Object-Oriented PHP

Developing Object-Oriented PHP

- Topics:
  - OOP concepts – overview, throughout the chapter
  - Defining and using objects
    - Defining and instantiating classes
    - Defining and using variables, constants, and operations
    - Getters and setters
  - Defining and using inheritance and polymorphism
    - Building subclasses and overriding operations
    - Using interfaces
  - Advanced object-oriented functionality in PHP
    - Comparing objects, Printing objects,
    - Type hinting, Cloning objects,
    - Overloading methods, (some sections WILL NOT BE COVERED!!!)

Object-Oriented Programming

- Object-oriented programming (OOP) refers to the creation of reusable software object-types / classes that can be efficiently developed and easily incorporated into multiple programs.
- In OOP an object represents an entity in the real world (a student, a desk, a button, a file, a text input area, a loan, a web page, a shopping cart).
- An OOP program = a collection of objects that interact to solve a task / problem.

Object-Oriented Programming

- Objects are self-contained, with data and operations that pertain to them assembled into a single entity.
  - In *procedural programming* data and operations are separate → this methodology requires sending data to methods!
- Objects have:
  - Identity; ex: 2 “OK” buttons, same attributes → separate handle vars
  - State → a set of attributes (aka member variables, properties, data fields) = properties or variables that relate to / describe the object, with their current values.
  - Behavior → a set of operations (aka methods) = actions or functions that the object can perform to modify itself – its state, or perform for some external effect / result.

Object-Oriented Programming

- Encapsulation (aka data hiding) central in OOP
  - = access to data within an object is available only via the object's operations (= known as the interface of the object)
  - = internal aspects of objects are hidden, wrapped as a birthday present is wrapped by colorful paper ☺
- Advantages:
  - objects can be used as black-boxes, if their interface is known;
  - implementation of an interface can be changed without a cascading effect to other parts of the project → if the interface doesn't change

Object-Oriented Programming

- Classes are constructs that define objects of the same type.

  A class is a template or blueprint that defines what an object's data and methods will be.

  Objects of a class have:

  - Same operations, behaving the same way
  - Same attributes representing the same features, but values of those attributes (= state) can vary from object to object

- An object is an instance of a class.

  (terms objects and instances are used interchangeably)

- Any number of instances of a class can be created.

OOP in Web Programming

- Small Web projects
  - Consist of web scripts designed and written using an *ad-hoc* approach; a function-oriented, procedural methodology
- Large Web software projects
  - Need a properly thought-out development methodology – OOP →
  - OO approach can help manage project complexity, increase code reusability, reduce costs.
  - OO analysis and design process = decide what object types, what hidden data/operations and wrapper operations for each object type
  - UML – as tool in OO design, to allow to describe classes and class relationships

Creating Classes in PHP

- A minimal class definition:

```
class classname {  // classname is a PHP identifier!
    //  the class body = data & function member definitions
}
```

- Attributes
  - are declared as variables within the class definition using keywords that match their visibility: public, private, or protected.

    (Recall that PHP doesn't otherwise have declarations of variables → data member declarations against the nature of PHP?)
- Operations
  - are created by declaring functions within the class definition.

Creating Classes in PHP

- Constructor = function used to create an object of the class
  - Declared as a function with a special name:

    `function __construct (param_list) { … }`

  - Usually performs initialization tasks: e.g. sets attributes to appropriate starting values
  - Called automatically when an object is created
  - A default no-argument constructor is provided by the compiler only if a constructor function is not explicitly declared in the class
  - Cannot be overloaded (= 2+ constructors for a class); if you need a variable # of parameters, use flexible parameter lists…

Creating Classes in PHP

- Destructor = opposite of constructor
  - Declared as a function with a special name, cannot take parameters

    ```
    function __destruct () { … }
    ```
  - Allows some functionality that will be automatically executed just before an object is destroyed
    - An object is removed when there is no reference variable/handle left to it
    - Usually during the "script shutdown phase", which is typically right before the execution of the PHP script finishes
  - A default destructor provided by the compiler only if a destructor function is not explicitly declared in the class

Instantiating Classes

- Create an object of a class = a particular individual that is a member of the class by using the `new` keyword:

```
$newClassVariable = new ClassName(actual_param_list);
```

- Notes:
- Scope for PHP classes is global (program script level), as it is for functions
- Class names are case insensitive as are functions
- PHP 5 allows you to define multiple classes in a single program script
- The PHP parser reads classes into memory immediately after functions ⇒ class construction does not fail because a class is not previously defined in the program scope.

