Object-Oriented Programming in PHP

Classes, Objects, Interfaces, Inheritance, Polymorphism

Mario Peshev

Technical Trainer

http://peshev.net

Software University

http://softuni.bg

Table of Contents

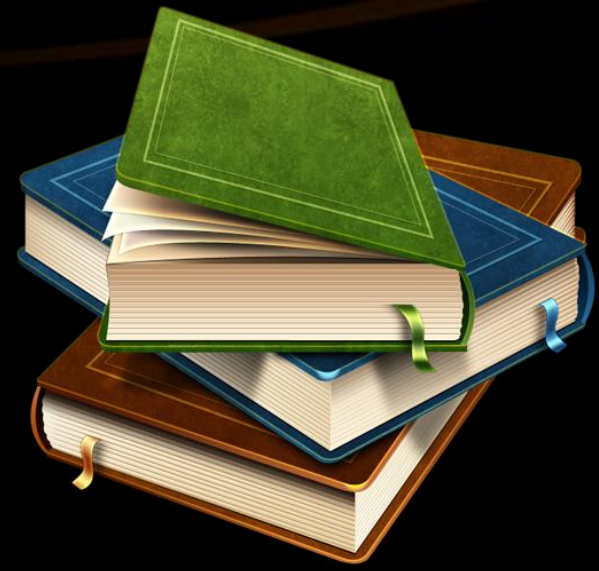
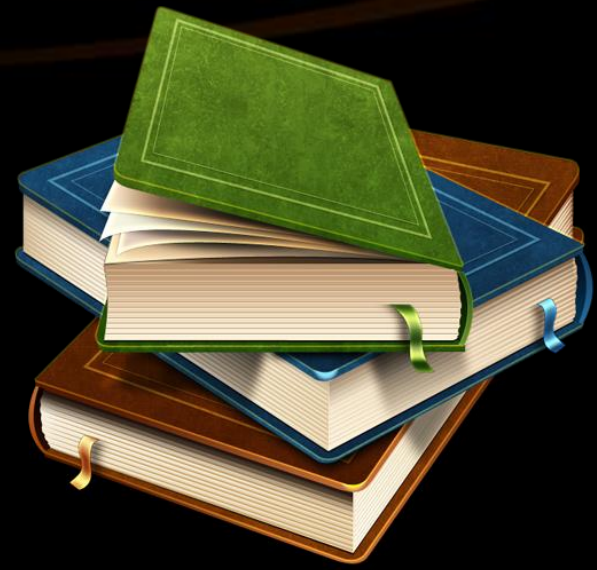
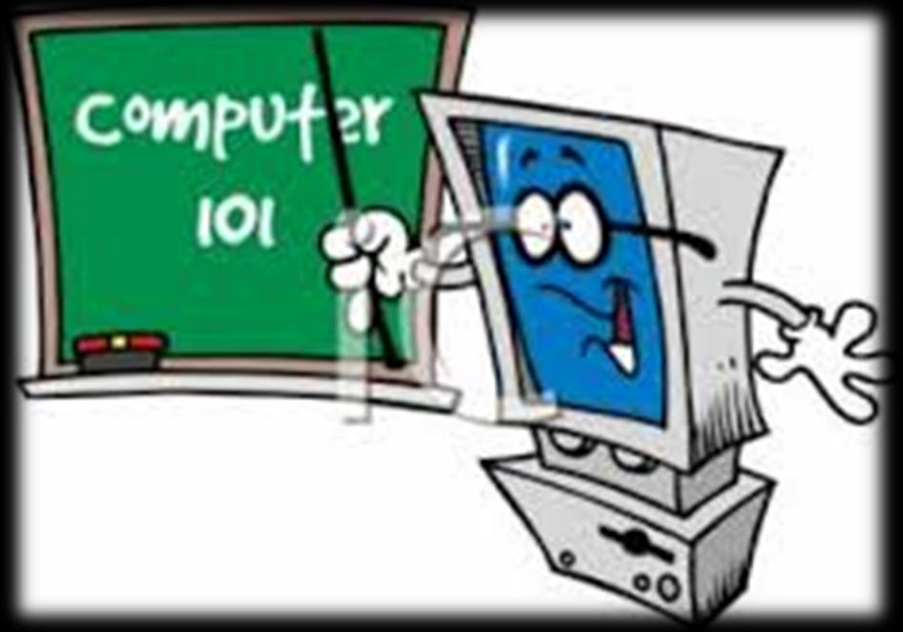


