create a new object and store it in a variable called $myObject, and then remove the variable by calling **unset($myObject)**, the object is also removed from memory. Similarly, if you create $myObject as a local variable in a function, the variable — and therefore the object — are removed when the function exits.

Example: Constructors Initialize Properties.php

```
<?php

class Member {

private $username;

private $location;

private $homepage;



public function __construct( $username,
    $location, $homepage ) {

$this->username = $username;

$this->location = $location;

$this->homepage = $homepage;

}
```

```
public function showProfile() {
    echo "<dl>“;
    echo "<dt>Username:</dt><dd>$this-
    >username</dd>";

echo "<dt>Location:</dt><dd>$this->location</dd>";



echo "<dt>Homepage:</dt><dd><a href= $this-
    >homepage> $this->homepage </a></dd>";

echo "</dl>";

}

}

$aMember = new Member( “mdh", “dhaka",
    "http://tinyurl.com/fasdas" );

$aMember->showProfile();

?>
```

Output
Username:mdh
Location:dhaka
Homepage:http://tinyurl.com/ fasdas

Example: Constructor Practical Factorial Example

```
<?php
class Factorial
{
private $result = 1;
private $number;
function __construct($number)
{
$this->number = $number;
        for($i=2; $i<=$number; $i++)
            {
                $this->result*=$i;
            }
        echo "__construct() executed. ";
}
public function showResult()
{
echo "Factorial of {$this->number} is {$this-
    >result}. ";
}
}
$fact = new Factorial(5);
$fact->showResult();
public function __destruct()
{
echo "I'm about to disappear - bye bye!";
}
?>
```

Output

__construct() executed. Factorial of 5 is 120.
I'm about to disappear - bye bye!

Example: Destructor Example

- To call a function when the object is destroyed, the __destruct() magic method is available. This is useful for class cleanup (closing a database connection, for instance). To explicitly trigger the destructor, you can destroy the object using the function unset()

```
<?php
 class MyClass
{  public $prop1 = "I'm a class property!";
   public function __construct()
  {     echo 'The class "', __CLASS__, '" was initiated!<br />';
  }
  public function __destruct()
  {     echo 'The class "', __CLASS__, '" was destroyed.<br />';
  }
   public function setProperty($newval)
  {     $this->prop1 = $newval;
  }
   public function getProperty()
  {     return $this->prop1 . "<br />";
  }
}
```

```
// Create a new object
$obj = new MyClass;
// Get the value of $prop1
echo $obj->getProperty();
// Destroy the object
unset($obj);
// Output a message at the end of the file
echo "End of file.<br />";
?>
```

Output

The class "MyClass" was initiated!
I'm a class property!
The class "MyClass" was destroyed.
End of file.

Inheritance

New classes can be defined very similar to **existing ones**. All we need to do is specify the **differences** between the new class and the existing one.

Data members and **methods** which are **not** defined as being **private** to a class are automatically accessible by the new class.

This is known as **inheritance** and is an extremely powerful and useful programming tool.

Example: Inheritance example.php

```
<?php

class Car {
  private $model;

  public function setModel($model)
  {
    $this -> model = $model;
  }
  public function getModel()
  {
    return $this -> model;
  }
}
 class SportsCar extends Car{

  private $style = 'fast and furious';

  public function driveItWithStyle()
  {
    return 'Drive a '   . $this -> getModel() . ' <i>' . $this ->
      style . '</i>';
  }
}
```

```
$sportsCar1 = new SportsCar();

$sportsCar1 -> setModel('Ferrari');

echo $sportsCar1 -> driveItWithStyle();

?>
```

Output

Drive a Ferrari *fast and furious*

Example: Inheritance override parent properties

```
<?php
// The parent class has hello method that returns "beep".
class Car {
  public function hello()
  {
    return "beep";
  }
}

//The child class has hello method that returns "Helllo"
class SportsCar extends Car {
  public function hello()
  {
    return "Hello";
  }
}

//Create a new object
$sportsCar1 = new SportsCar();

//Get the result of the hello method
echo $sportsCar1 -> hello();
?>
```

