Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

3) PHP an object-oriented language

Emmanuel Benoist

Spring Term 2017

Computer Science Division

Table of Contents

Object Oriented Programming

Object Oriented Programming I

- **The concepts of OO-Programming are very similar to the ones used in Java**
  - Class
  - Abstract class
  - Interface
- **Heritage**
  - a class can extend only one class (including abstract class)
  - a class can implement many interfaces

Object Oriented Programming II

```
class ClassA{
  var $val1;
  function __construct($v){
    $this->val1=$v;
  }
  function f1(){
    echo $this->val1;
  }
}
$a1 = new ClassA(10);
$a1->f1(); // Output: 10
echo $a1->val1; // Output: 10
```

Heritage I

```
class A{
   var $valA;
 function mult($arg){
    return $arg * $this->valA;
 }
}
class B extends A{
   var $valB;
 function add(){
    return $this->valA + $this->valB;
 }
}

$b = new B();
$b->valA = 5;
```

Heritage II

```
$b->valB = 10;
echo "result = ".$b->mult(5)
```

Abstract class

- **An abstract class is a class that have at least one abstract method**
  - An abstract method contains just the signature
  - An abstract class can not have any instance

```
abstract class A{
   var $valA;
   abstract function add();
}
class B extends A{
   var $valB;
 function add(){
    return $this->valA + $this->valB;
 }
}
$b = new B();
$b->valA = 5;
$b->valB = 10;
echo "result = ".$b->add()
```

Interface I

- **Defines the functions that must be implemented**
  - Each class implementing the interface must implement all the methods
  - Otherwise the class is abstract
  - One class can implement many interfaces
- **Usage**
  - An interface is a *"contract"*
  - You define an interface, to have the possibility to change the class
  - You just use an instance of a interface, without knowing the real implementation.
  - See Factory, Builder, Adapter, Decorator, Composite, Command, Proxy, . . . design patterns for instance

Interface I

```
interface Stack{
  function push($item);
  function pop();
  function size();
}
class MyStack implements Stack{
  var $stackArray = array();
  function push($item){
    $this->stackArray[]=$item;
  }
  function pop(){
     return array_pop($this->stackArray);
  }
  function size(){
    return count($this->stackArray);
  }
}

$stack = new MyStack();
```

Interface II

```
$stack->push("A");
$stack->push("B");
$stack->push("C");
$stack->push("D");
while($stack->size()>0){
   echo $stack->pop().",";
}
/* Output :
D,C,B,A,
*/
```

Type Hinting

- **You can "test" the type of an element during execution**
  - Unlike java, the test is done while the program runs
  - Two possiblities `intanceOf()` tests the

```
if($var instanceOf MyClass){
  ...
}
if($var instanceOf MyInterface){
  ...
}

// the function expects an argument from type
// MyType (can be a class or an interface)
function myFunction(MyType $var){
   ...
}
```

Access modifiers

PHP5 has introduced 3 access modifiers to control the visibility of the attributes and methods. These modifiers are :

- `public`, which mean that an attribute or a method declared as public may be accessed from inside and outside of the class. For compatibility reason with the earlier version of PHP, this modifier is the default if none is specified.
- `private`, which mean that this entity may only be accessed from inside the class. All attributes and methods whose are only used within the class should be declared `private`
- `protected`, which mean that the entity may only be accessed from within the class and also from any subclass of the current class.

Access modifiers (Example)

These modifiers are declared in front of the attributes or methods name. For example :

```
class ClassName{
    private $attribute;
    public function getAttribute() {
        return $this->attribute;
    }
    public function setAttribute ($value){
        $this->attribute = $value;
    }
}
$g=new ClassName();
$g->setAttribute(10);
echo $g->getAttribute();
```

Forms and Cookies

Forms

- **HTML uses forms to send information from the client to the server**
  - Form is defined by: Method and action
- **Method=GET**
  - Information is encoded in the URL
  - Information must be short
  - Information can be cached
- **Method=POST**
  - Informtion is in the HTTP Body of the request (i.e. not in the URL)
  - It can be much larger
  - It can not be cached

```
<form action="test.php" method="GET">
  <input type=''text'' name="var1">
  <input type=''submit'' value="submit">
</form>
```

Read forms inputs I

- **PHP parses and decodes the input**
- **The inputs are stored inside an array**
  - For the GET method: `$_GET`
  - For the POST method: `$_POST`

```
<?php
$textSent = $_GET["var1"];
...
?>
```

Example: a multiplicator I

```
<?php
if(isset($_GET['add'])){
 $x = $_GET['x'];
 $y = $_GET['y'];
 echo "$x plus $y = ".($x+$y);
}
if(isset($_GET['mult'])){
 $x = $_GET['x'];
 $y = $_GET['y'];
 echo "$x times $y = ".($x*$y);
}
?>
<form action="multiplier-get.php" method="GET">
x=<input type="text" name="x"><br>
y=<input type="text" name="y"><br>
<input type="submit" value="Add" name="add"><br>
<input type="submit" value="Multiply" name="mult">
</form>
```

