CTIS 256 Web Technologies II

Notes # 7

Object Oriented PHP

Serkan GENÇ

Object-Oriented PHP

- Object oriented programming languages try to minimize **code repetition** for *simpler organization*, *easier maintenance*, *modularity* and *code reusability*.
- Organization : group similar objects having the same properties and behaviors under the same class.
- Maintenance: to make some updates, minimize code changes. Change only one line, and many places in the code change automatically.
- Modularity: easier to divide the whole code into manageable small pieces. Distributing tasks to developers is easy. It provides parallel construction, and thus increases productivity.
- Code Reusability: offers extending an existing class to add more functionality or change some part of it.

Encapsulation

- Encapsulation: pack all related properties and methods in a container.
- Access Modifiers: public, protected and private for Information Hiding.
- $this is a reference to newly created object in memory.
- -> arrow to access instance properties and methods.

```php
class User {

    // instance properties.
    protected $fullname, $gender ;

    // instance methods
    public function __construct($fullname, $gender)
    {
        $this->fullname = $fullname ;
        $this->gender = $gender ;
    }

    // instance method
    public function display() {
        return "Username: {$this->fullname} Gender: {$this->gender}" ;
    }
}
```

Class Constants

- To access a constant in a class: ClassName::Constant
- User::MALE outside of the Class, and self::MALE within Class.
- "::" is called scope operator. -> arrow operator is used only for instances.

```php
class User {

    // instance properties.
    protected $fullname, $gender ;

    // instance methods
    public function __construct($fullname, $gender)
    {
        $this->fullname = $fullname ;
        $this->gender = $gender ;
    }

    public function display() {
        return "Username: {$this->fullname}, Gender: "
            . ($this->gender === self::MALE ? "MALE" : "FEMALE" ) ;
    }

    // class constants
    const MALE = 1 ;
    const FEMALE = 2 ;
}
```

Class Properties and Methods

- Class properties are created per class not instance.
- **static** keyword is used to create class properties and methods.
- Class method can access only class properties since it does not have **$this**.

```php
class User {
    // class properties
    private static $count = 0 ;

    // class method
    public static function getCount() {
        return self::$count ;
    }

    // instance properties.
    protected $fullname, $gender ;

    // instance methods
    public function __construct($fullname, $gender)
    {
        $this->fullname = $fullname ;
        $this->gender = $gender ;

        self::$count++ ;
    }
```

Using Class

```php
// import class
require_once './App/Classes/User.php' ;

$p1 = new User("Mine Kaval", User::FEMALE) ;
$p2 = new User("Ali Gül", User::MALE) ;

// call instance methods
echo "<p>", $p1->display(), "</p>" ;
echo "<p>", $p2->display(), "</p>" ;

// call class method
echo "<p>", User::getCount() , "</p>" ;
```

Inheritance

- IS-A relation between User and Student. (User: parent, Student: child)
- Inheritance prevents manual copy/paste for common codes.
- Inreases code reuse, and improves maintanence.

```php
require_once "User.php" ;

class Student extends User {

  private $id ;

  public function __construct($fullname, $gender, $id)
  {
      parent::__construct($fullname, $gender) ;
      $this->id = $id ;
  }

  public function display() {
      $result = parent::display() ; // it returns fullname and gender string.
      return $result . "  ID : {$this->id}" ;
  }
}
```

Polymorphism

- PHP supports "duck typing" where there is no compile time type checking.
- Polymorphism is possible without inheritance.
- Polymorphism is very easy to implement in "duck type" languages.
- PHP Engines checks the type in run time.
- It may lead to run time error if the object does not have the method or property.
  - $list = [$p1, $p2, $p3, "hello world"] throws an exception since "hello world" does not have display() method.

```
//
// polymorphism
//
$list = [$p1, $p2, $std] ; // $list is a polymorphic array.

// what is the type of obj? it may be User or Student
// $obj is a polymorphic variable.
foreach( $list as $obj) {
    echo "<p>", $obj->display() , "</p>" ;
}
```

Namespace

- Prevents name collisions
- Alias for long name spaces.
- By default, all functions, classes and constants are in global space.
- "namespace" is the first statement in the file, and each file usually has a single namespace.

User.php

```php
<?php
  namespace App\Classes ;

  class User {
```

Student.php

```php
<?php
  namespace App\Classes ;

  require_once "User.php" ;

  class Student extends User {
```

index.php

```php
<?php

  require_once './App/Classes/User.php' ;
  require_once './App/Classes/Student.php' ;

  use App\Classes as A ;  // A is an alias to App\Classes namespace

  $p1 = new App\Classes\User("Mine Kaval", App\Classes\User::FEMALE) ;
  $p2 = new A\User("Ali Gül", A\User::MALE) ;
  $std = new A\Student("Özge Şener", A\User::FEMALE, 123456) ;

  echo "<p>", $p1->display(), "</p>" ;
  echo "<p>", $p2->display(), "</p>" ;
  echo "<p>", $std->display(), "</p>" ;

  echo "<p>", A\User::getCount() , "</p>" ;
```

