CSIS 3280: Lecture 6

Announcements

- No lab today
- Project
  - Guideline document is uploaded
  - ~~Group~~ will be updated today
- Midterm is next week
  - Focus on Form, File IO, and Object-Oriented PHP
  - You will have (at least) two programming tasks ~ 1.5 hour
  - We may have a short lecture after the midterm

- Using **GLOBAL VARIABLE** is **BAD PROGRAMMING PRACTICE**
- It breaks the logic, bypassed the function scope and confuse yourself as the developer and moreover other people who read your code
- Similarly, other function or language construct that breaks the flow of loop like goto
- If (exist(Global variable) || exist(goto)) in MIDTERM

  ➔ ZERO MARK

The keyword instanceof

- Detects the class of object.
- More useful than gettype() (gettype will return $object) for all objects.
- Notice you are not comparing against a string value.

```
3    $manager = new Employee();
4    ...
5    if ($manager instanceof Employee) echo "Yes";
```

Type Hinting

- Type hinting is where the passed data is checked against a specific class (see above the parameter for takeLunchbreak is expecting an object of type “Employee”.

```
10    private function takeLunchbreak(Employee $employee) {
11        ...
12    }
```

- PHP also support return type declaration, which is particularly useful for abstract classes and interface. In the following, the colon after parenthesis signify the return type to be float

```
function sum($a, $b): float {
   return $a + $b;
}
```

Inheritance

- When a class extend another class, the property that belong to the parent may be accessible by the child (depends on the access modifier)
- When we instantiate a child class, by default it will call the __construct of the child
  - You can call the parent::__construct() if you want
- We can only extend from a single parent (concrete or abstract)
- But a child class can implement from multiple interfaces
- A class can also get multiple traits from different classes

Inheritance

```php
class Shape{

    // start with private to show the errors
    private $height;
    private $width;

    function setHeight($height){
        $this->height = $height;
    }

    function setWidth($width){
        $this->width = $width;
    }

    function getArea(){
        return $this->height * $this->width;
    }

    function getHeight(){
        return $this->height;
    }
    function getWidth(){
        return $this->width;
    }

}
```

```php
class Triangle extends Shape{
    private $type = "Triangle";

    function setType(){
        $this->type = "Triangle";
    }

    function getType(){
        return $this->type;
    }

    // override the parent method
    function getArea(){
        // will give a fatal error if height and width
        // in parent is not protected or public
        return 0.5 * $this->height * $this->width;
    }
}
```

- Class Triangle is child class of Shape
- The getArea() in Triangle class above will produce fatal error since it cannot access parent class's height and width

Abstract class

- PHP supports Abstract methods and classes
- Abstract classes can have concrete methods, but it needs to have at least one abstract method
- Abstract classes can have properties and implemented (concrete) methods
- The child class need to implement the abstract method
  - It needs to have the same function signature
  - It needs to have similar or lower visibility, e.g., protected in parent → implemented as protected or public
- The most common use-case for these is API implementations where a developer needs to know how to program an integrated or consumable class.
- A child class can only inherit one abstract class
- An abstract class is the foundation for another object.
  - Extending an abstract class is like completing the partial parent class

Abstract class

```php
// taken from W3schools
// Parent class
// abstract class can have properties and completely implemented methods
abstract class Car {
  public $name;
  public function __construct($name) {
    $this->name = $name;
  }
  abstract public function intro() : string;
}

// Child classes
class Audi extends Car {
  public function intro() : string {
    return "\nChoose German quality! I'm an $this->name!";
  }
}

class Citroen extends Car {
  public function intro() : string {
    return "\nFrench extravagance! I'm a $this->name!";
  }
}
```

Interface

- Interfaces allow you to create code which specifies which methods a class must implement
  - The interface is an agreement to have a specific set of public methods for your class.
- Use interface keyword rather than the class keyword and none of the methods have their contents defined.
- All methods is declared as public
- Interfaces cannot have any properties
- Interfaces can have constants but it cannot be overridden
- With interface, we can describe a set of functions and then hide the final implementation of those functions in an implementing class.
  - This allows you to change the IMPLEMENTATION of those functions without changing how you use it.
  - For example, we have a Database interface that describe some methods like addRecord() or RemoveRecord(). We can implement MySQLDatabase class or OracleDatabase from this interface. Whichever class we use, the methods are the same
- A child class can implement multiple interfaces

Interface

```php
interface Car {
  public function setModel($name);

  public function getModel();
}

interface Vehicle {
  public function setWheelCount($count);

  public function getWheelCount();
}
```

```php
class Sedan implements Car, Vehicle {
  private $model;
  private $wheelCount;

  public function setModel($name)
  {
    $this -> model = $name;
  }

  public function getModel()
  {
    return $this -> model;
  }

  public function setWheelCount($count)
  {
    $this -> wheelCount = $count;
  }

  public function getWheelCount()
  {
  return $this -> wheelCount;
  }
}
```

Traits

- Traits are used to declare methods that can be used in multiple classes.
- It is a way to implement code reuse where multiple classes implement the same functionality
- Traits can have methods and abstract methods that can be used in multiple classes, and the methods can have any access modifier (public, private, or protected).
- Traits are declared with the *trait* keyword
- In order to use the trait in a class, we need to use the keyword *use*

Trait

```php
trait Log {
    function printMessage($message) {
        printf("\nThe message is %s",$message);
    }
}
class A {
    use Log;
    function __construct() {
        $this->printMessage("Constructor A called");
    }

}

class B {
    function __construct() {
        $this->printMessage("Constructor B called");
    }
    use Log;
}

$a = new A();
$a = new B();
```

Namespace

- You may encounter a situation where you see two libraries (PHP include files) declaring identical class name
- For example, you have class Clean declared both in Library.inc.php and DataCleaner.inc.php → Fatal Error
- In order to avoid that, you need to declare namespace
  - Place at the top of Library.inc.php

```
namespace Library;
```

  - Place at the top of DataCleaner.inc.php

```
namespace DataCleaner;
```

- Then, you can use the Library's Clean as follow

```
require "Library.inc.php";
require "Data.inc.php";

use Library;
use DataCleaner;

// Instantiate the Library's Clean class
$filter = new Library\Clean();
```