Example (POST): a multiplicator I

```
<?php
if(isset($_POST['add'])){
 $x = $_POST['x'];
 $y = $_POST['y'];
 echo "$x plus $y = ".($x+$y);
}
if(isset($_POST['mult'])){
 $x = $_POST['x'];
 $y = $_POST['y'];
 echo "$x times $y = ".($x*$y);
}
?>
<form action="multiplier-post.php" method="POST">
x=<input type="text" name="x"><br>
y=<input type="text" name="y"><br>
<input type="submit" value="Add" name="add"><br>
<input type="submit" value="Multiply" name="mult">
</form>
```

Cookies

Cookies

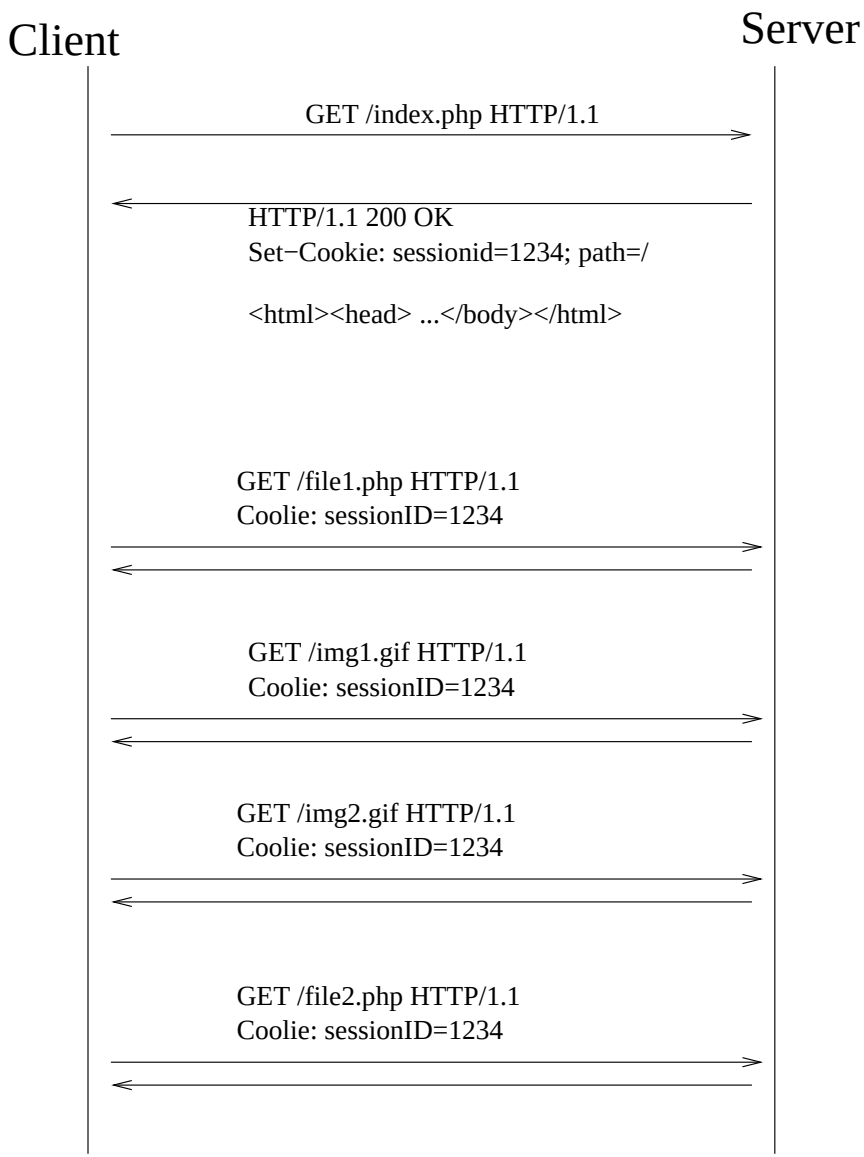
Principles

- Very small informations
- Sent once by the server to the client
- Resent within each request by the client to the server

Usage

- For holding configuration from the user
- Example: language, Number of visits, ...
- Now: used only to store session IDs

Cookies (Cont.)



Working with cookies:

- PHP has the ability to work with cookies. For that goal, it provides different functions to create, manage and destroy cookies.
- **Setting cookies**
  To create cookies, one should call the function `setcookie`.
  For example :

  setcookies ('aroma', 'chocolate');

- Since cookies are sent within HTTP headers, they must be sent *before* any message body.
  The `setcookie()` functions may take up to 5 arguments.
  Only the first two are mandatory.