Output

Hello

Example: prevent child class from overriding the parents methods

```
<?php
class Car {
  final public function hello()
  {
    return "beep";
  }
}

class SportsCar extends Car {
  public function hello()
  {
    return "Hallo";
  }
}

//Create a new object
$sportsCar1 = new SportsCar();

//Get the result of the hello method
echo $sportsCar1 -> hello();

 ?>
```

In order to prevent the method in the child class from overriding the parent's methods, we can prefix the method in the parent with the **final** keyword.

Thus output will show an error

Output

Fatal error: Cannot override final method Car::hello()

Example: Parents Class Constructor

- To add new functionality to an inherited method while keeping the original method intact, use the parentkeyword with the **scope resolution operator** (::)

```
<?php
 class MyClass
{  public $prop1 = "I'm a class property!";
   public function __construct()
  {     echo 'The class "', __CLASS__, '" was initiated!<br />';
  }
  public function __destruct()
  {    echo 'The class "', __CLASS__, '" was destroyed.<br />';
  }
       public function __toString()
  {   echo "Using the toString method: ";
     return $this->getProperty();
  }
  public function setProperty($newval)
  {     $this->prop1 = $newval;   }
public function getProperty()
  {     return $this->prop1 . "<br />“;    }
}
class MyOtherClass extends MyClass
{  public function __construct()
  {   parent::__construct(); // Call the parent class's constructor
      echo "A new constructor in " . __CLASS__ . ".<br />";
  }
  public function newMethod()
  {  echo "From a new method in " . __CLASS__ . ".<br />";
  }
}
$newobj = new MyOtherClass;
echo $newobj->newMethod();
echo $newobj->getProperty();
?>
```

Output

The class "MyClass" was initiated!
A new constructor in MyOtherClass.
From a new method in MyOtherClass.
I'm a class property!
The class "MyClass" was destroyed.

Polymorphism

A concept where a number of **related classes** all have a **method**, which **shares the same name**.

class **Fish { draw()... //draws a fish... }**
class **Dog { draw()... //draws a dog... }**
class **Bird { draw()... //draws a bird... }**

- We can write a generic code that **can operate on any** of these classes, invoking the appropriate **draw()** method based on certain conditions.
- To implement the polymorphism principle, we can choose between one of the two options of either abstract classes or interfaces.

Method Overloading, Overriding

- Overloading and Overriding are forms of polymorphism in OOP.
- According to Object Oriented Programming (OOP) concept if a class has methods of the same name but different parameters then we say that we are overloading that method.
- Also if we were to create a method in the child class having the same name, same number of parameters and the same access specifier as in its parent then we can say that we are doing method overriding.
- PHP does not support method overloading compared to other language like Java or C++. For example:

Example

```
<?php
class ABC {
    public function displayMessage($para1) {
        echo "First function displayMessage with parameter as para1";
    }
    public function displayMessage($para1,$para2) {
                echo "Second function displayMessage with parameter as para1\2";
    }
}
$obj1 = new ABC;
$obj1->displayMessage('Hello');
?>
```

- if above example code convert to Java or C++, it will work without any errors.
- But if we run above code, it will throw error the error “Cannot redeclare ABC::displayMessage()”.
- Simple overloading is not supported by PHP. But you can implement overloading by using the PHP magic method __call().

Overriding in PHP

```
<?php
class Foo {
  function myFoo() {
    return "Foo";
  }
}
class Bar extends Foo {
  function myFoo() {
    return "Bar";
  }
}
$foo = new Foo;
$bar = new Bar;
echo $foo->myFoo(). "<br/>"; //"Foo"
echo $bar->myFoo(). "<br/>"; //"Bar“
?>
```

- **Overriding** is only pertinent to derived classes, where the parent class has defined a method and the derived class wishes to **override** that method.

