PHP 5 and the new OO features for enterprise solutions, part 1

Marcus Börger

international **PHP**2003 conference

Overview

- ☑ PHP5 vs PHP4
- ☑ Is PHP5 revolutionary?
- ☑ PHP 5 OO
  - ☑ Why is OO a good thing?

Revamped OO Model

- ☑ PHP5 has really good OO
  - ☑ Better code reuse
  - ☑ Better for team development
  - ☑ Easier to refactor
  - ☑ Some patterns lead to much more efficient code
  - ☑ Fits better in marketing scenarios

PHP 4 and OO ?

- Poor Object model
  - ☑ Methods
    - ☒ No visibility
    - ☒ No abstracts, No final
    - ☒ Static without declaration
  - ☑ Properties
    - ☒ No default values
    - ☒ No static properties
  - ☑ Inheritance
    - ☒ No abstract, final inheritance, no interfaces
  - ☑ Object handling
    - ☒ Copied by value
    - ☒ No destructors

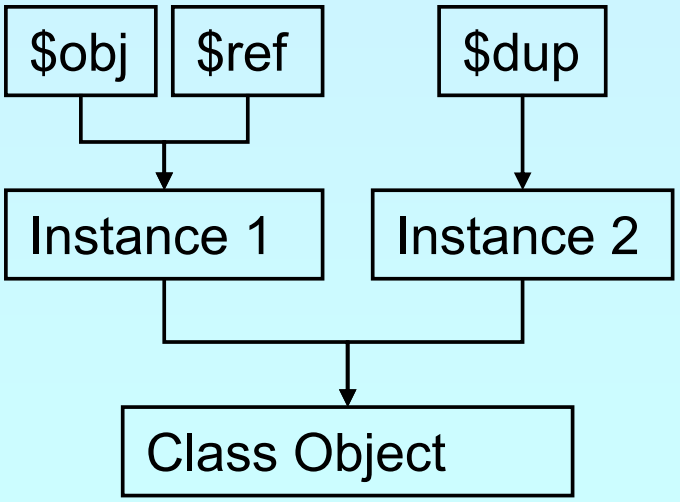
ZE2's revamped object model

- ☑ Objects are referenced by identifiers
- ☑ Constructors and Destructors
- ☑ Static members
- ☑ Default property values
- ☑ Constants
- ☑ Visibility
- ☑ Interfaces
- ☑ Final and abstract members
- ☑ Interceptors
- ☑ Exceptions
- ☑ Reflection API
- ☑ Iterators

Objects referenced by identifiers

- ☑ Objects are no longer copied by default
- ☑ Objects may be copied using __clone()

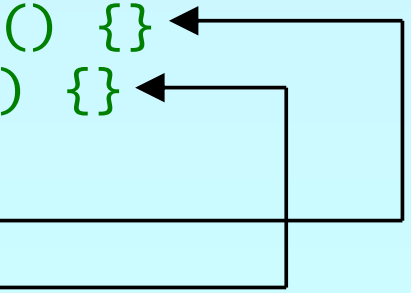
```
<?php

class Object {};

$obj = new Object();

$ref = $obj;

$dup = $obj->__clone();

?>
```



Constructors and Destructors

- Constructors/Destructors control object lifetime
  - Constructors may have both new OR old style names
  - Destructors are called when deleting last reference

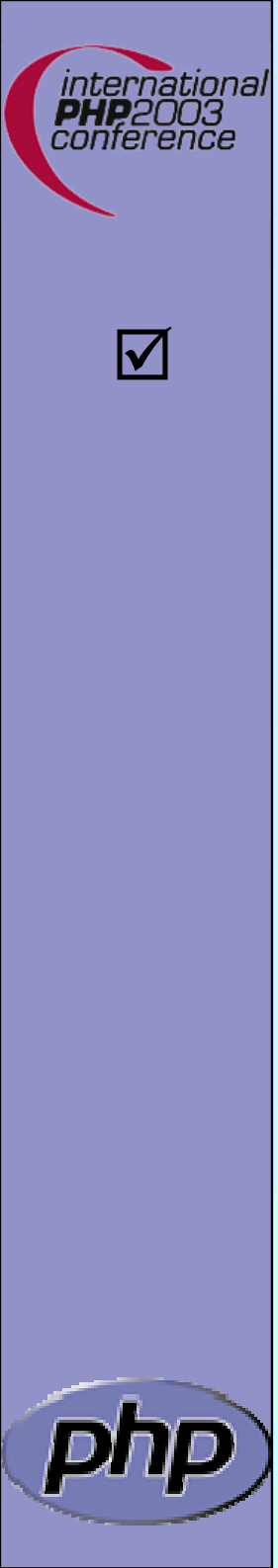
```
<?php

class Object {
  function __construct() {}
  function __destruct() {}
}
$obj = new Object();
unset($obj);

?>
```