Table of Contents



SOFTWARE UNIVERSITY FOUNDATION

Classes And Objects



Object Oriented Programming

- The idea of Object Oriented Programming is to move the architecture of an application closer to real world
  - Classes are types of entities
  - Objects are single units of a given class
  - Example – Cat is a class, your dog Sharo is an object of class Dog
  - Classes have methods and properties
  - Classes and objects help to create well-structured application

Classes in PHP

- Declaring of a class in PHP can be done anywhere in the code

```
class Foo {
    //Methods and Properties
}
```

  - Two special methods: constructor and destructor
    - Executed when creating or destroying new objects of this class
    - Used to initialize or cleanup properties and etc.

Classes in PHP

- Class definition begins with the `class` keyword, followed by its name and met… and pr…perties list

Class Name

Object Method and body

```
class Foo {
    function boo() {
        echo "This is Foo";
    }
}

$myFirstObject = new Foo();
$myFirstObject->boo(); //Output: This is Foo
```

Object creating

Call object method

- Objects of class (instances) are created with the keyword `new`

Constructors

- Each class can have only one constructor

```
class Foo {
    function __construct ($boo) {
        echo $boo;
    }
    function boo () {
        echo "boo function";
    }
}
$myFirstObject = new Foo("Constructor");//Output: Constructor
$myFirstObject->boo(); //Output: This is Foo
```

  - All parameters of the creating of the object are passed to the constructor

Properties

- Class can have unlimited number of properties

```
class Foo {
    var $prop;
    function __construct ($boo) {
        $this->prop = $boo;
    }
    function boo () {
        echo $this->prop;
    }
}
$myFirstObject = new Foo("This is Property");
$myFirstObject->boo(); //Output: This is Property
```

- The `$this` variable points to the current object – called execution context

$this

- Can be used to access methods too

```
class Foo {
    function printClassName() {
        echo $this->getName();
    }
    function getName() {
        return  get_class($this);
    }
}

$fooObj = new Foo();
$fooObj->printClassName(); //Output: Foo
```

More About Properties

- Class can specify default value for a property

```
class Foo {
    var $prop = "Default value";
…
```

- Properties can be accessed from the outside world

```
class Foo {
    var $prop = "Default value";
…
}
$fooObj = new Foo();
$fooObj->$prop; //Output: This is Property
```

Destructor

- Each class can have only one destructor
  - Must be public

```
class Foo {
    function  __destruct(){
        echo "Call ";
    }
}
$fooObj = new Foo();
```

  - The destructor method will be called as soon as there are no other references to a particular object, or in any order during the shutdown sequence.

Classes And Objects

Live Demo

Scope

- Each method and property has a scope
  - It defines who can access it
  - Three levels – public, protected, private
    - Private can be access only by the object itself
    - Protected can be accessed by descendant classes *(see inheritance)*
    - Public can be accessed from the outside world
  - Level is added before the `function` keyword or instead of `var`
    - `var` is old style (PHP 4) equivalent to public
  - Constructors always need to be public

Scope Example

```
class Foo {
    public function printObjName() {
        echo $this->getName();
    }
    private function getName() {
        return get_class($this);
    }
}
$fooObj = new Foo();
$fooObj->printObjName();//Output: Foo
//$fooObj->getName(); This will not work
```

The getName() method is private so only the object can access it

The printObjName method is public, so everyone can call it

Inheritance

- A class can inherit (extend) another class
  - It inherits all its methods and properties

```
class Foo {
    public $fooProp = "Foo Property";
    public function printProp() {
        echo $this->fooProp;
    }
}
class Boo extends Foo {}

$booObj = new Boo();
$booObj->printProp();//Output: Foo Property
```

Protected

- Method or property, declared as `protected` can be accessed in classes that inherit it, but cannot be accessed from the outside world

```
class Foo {
    protected $fooProp = "Foo Property";
}
class Boo extends Foo {
    public function printProp() {
        echo $this->fooProp;
    }
}

$booObj = new Boo();
$booObj->printProp(); //Output: Foo Property
//$booObj->$fooProp; Will not work
```