Using Data/Method Members

- From operations *within* the class, class's data / methods can be accessed / called by using:
  - `$this` = a variable that refers to the current instance of the class, and can be used only in the definition of the class, including the constructor & destructor
  - The pointer operator `->` (similar to Java's object member access operator ".")
  -

```
class Test {
    public $attribute;
    function f ($val) {
        $this -> attribute = $val;  // $this is mandatory!
    }                               // if omitted, $attribute is treated
}                                   // as a local var in the function
```

No $ sign here

Using Data/Method Members

- From *outside* the class, accessible (as determined by access modifiers) data and methods are accessed through a variable holding an instance of the class, by using the same pointer operator.

```
class Test {
  public $attribute;
}
$t = new Test();
$t->attribute = “value”;
echo $t->attribute;
```

Defining and Using Variables, Constants and Functions

- Three access / visibility modifiers introduced in PHP 5, which affect the scope of access to class variables and functions:
  - public : public class variables and functions can be accessed from inside and outside the class
  - protected : hides a variable or function from direct external class access + protected members are available in subclasses
  - private : hides a variable or function from direct external class access + protected members are hidden (NOT available) from all subclasses
- An access modifier has to be provided for each class instance variable
- Static class variables and functions can be declared without an access modifier → default is public

Getters and Setters

- Encapsulation : hide attributes from direct access from outside a class and provide controlled access through accessor and mutator functions
  - You can write custom getVariable() / setVariable($var) functions *or*
  - Overload the functionality with the __get() and __set() functions in PHP
- __get() and __set()
  - Prototype:
     mixed __get($var);
    // param represents the name of an attribute, __get returns the value of that attribute
    void __set($var, $value);
    // params are the name of an attribute and the value to set it to

Getters and Setters

- __get() and __set()
  - Can only be used for non-static attributes!
  - You do not directly call these functions;

    For an instance $acc of the BankAccount class:

    $acc->Balance = 1000;

    implicitly calls the __set() function with the value of $name set to 'Balance', and the value of $value set to 1000.

    (__get() works in a similar way)

Getters and Setters

- __get() and __set() functions’ value: a single access point to an attribute ensures complete control over:
  - attribute’s values

```
function __set($name, $value) {
  echo "<p>Setter for $name called!</p>";
  if (strcasecmp($name, "Balance")==0 && ($value>=0))
        $this->$name = $value;
  ...
}
```

  - underlying implementation: as a variable, retrieved from a db when needed, a value inferred based on the values of other attributes

    → transparent for clients as long as the accessor / mutator functions’ contract doesn’t change.

Designing Classes

- Classes in Web development:
  - Pages
  - User-interface components
  - Shopping carts
  - Product categories
  - Customers
- TLA Consulting example revisited - a Page class, goals:
  - A consistent look and feel across the pages of the website
  - Limit the amount of HTML needed to create a new page: easily generate common parts, describe only uncommon parts
  - Easy maintainable when changes in the common parts
  - Flexible enough: ex. allow proper navigation elements in each page

Class Page

- Attributes:
  - $content → content of the page, a combination of HTML and text
  - $title → page's title, with a default title to avoid blank titles
  - $keywords → a list of keywords, to be used by search engines
  - $navigation → an associative array with keys the text for the buttons and the value the URL of the target page
- Operations:
  - __set()
  - Display() → to display a page of HTML, calls other functions to display parts of the page:
  - DisplayTitle(), DisplayKeywords(), DisplayStyles(), DisplayHeader(), DisplayMenu(), DisplayFooter() → can be overridden in a possible subclass