Constructors and Destructors

- ☑ Parents must be called manually

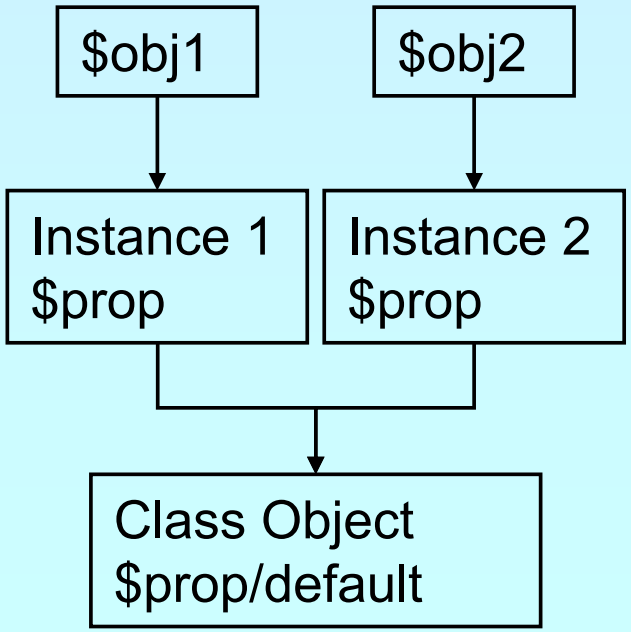
```
<?php
class Base {
   function __construct() {}
   function __destruct() {}
}
class Object extends Base {
   function __construct() {
      parent::__construct();
   }
   function __destruct() {
      parent::__destruct();
   }
}
$obj = new Object();
unset($obj);
?>
```



Default property values

- Properties can have default values
  - Bound to the class not to the object
  - Default values cannot be changed but overwritten

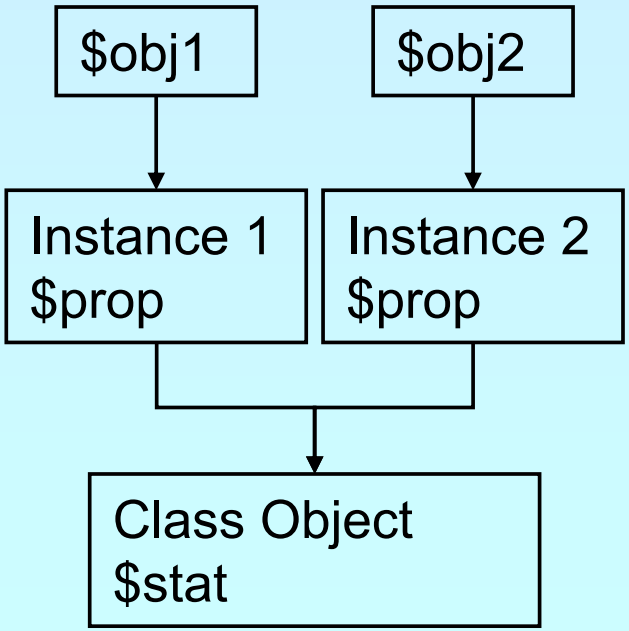
```
<?php

class Object {
  var $prop = "Hello\n";
}

$obj1 = new Object;
$obj1->prop = "Hello World\n";

$obj2 = new Object;
echo $obj2->prop; // Hello

?>
```



Static members

- Static methods and properties
  - Bound to the class not to the object
  - Can be initialized

```php
<?php
class Object {
  var $pop;
  static $stat = "Hello\n";
  static function test() {
    echo self::$stat;
  }
}
Object::test();
$obj1 = new Object;
$obj2 = new Object;
?>
```



New pseudo constants

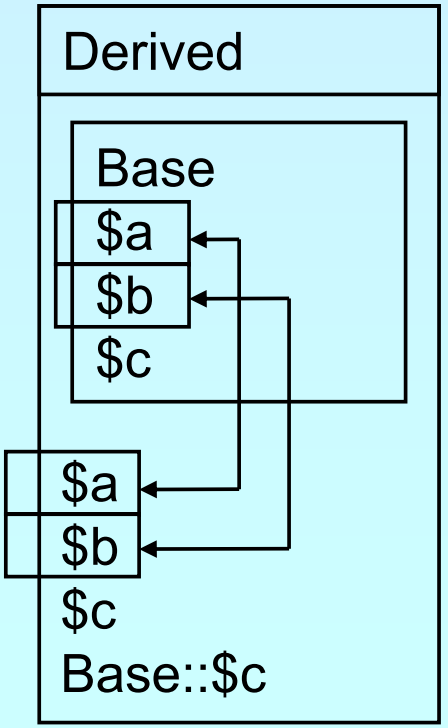
- ☑ __CLASS__ shows the current class name
- ☑ __METHOD__ shows class and method or function
- ☑ Self references the class itself
- ☑ Parent references the parent class
- ☑ $this references the object itself

```php
<?php
class Base {
    static function Show() {
        echo __FILE__.'('.__LINE__.'):'.__METHOD__."\n";
    }
}
class Object extends Base {
    static function Use() {
        Self::Show();
        Parent::Show();
    }
    static function Show() {
        echo __FILE__.'('.__LINE__.'):'.__METHOD__."\n";
    }
}
?>
```

