Week 4: OOP with PHP

FACULTY OF ENGINEERING, RUPP

COURSE: WEB AND CLOUD TECHNOLOGY PART II

INSTRUCTOR: TOUCH NGUONCHHAY

What Did You Know About OOP?

OOP

Object-oriented programming is a programming model based on the concept of "**objects**" contain data (fields or variables) or code (methods or procedures) to organizes software design.

OOP (con’t)

- OOP resolves around objects or data instead of logic and function.
- OOP focuses on what developers wants to manipulate rather than how.

Object (con’t)

- Object can be anything as an entity
  - Human being
  - Animal
  - Plant
  - Thing
  - Event
  - System or program
- Object may contain data (fields or variables) or code (methods or procedures)

Dog Object

Dog Object (con’t)

Data:

- name
- color
- breed

Function:

- bark()
- walk()
- eat()

Why OOP?

- Large (code reusability)
- Complex (scalability)
- Actively updated and maintained (effectively)

Complain About OOP

- Ignoring the computation and algorithm components
- Complexity of writing OOP code
- Time to compile

Alternative To OOP

- Functional programming
- Structured programming (modular programming)
- Imperative programming
- Declarative programming

Principles of OOP

- Encapsulation
- Inheritance
- Abstraction
- Polymorphism

Encapsulation

Encapsulation in OOP refers to binding the data and the methods to manipulate that data together in a single unit (class).

Encapsulation (con’t)

Dog class with following information:

- Data: name, color, breed
- Method: bark(), eat()

Encapsulation (con’t)

```
class Dog {
        private $name;
        private $color:
        private $breed;

        public function bark() {}
        public function eat() {}
}
```

Inheritance

Inheritance is the process of creating new classes, called derived classes, from base classes where an “is-a” relationship exists.

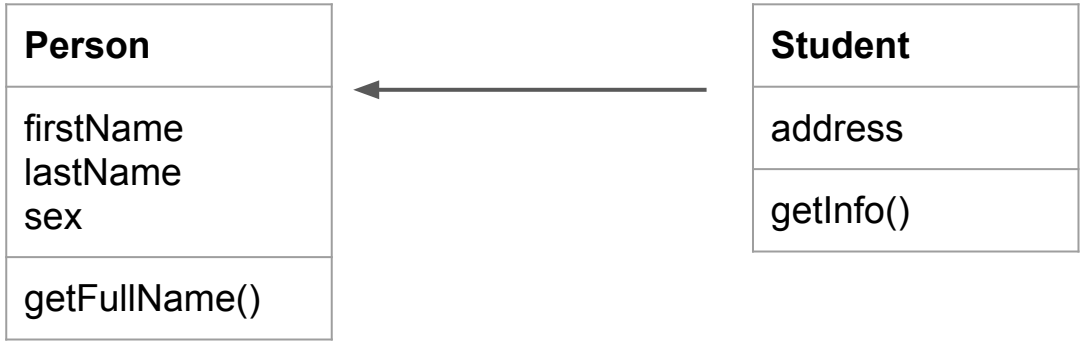
Inheritance (con’t)

- Base class, superclass, parent class
- Derived class, child class

Benefits of Inheritance

- Reusable code
- Reduce development time
- Better organize
- Easy to debug

Inheritance Example



Method Overloading

Method Overloading is a feature that allows a class to have more than one method having the same name.

How to make method overloading?

Method Overloading Rules

- Number of parameters
- Sequentials of parameters
- Data type parameters (with strict type)

Method Overloading?

1)

function sum ($a, $b): int {}

function sum ($a, $b, $c): int {}

2)

function mul ($a, $b): int {}

function mul ($a, $b): float {}

Abstraction

- Abstraction in OOP refers to showing only the essential features of an object to the user and hiding the other details to reduce complexity.
- Abstraction is enables the developer to implement more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden complexity.

Abstraction (con’t)

Payment abstraction:

- ABA API
- ACLEDA API
- WING API

Abstraction (con’t)

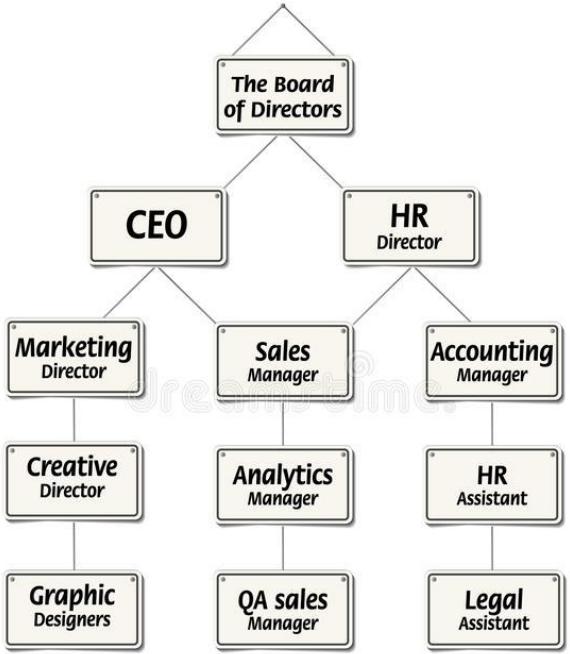
Notification abstraction:

- SMS
- Email
- Push notification

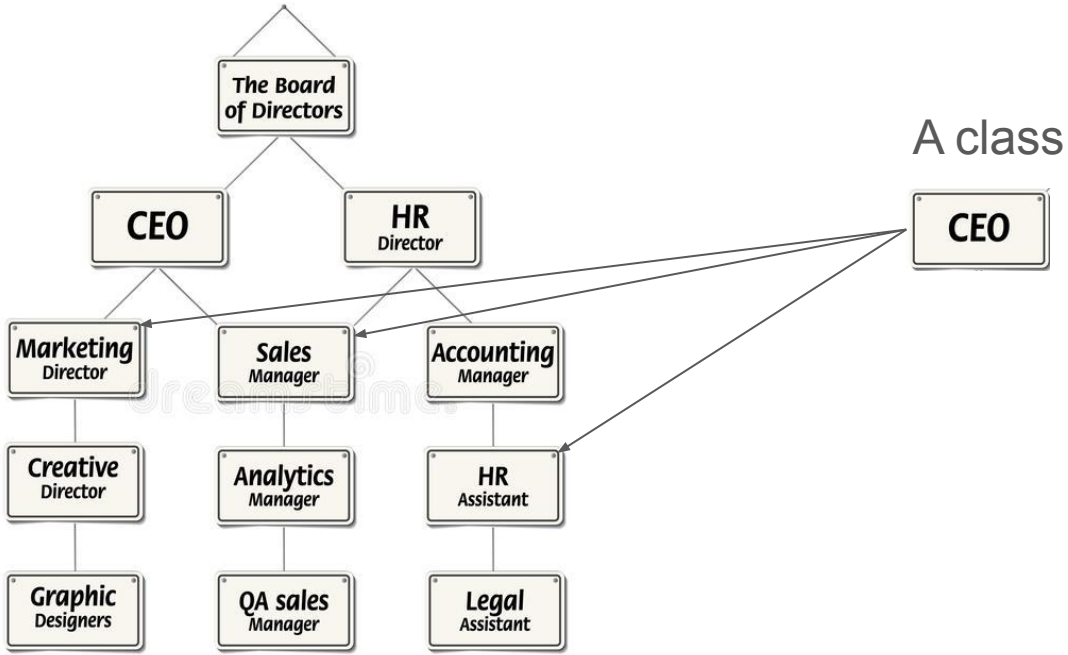
Polymorphism

Polymorphism is the ability of an object to take, represent, on many forms in the same threads.

Company Hierarchy



Company Hierarchy (con't)



Method Overriding

Method overriding is a language feature that allows a subclass to provide a specific implementation of a method that is already provided by one of its superclasses.

Method Overriding Rules

- Same name as in the super class
- Same parameter as in the parent class
- Implement IS-A relationship

IS-A vs HAS-A?

Implement OOP in PHP

- Create a class with **class** keyword
- Create object with **new** keyword
- Inherit a class or abstract class with **extends** keyword
- Create abstract class or method with **abstract** keyword
- Create an interface with **interface** keyword
- Inherit an interface with **implements** keyword

Create A Class And Object

```
<?php
class CLASS_NAME {
        access_modifier data_members;
        constructor() {}
        methods
}

$obj = new CLASS_NAME();
?>
```

Inherit From A Class

```
<?php

class A {
}

class B extends A {
}

?>
```

Create Abstract Class

```
<?php

abstract class A {
        [ abstract public function fun1(); ]
}

?>
```

Inherit From An Abstract Class

```
<?php

abstract class Shape {
	abstract function getArea();
}

class Triangle extends Shape {
	function getArea() {}
}

?>
```

Create An Interface

```
<?php

interface IPayment {
	function pay();
}

interface ShapeInterface {
	function getArea();
}
```

Class Inherits From Interface

```
<?php

interface IPayment {
	function pay();
}

class ABAPayment implements IPayment {
	function pay() {
		// statement
	}
}
```

Interface Inherits From Interface

```php
<?php

interface IPayment {
    function pay();
}

interface ITax {
    function getTax();
}

interface IPayment implements ITax {
    function pay() {};
}
```

Polymorphism with PHP

- Polymorphism with abstract class
- Polymorphism with interface

Abstract vs Interface?

Abstract Class?

- Share code among several closely related classes
- Share many common methods or fields or require access modifiers other than public (such as protected and private).
- Declare non-static or non-final fields.
- Implement a polymorphism

Interface?

- Expect that unrelated classes would implement your interface
- Specify the behavior of a particular data type, but not concerned about who implements its behavior
- Take advantage of multiple inheritances
- Implement a polymorphism

Traits in PHP

- A class can inherit only one superclass (OOP)
- Traits allows multiple inheritance in PHP
- A class inherits from trait with **use** keyword

Traits Syntax

```
<?php

trait Trait_Name {
        // statement
}
```

Inherit From Traits

```
<?php

class A {
	use Trait_Name_1, Trait_Name_2;
	// statement
}
```