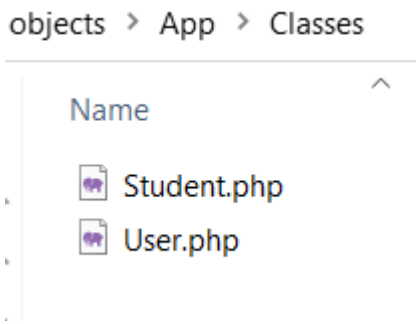
Auto Loading

- When php tries to create an object from a class, it needs the class definition. If it can not find it, it calls a globally registered ***function __autoload($classname)***, or you can register your own function name with ***spl_autoload_register('autoload');***
- If your php file requires many php classes, normally you should include each classes one by one. It takes many lines of code and it is not easy to manage.
- The solution is autoloading. If you organize your classes in a folder with a format, you can take advantage of __autoload functionality.
- Check out the example at the right side.
- **PSR-4** (PHP Standards Recommendations) is a specification for autoloading classes from their file paths.

Folder structure of classes:

Root: c:/wamp/objects

Class: {root}/App/Classes



Automatic way to class loading

```php
// Autoloading
spl_autoload_register(function($class){
    //echo "./{$class}.php" ;
    require_once "./{$class}.php" ; // load the class.
}) ;

 //require_once ".\App\Classes\user.php" ;
 //require_once ".\App\Classes\student.php" ;

use \App\Classes as A ;

$p1 = new A\User("Ali Gül", A\User::MALE) ;
$p2 = new A\User("Mine Gül", A\User::FEMALE) ;
$std1 = new A\Student("Özge Kaya", A\User::FEMALE, 1234) ;
```

JSON

- **J**ava**S**cript **O**bject **N**otation
- Lightweight Text Format for Storing and Exchanging Data
- Language Independent

```
{
    "name" : "Ali",
    "id" : 12345,
    "emails" : [ "ali@hotmail.com", "ali@gmail.com"],
    "address" : {
        "street" : "Lizbon cd.",
        "city" : "Ankara"
    }
}
```

- JSON object
  - started by "{"
  - end with "}"
- "property" : "value"
- separated by comma ","
- value can be
  - string
  - number
  - json object
  - array
  - Boolean
- Use double quotes for properties and strings, ***no single quotes***.

JSON decode

json_decode($str [, flag]) : Parses JSON string to PHP Array or Object.
flag : true converts to PHP Array.

```php
$inputText = '{
    "name" : "Ali",
    "id" : 12345,
    "emails" : [ "ali@hotmail.com", "ali@gmail.com"],
    "address" : {
        "street" : "Lizbon cd.",
        "city" : "Ankara"
    }
}' ;

// converts json string to php OBJECT
$person = json_decode($inputText) ;
echo "<p>", $person->name , " First Email: ", $person->emails[0], "</p>";

// converts json string to php ASSOCIATIVE ARRAY
$person = json_decode($inputText, true) ;
echo "<p>", $person["name"] , " First Email: ", $person["emails"][0], "</p>";
```

JSON encode

json_encode($array) : string : Converts PHP Array to JSON String

```php
// PHP Array
$person = [
    "name" => "Ali",
    "id" => 12345,
    "emails" => [ "ali@hotmail.com", "ali@gmail.com"],
    "address" => [
        "street" => "Lizbon cd.",
        "city" => "Ankara"
    ]
] ;

// Convert PHP Array to JSON String
$jsonString = json_encode($person) ;
echo $jsonString ;
```