Overriding

- When a class inherits another, it can declare methods that override parent class methods
  - Method names are the same
  - Parameters may differ

```
class Foo {
    public function printProp(){}
}
class Boo extends Foo {
    public function printProp(){}
}
```

Parent Class

- As `->` is used to access an object's methods and properties, the `::` (double colon) is used to change scope
  - Scope Resolution Operator
- `parent::` can be used to access parent's class overridden methods
  - Example: call parent's constructor in the child one

Parent Access

The static Keyword

- Defining method or property as 'static' makes them accessible without needing an instantiation of a class
  - Accessed with the double-colon (`::`) operator instead of the member (`->`) operator
  - `$this` is not available in static methods
  - Static properties and methods can also have scope defined – public, private or protected

The static Keyword

- Example of static method and property

```
class Foo {
    public  static  $myProp = "Static Property";
    public static function printProp() {
        echo self::$myProp;
    }
}
$booObj = new Foo();
$booObj::printProp(); //Output: Static Property
```

- Class can access statics with the `self` keyword
- Outside world accesses statics with the class name

Class Constants

- Constants in PHP usually are declared with the `define` function
- Constants can be defined in class
  - Differ from normal variables – no need for `$` symbol to declare and access
  - Declared with the `const` keyword
  - Value must be supplied with the declaration

```
const myConstant = 'value'
```

Class Constants

- Accessed with scope operator (`::`)
- Can be overridden by child classes
- Value must be constant expression, not a variable, class member, result of operation or function call

```
class Foo {
    const myConst = 'value';
}
class Boo extends Foo {
    const myConst = 'value2';
    public function printConst() {
        echo $this::myConst;
    }
}
```

Abstraction

- Classes, defined as abstract, cannot have instances (cannot create objects of this class)
  - Abstract class must have at least one abstract method
  - Abstract methods do not have implementation (body) in the class
    - Only signature
  - The class must be inherited
  - The child class must implement all abstract methods
    - Cannot increase visibility