Working with cookies:

The generic syntax of `setcookie()` is :

setcookie(<cookie name>, <cookie value>, <expiration date>, <path>, <host or domain>, <secure>);

The meaning of these attributes is :

1. `cookie name` : name of the attribute stored into the cookie;
2. `cookie value` : value of the attribute;
3. `expiration date` : The "time to live" of the cookie (unix timestamp).
4. `path of requesting pages` : only web pages whose path starts with the specified path can request the cookie;
5. `host or domain` : only hosts belonging to the specified domain can request the cookie;
6. `secure` : if set to 1, only client connected through a secure SSL connection can request the cookie.

Examples of cookies

```
setcookie("username", "BIE1", time()+86400,
          "/WebApps/",
          ".benoist.ch", "1");
```

- The cookie name is `username`
- The cookie value is `Emmanuel`
- The cookie will expire one day after his creation
- This cookie may be only required from pages in the `/WebApps/` hierarchy;
- This cookie is only resent to my servers.
- Since this cookie contains a username (may be a sensitive information), it will be delivered only to servers connected through a secure connection.

Requesting a cookie

Requesting a cookie is just like testing if a variable is set. Therefore, the function to test cookies is the same as the function for testing variables.

boolean isset($_COOKIE[<cookie name>])

- All cookies of the browser are available through the array `$_COOKIE`;
- When a browser returns a cookie, it only sends the cookie name and values.

Requesting a cookie (Cont.)

Example:

```
if (isset($_COOKIE["username"])) {
    echo "Your username is" . $_COOKIE["username"];
}
```

All available cookies of a browser may be requested by the following code:

```
foreach ($_COOKIE as $cookie_name => $cookie_value) {
    echo "Name = $cookie_name ; Value = $cookie_value<br↘
    →>\n";
}
```

Deleting a cookie

It is possible to delete a cookie which is no longer used. To delete a cookie, it must be resent with exactly the same parameter, except the `time to live` parameter which must be set in the paste.
Example

setcookie("username", "", time()−86400);

Session Management

Create a session

Session management

- PHP provides tools to follow a user during a session.
- To create a session, one has to call the function `session_start();`
- All information of the session are available through the global array `$_SESSION;`

Example:

```
session_start();
$_SESSION["visit"]++;
echo 'You were here' . $_SESSION['visit'] . ' times';
```

Session management (Cont.)

- PHP try to send a cookie with the session ID;
- If the client does not allow cookies, PHP adds the session ID to all URLs that will be sent. Example:
  - with a client accepting cookies :

    ```
    echo '<a href="train.php">Take the train</a>';
    ```

  - With a client refusing cookies :

    ```
    echo '<a href="train.php?PHPSESSID=2eb89f33445">'
         .'Take the train</a>';
    ```

  - In this example, the session name is `PHPSESSID` and the session id is `2eb89f33445`

Session Security

- The session module cannot guarantee that the information you store in a session is only viewed by the user who created the session. You need to take additional measures to actively protect the integrity of the session, depending on the value associated with it.
- Assess the importance of the data carried by your sessions and deploy additional protections – this usually comes at a price, reduced convenience for the user. For example, if you want to protect users from simple social engineering tactics, you need to enable `session.use_only_cookies`. In that case, cookies must be enabled unconditionally on the user side, or sessions will not work.

Example

Example of use of session

```
<?php
// page1.php
session_start();
echo 'Welcome to page #1';
$_SESSION['favcolor'] = 'green';
$_SESSION['animal'] = 'cat';
$_SESSION['time'] = time();

echo '<br /><a href="page2.php">page 2</a>';
?>
```

Example of use of session (Cont.)

```
<?php
// page2.php
session_start();
echo 'Welcome to page #2<br />';
echo $_SESSION['favcolor']; // green
echo $_SESSION['animal']; // cat
echo date('Y m d H:i:s', $_SESSION['time']);
echo '<br /><a href="page1.php">page 1</a>';
?>
```

Include files

require()

- **The `require()` statement replaces itself with the specified file, much like the C preprocessor's `#include` works.**
  `require()` and `require_once()` produce a fault when the resource is not available.

```
// imports all the functions and classes
require ("file.php");

// This file is loaded only once (even if required many times→)
require_once("test.php");
```

- **The `include()` statement includes and evaluates the specified file.**
  Almost the same without creating an error.

Conclusion I

- **PHP is similar to Java**
  - Basic syntax (if, while, for, . . .)
  - Object Oriented design (classes, interfaces, abstract classes)
- **It is also very different from Java**
  - Variables are loosly typed
  - Objects are per default open
  - No type verification before execution!
- **Extensive tests are needed**
  - Since less testing is done durring compilation (compared to Java)