Output

Foo

Bar

Abstract Class

- We use abstract classes and methods when we need to commit the child classes to certain methods that they inherit from the parent class.
- An abstract class is a class that has at least one abstract method.
- Abstract methods can only have names and arguments, and no other code. Thus, we cannot create objects out of abstract classes. Instead, we need to create child classes that add the code into the bodies of the methods, and use these child classes to create objects.

Example: Abstract Class

```
<?php
 abstract class Car {
protected $tankVolume;
public function setTankVolume($volume)
 {
   $this -> tankVolume = $volume;
 }
abstract public function calcNumMilesOnFullTank();
}
class Honda extends Car {
public function calcNumMilesOnFullTank()
 {
   $miles = $this -> tankVolume*30;
   return $miles;
 }
}
```

```
class Toyota extends Car {
public function calcNumMilesOnFullTank()
 {   return $miles = $this -> tankVolume*33;
 }
 public function getColor()
 {   return "beige“;   }
}
$toyota1 = new Toyota();
$toyota1 -> setTankVolume(10);
echo "Toyota Details: " . $toyota1 -> calcNumMilesOnFullTank(). "</br>" ;
echo $toyota1 -> getColor() . "</br>" ;
$honda = new Honda();
$honda -> setTankVolume(10). "</br> ";
echo "Honda Details: " . $honda -> calcNumMilesOnFullTank();
?>
```

Output

Toyota Details: 330
beige
Honda Details: 300

Interface

- An interface is similar to a class except that it cannot contain code.
- An interface can define method names and arguments, but not the contents of the methods.
- Any classes implementing an interface must implement all methods defined by the interface. A class can implement multiple interfaces.

Example: Interface Class

```
<?php
interface Shape {
  public function calcArea();
}
class Circle implements Shape {
  private $radius;
 public function __construct($radius)
  {
    $this -> radius = $radius;
  }
   // calcArea calculates the area of circles
  public function calcArea()
  {
    return $this -> radius * $this -> radius * pi();
  }
}
```

```
class Rectangle implements Shape {
  private $width;
  private $height;
   public function __construct($width, $height)
  { $this -> width = $width;
    $this -> height = $height;
  }
 public function calcArea()
 {   return $this -> width * $this -> height;
  }
}
$circ = new Circle(3);
$rect = new Rectangle(3,4);
echo "Area of Circle:" . $circ -> calcArea(). "</br>";
echo "Area of Rectangle:" .  $rect -> calcArea(). "</br>“;
?>
```

Output

Area of Circle:28.274333882308
Area of Rectangle:12

Example: Polymorphism Class

```php
<?php
abstract class Shape {
    private $x = 0;    private $y = 0;
    public abstract function area();
}
class Rectangle extends Shape {
 function __construct($x, $y) {
        $this->x = $x;        $this->y = $y;
    }
  function area() {
     return $this->x * $this->y;
    }
}
class Square extends Shape {
    function __construct($x) {
        $this->x = $x;    }
    function area() {
        return $this->x * $this->x;    }
}
class Circle extends Shape {
    function __construct($x) {
        $this->x = $x;
    }
    function area() {
         return 3.1416 * $this->x * $this->x;
    }
}
$shapes = new Square(5);
echo "Square Area: " . $shapes->area() . "</br>";
$shapes = new Rectangle(12, 4);
echo "Rectangle Area: " .  $shapes->area() . "</br>";
$shapes = new Circle(2);
echo "Circle Area: " .  $shapes->area() . "</br>";
?>
```

Output

Square Area: 25
Rectangle Area: 48
Circle Area: 12.5664

Remember these PHP Constructs?

- **`require(….)`**
  - Includes file specified, **terminates on errors**
- **`include(…)`**
  - Includes file specified, gives **warning on errors**

- **`require_once(….)`**
  - Includes file specified only if it has not already been included, **terminates on errors**
- **`include_once(….)`**
  - Includes file specified only if it has not already been included, gives **warning on errors**