Abstraction

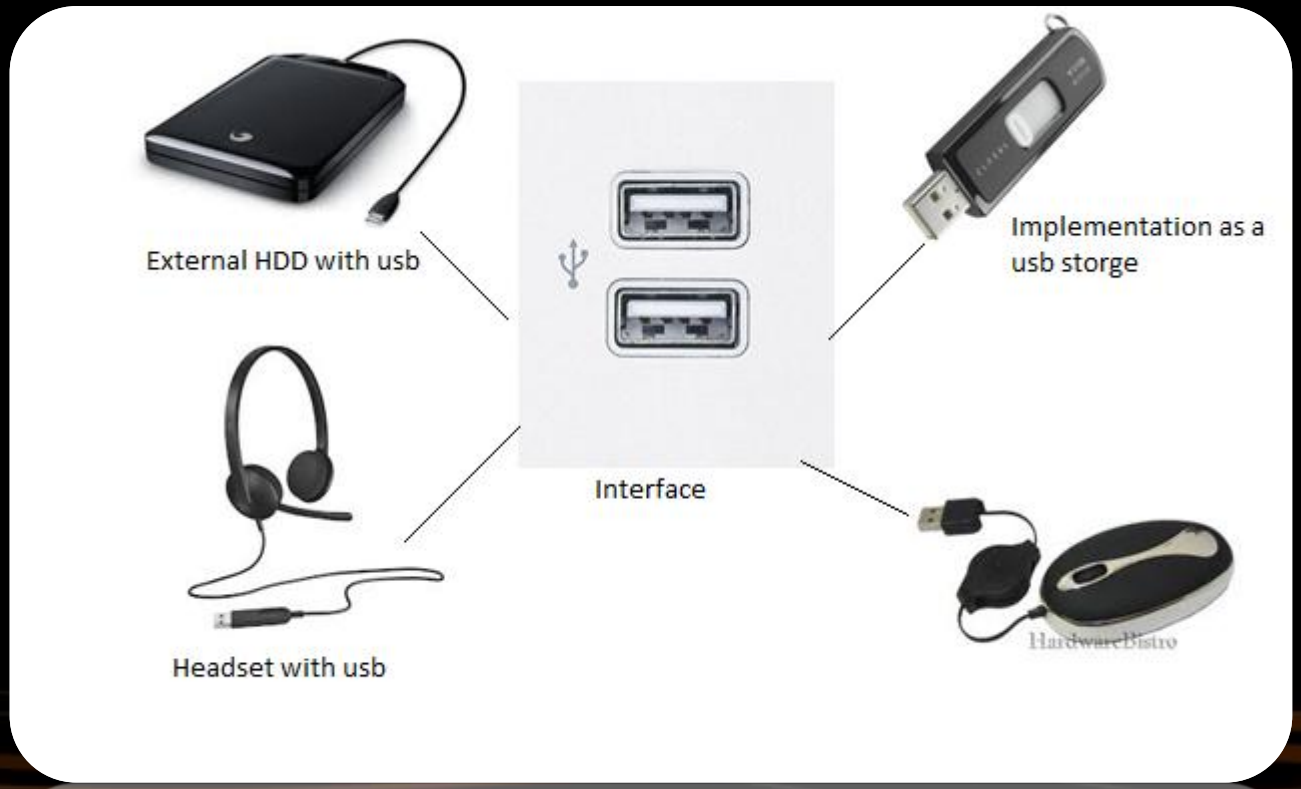
Live Demo

Interfaces

- Object interfaces allow you to specify what methods a child class must implement
  - Declared with the `interface` keyword
  - Similar to abstract class
  - Interface can have only public methods
  - No method in an interface can have implementation
- Interfaces are inherited with the `implements` keyword (instead of `extends`)
  - One class may implement multiple interfaces, if they do not have methods with same names

Interface

Live Demo



Overloading

- Overloading in PHP provides the means to dynamically create members and methods via set of "magical" methods
  - Invoked with interacting with members or methods that have not been declared or are not visible in the current scope
  - All of the magic methods must be declared as public
  - None of the magic functions can be called with arguments, passed by reference

Overloading Methods

- All overloading methods are invoked when accessing variable or method that is not declared or is inaccessible
- `__set($name, $value)` – when writing
- `__get($name)` – when reading
- `__isset($name)` – when calling `isset()` function
- `__unset($name)` – when calling `unset()` function

Overloading Methods

- `__call ($name, $arguments)` - when calling a method
- `__callStatic ($name, $arguments)` – when calling a method in a static context
  - Added after PHP 5.3
  - Must always be declared as static
- PHP "overloading" is a lot different from most languages "overloading"
  - Usually it means the ability to declare two methods with different sets of parameters but same names

Object Iteration

- PHP provides a way for object to be iterated trough as a list of items (array)
  - foreach can be used
  - By default iterates all visible properties

Overloading

Live Demo

Object Iteration

- To take object iteration a step further, you can implement one of the PHP interfaces
  - Provided by the Standard PHP Library
  - Allows the objects to decide what to show and what not
  - Some provided interfaces:
    - `Iterator` – very long to implement but provides dull features
    - `IteratorAggregate` – simple version of Iterator interface
    - `ArrayIterator`, `DirectoryIterator`, etc.

Object Cloning

- An object can be cloned with the `clone` keyword

```
$obj1 = new A();
$obj2 = clone $obj1;
```

  - This will create new independent object
- Creating a copy of an object with fully replicated properties is not always the wanted behavior

Object Cloning

- A class can implement the magic method `__clone` which is called for the newly created object
- Called "clone constructor"
- Allows necessary changes to be done on the newly created object
- Example: Object holding reference to resource – the new object must have new references, instead of copies
- Example: Object holding reference to another object that must not be copied

Cloning

Live Demo

Serializing Objects

- Serializing is the process of transforming an object into a string, that can be stored
  - This string can be used to restore the object
  - Useful for storing objects in session data
  - Saves only properties values and class names – no methods
  - PHP provides the `serialize` and `unserialize` functions