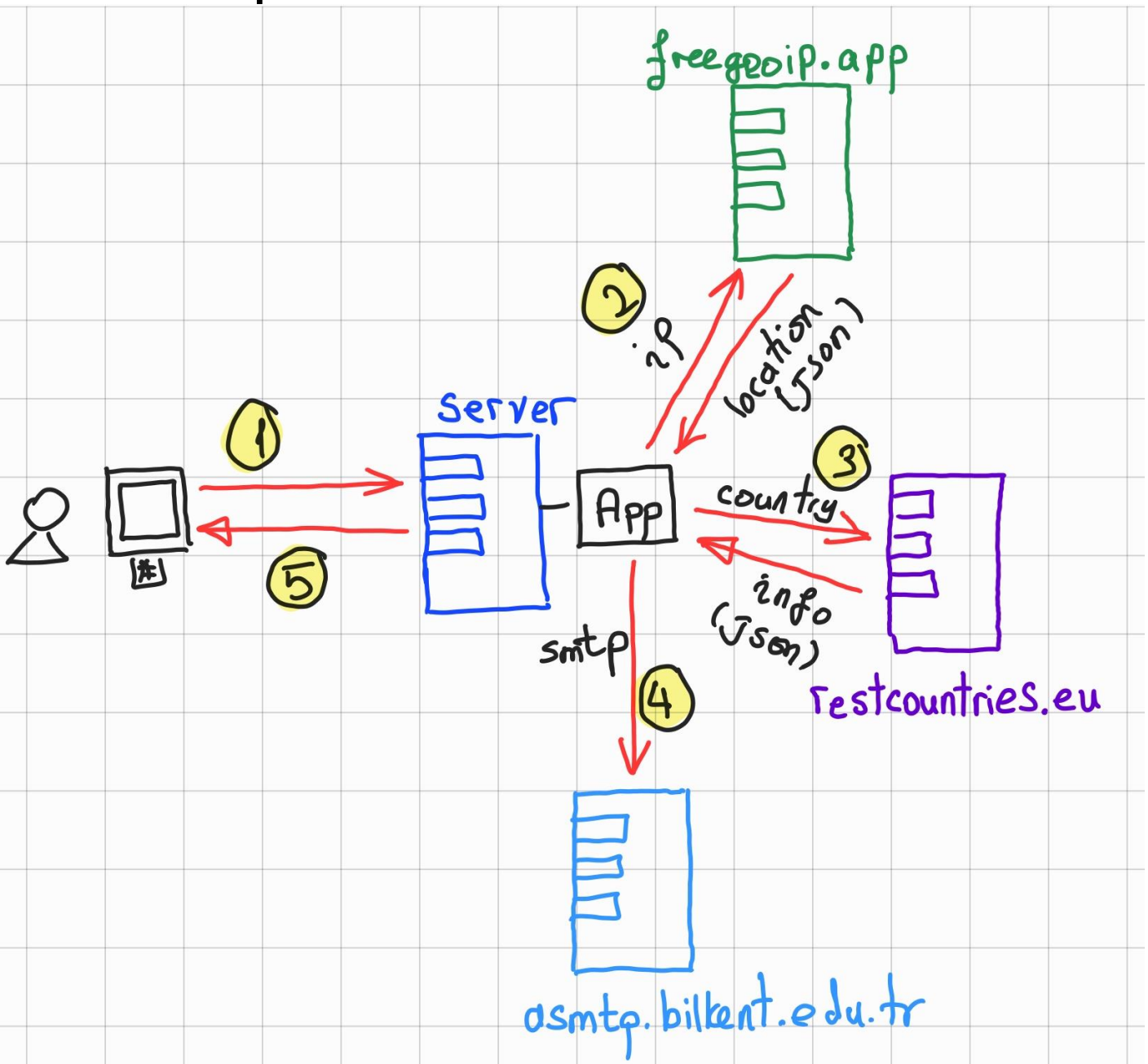
JSON String

{"name":"Ali","id":12345,"emails":["ali@hotmail.com","ali@gmail.com"],"address":{"street":"Lizbon cd.","city":"Ankara"}}

Composer

- Dependency Package Manager for PHP
- It downloads third party packages from the repository "packagist.org"
  - dependency resolution for PHP packages
- Initialize Composer
  - composer init
  - composer.json is the configuration file
- Install a Package
  - composer require package_name
  - composer require guzzlehttp/guzzle
- Update Package
  - composer update
  - composer update package_name
- Autoload classes in packages
  - **require_once** 'vendor/autoload.php'

Example



Libraries :

- HTTP Client Library
  - GuzzleHttp
  - Send HTTP Request
  - Receive in json
- SMTP Library
  - Phpmailer
  - To send email

Web Services :

- IP to Location (Country, City, Long/Lat)
  - https://freegeoip.app/json/{ip/domain}
- Detail Information about a Country
  - https://restcountries.eu/rest/v2/name/
- SMTP Service
  - asmtp.bilkent.edu.tr

Example for an HTTP Client

Install Guzzle HTTP Client Library

```
composer require guzzlehttp/guzzle
```

PHP Code to use HTTP Client library

```php
require_once 'vendor/autoload.php' ;

$ip = "139.179.10.13" ;

$http = new \GuzzleHttp\Client(["verify" => false]);

$response = $http->request("GET", "https://freegeoip.app/json/$ip") ;

$ipInfo = json_decode($response->getBody()->getContents()) ;

echo "<p>Country : " , $ipInfo->country_name , "</p>" ;
```

Example for an SMTP Client

Install phpmailer

```
composer require phpmailer/phpmailer
```

PHP Sample Code to use SMTP Client library

```php
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\SMTP;
use PHPMailer\PHPMailer\Exception;

$mail = new PHPMailer(true);
$mail->isSMTP(); //Send using SMTP
$mail->Host = 'asmtp.bilkent.edu.tr'; //Set the SMTP server to send through
$mail->SMTPAuth = true; //Enable SMTP authentication
$mail->Username = "username@bilkent.edu.tr"; //SMTP username
$mail->Password   = "password"; //SMTP password
$mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS; //Enable TLS encryption;
$mail->Port       = 587;  //TCP port to connect to
//Recipients
$mail->setFrom('username@bilkent.edu.tr', 'CTIS256 APP');
$mail->addAddress('anyuser@gmail.com', 'Username'); //Add a recipient

//Content
$mail->isHTML(true); //Set email format to HTML
$mail->Subject = 'Email from CTIS256';
$mail->Body    = "<b>Hello There!!</b><p>Html content here</p>";
$mail->send();
```