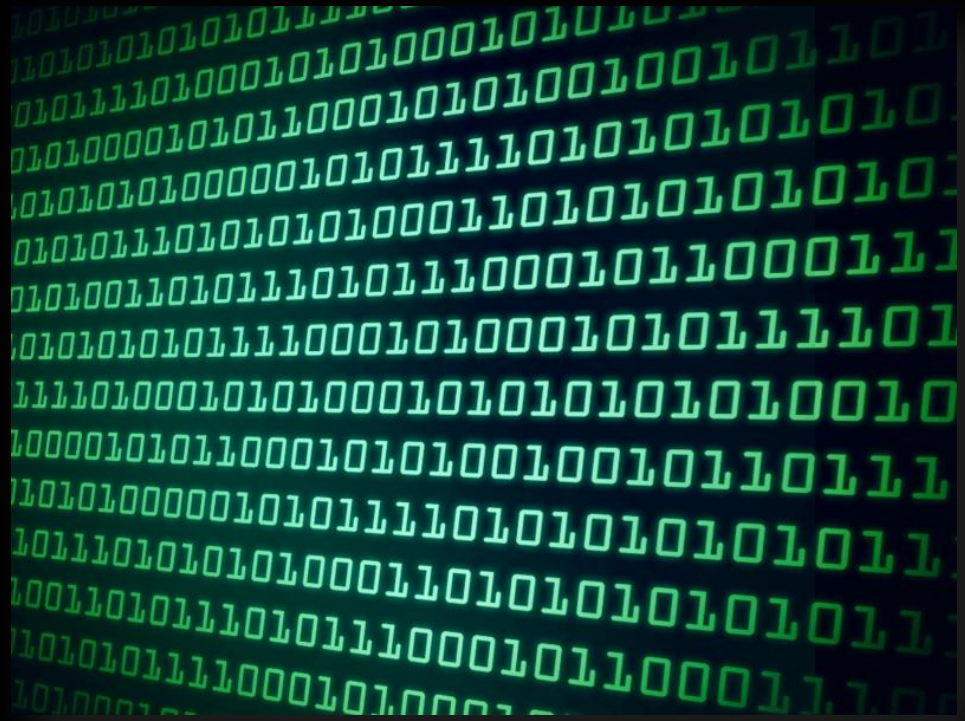
Serializing Objects

- `serialize ($object)` – returns string, representing the object
- `unserialize ($string)` – returns new object, that is restored from the serialized string
- `unserialize` requires the class to be defined before calling it

SOFTWARE UNIVERSITY FOUNDATION

Serialization

Live Demo



Serializing Methods

- Before serializing and after unserializing PHP checks if the class has the magic methods `__sleep` and `__wakeup`
  - `__sleep` allows the class to commit pending data, cleanup or define what needs to be stored if the object is very large
    - Should return array with names of properties to be stored
  - `__wakeup` allows the class to restore connections or other re-initialization

__sleep and __wakeup


```
class Connection {
        protected $link;
        private $server, $user, $pass, $db;

        public function __construct($server, $user, $pass, $db) {
                $this->server = $server;
                $this->user = $user;
                $this->pass = $pass;
                $this->db = $db;
                $this->connect();
        }

        private function connect () {
                $this->link = mysql_connect (
                        $this->server, $this->user,
                        $this->pass);
                mysql_select_db($this->db, $this->link);
        }
        // continues on next slide
```

__sleep and __wakeup


```
        // continues from previous slide

        public function __sleep () {
                // skip serializing $link
                return array ('server', 'user',
                        'pass', 'db');
        }

        public function __wakeup () {
                $this->connect();
        }
}
```

Namespaces

- Namespaces in PHP are designed to resolve scope problems in large PHP libraries
  - Simplify development in object oriented environment
  - Clears the code – no long classes names
- In PHP all classes declarations are global
  - Namespaces allow to have two classes with same name
  - Old approach was adding prefixes to class names (Like the `mysql_* functions`)
- Available since PHP 5.3

Namespace Definition

- Namespaces are declared with the `namespace` keyword
  - Should be always in the beginning of the file

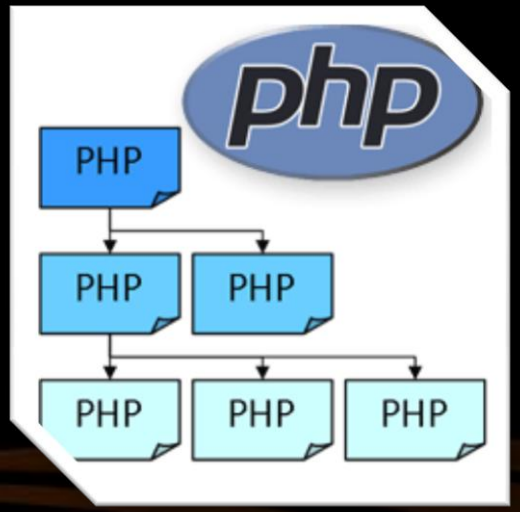
```
<?
namespace Project;

class MyTemplate { … }
function print_headers () { … }
…
?>
```

  - Namespace can contain classes, constants, functions but no free code

Namespaces

- Classes, function and etc. in a namespace are automatically prefixed with the name of the namespace
  - So in the example we would use `Project\MyTemplate` to access the class
  - Constants in namespaces are defined with `const` keyword, not with `define`



Namespaces – Example


```
// file Project.php
namespace Project;
// declare base classes and etc.
…
// file project/db.php;
namespace Project\DB;
// declare DB interface for work with database
…
// file project/db/mysql.php
namespace Project\DB\MySQL;
// implement the DB interface for mysql
…
// somewhere in the project
require "project/db/mysql.php";
$a = new Project\DB\MySQL\Connection();
Project\DB\MySQL::connect();
```

Using Namespaces

- The use operator allows aliasing namespaces names

```
use Project\DB\MySQL as DBLink;
$x = new DBLink::Connection();
DBLink::connect();
```

  - If new name is not specified the namespace is imported in the current context (global namespace)

```
use Project\DB\MySQL;
$x = new MySQL::Connection();
MySQL::connect();
```

- Even if aliased, every class and function can be accessed at any time by full name

Global Namespace

- By default PHP works in the global namespace
  - All the project is executed there
  - Method from the global namespace can be referred to with empty scope operator

```
namespace Project\Files;
// this is the Project\\Files::fopen function
function fopen (…) {
        …
        $f = ::fopen (…); // calls global fopen
        …
}
```

Autoloading Classes

- Usually every class is declared in separate file
  - In big object oriented projects on every page you may have to include dozens of files
  - You can define `__autoload` function that is called when trying to access class that is not defined
    - It can include the necessary file for the class

Autoload Example

```
<?
function __autoload ($class_name) {
        $name = "includes/".$class_name.".inc.php";
        if (file_exists ($name))
                include $name;
        else
                echo 'Class not found';
}
?>
```

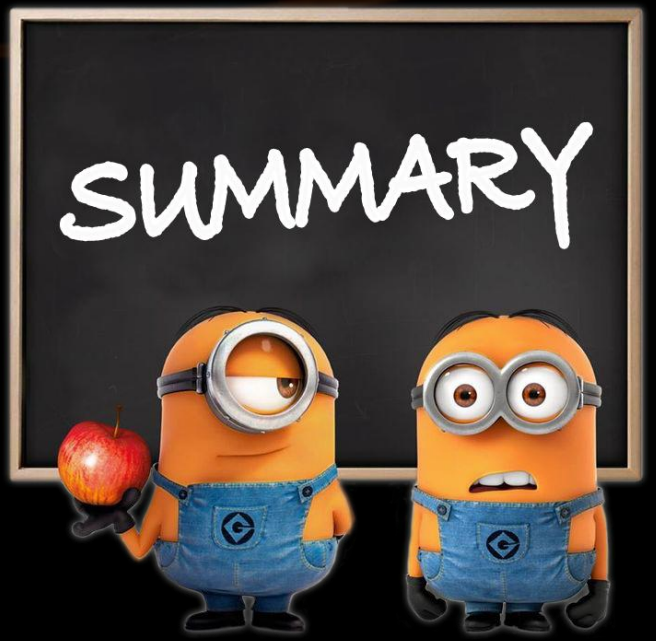
- Exceptions, thrown in `__autoload` cannot be caught and result in fatal error

Late Static Binding

- PHP 5.3 introduces the late static binding which allows to reference the called class in context of static
  - In practice – this adds static`::` scope
  - So if in the above example we use `static::whoami()` in the `test()` method body we get output 'B'

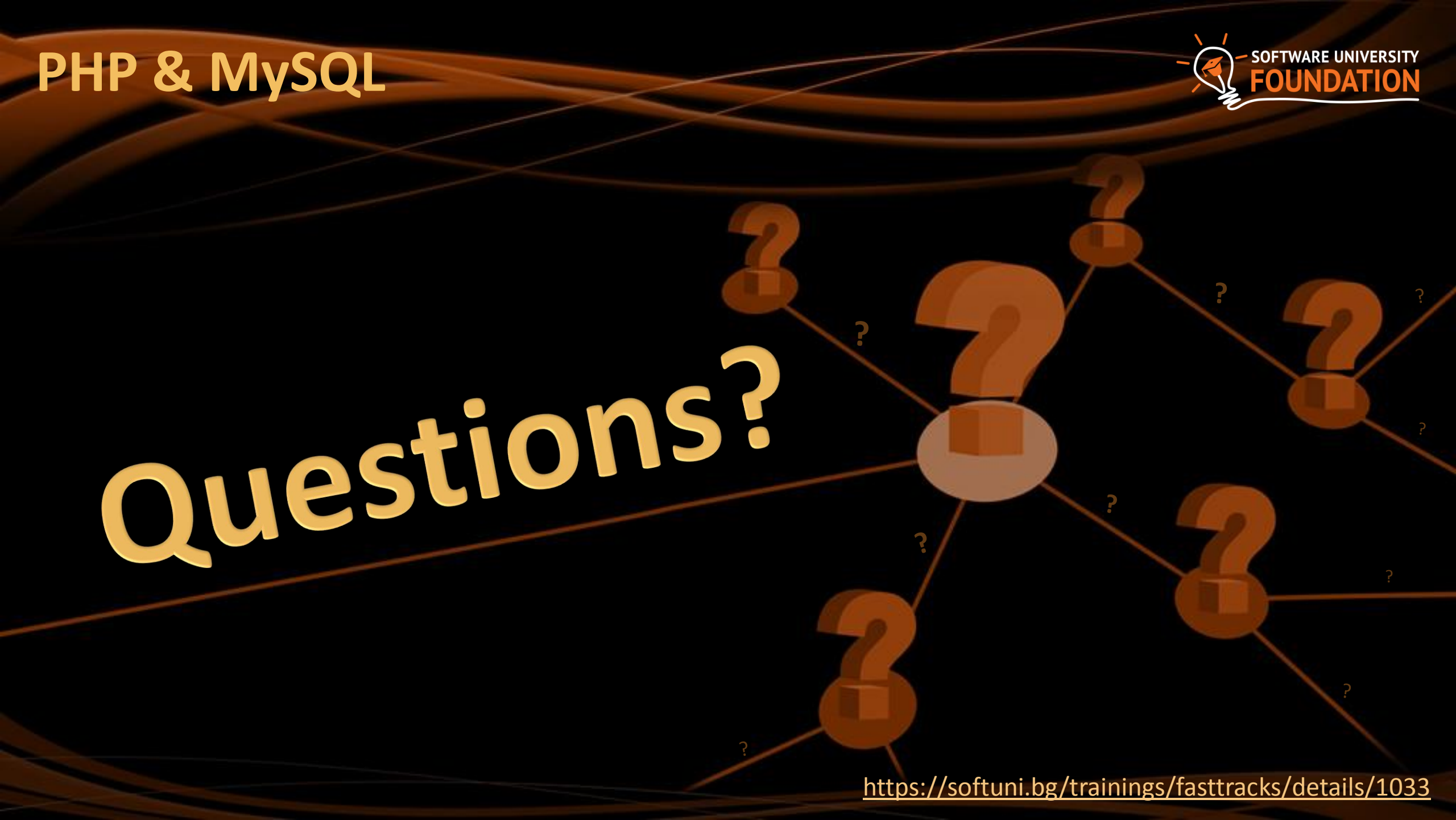
Summary

- OOP reproduces objects from the real world
- Serves for better code organization and data grouping
- Namespaces are like packages for different groups of code
- Prevent from code collision



PHP & MySQL

SOFTWARE UNIVERSITY FOUNDATION



https://softuni.bg/trainings/fasttracks/details/1033

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license

- Attribution: this work may contain portions from
  - "PHP Manual" by The PHP Group under CC-BY license
  - "PHP and MySQL Web Development" course by Telerik Academy under CC-BY-NC-SA license

Free Trainings @ Software University

- Software University Foundation – softuni.org
- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University @ YouTube
  - youtube.com/SoftwareUniversity
- Software University Forums – forum.softuni.bg