Visibility

- Controlling member visibility / Information hiding
  - A derived class does not know inherited privates
  - An inherited protected member can be made public

```php
<?php
class Base {
  public $a;
  protected $b;
  private $c;
}
class Derived extends Base {
  public $a;
  public $b;
  private $c;
}
?>
```



Constructor visibility

- ☑ A protected constructor prevents instantiation

```
<?php
class Base {
   protected function __construct() {
   }
}
class Derived extends Base {
   // constructor is still protected
   static function getBase() {
      return new Base; // Factory pattern
   }
}
class Three extends Derived {
   public function __construct() {
   }
}
?>
```

Clone visibility

- ☑ A protected __clone prevents external cloning
- ☑ A private final __clone prevents cloning

```
<?php
class Base {
   protected function __clone() {
   }
}
class Derived extends Base {
   // public function __clone() {
   // return new Base;
   // }
   // public static function copyBase($that) {
   // return Base::__clone($that);
   // }
}
?>
```

Constants

- ☑ Constants are read only static properties
- ☑ Constants are always public

```
<?php
class Base {
  const greeting = "Hello\n";
}
class Dervied extends Base {
  const greeting = "Hello World\n";
  static function func() {
    echo parent::greeting;
  }
}
echo Base::greeting;
echo Derived::greeting;
Derived::func();
?>
```

Abstract members

- ☑ Properties cannot be made abstract
- ☑ Methods can be abstract
  - ☑ They don't have a body
  - ☑ A class with an abstract method must be abstract
- ☑ Classes can be made abstract
  - ☑ The class cannot be instantiated

```
<?php
abstract class Base {
  abstract function no_body();
}
class Derived extends Base {
  function no_body() { echo "Body\n"; }
}
?>
```

Final members

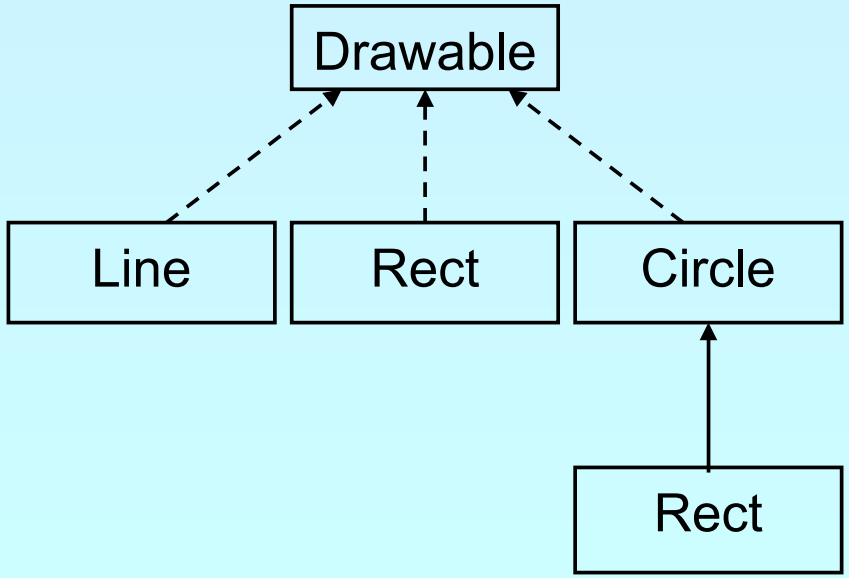
- ☑ Methods can be made final
  - ☑ They cannot be overwritten
  - ☑ They are class invariants
- ☑ Classes can be made final
  - ☑ They cannot be inherited

```
<?php
class Base {
  final function invariant() { echo "Hello\n"; }
}
class Derived extends Base {
}
final class Leaf extends Derived {
}
?>
```

Interfaces

- ☑ Interfaces describe an abstract class protocol
- ☑ Classes may inherit multiple Interfaces

```php
<?php
interface Drawable {
  function draw();
}
class Line implements Drawable {
 function draw() {};
}
class Rect implements Drawable {
 function draw() {};
}
class Circle implements Drawable {
 function draw() {};
}
class Ellipse extends Circle {
 function draw() {};
}
?>
```



Property types

- ☑ Declared properties
  - ☑ May have a default value
  - ☑ Can have selected visibility
- ☑ Implicit public properties
  - ☑ Declared by simply using them in ANY method
- ☑ Virtual properties
  - ☑ Handled by interceptor methods
- ☑ Static properties

Object to String conversion

☑ __toString(): automatic object string conversion

```php
<?php
class Object {
    function __toString() {
        return 'Object as string';
    }
}

$o = new Object;

echo $o;

$str = (string) $o;
?>
```

Interceptors

- ☑ Allow to dynamically handle non class members
  - ☑ Lazy initialization of properties
  - ☑ Simulating Object aggregation, Multiple inheritance

```php
<?php
class Object {
   protected $virtual;
   function __get($name) {
      return @$virtual[$name];
   }
   function __set($name, $value) {
      $virtual[$name] = $value;
   }
   function __call($func, $params) {
      echo 'Could not call ' . __CLASS__ . '::' . $func . "\n";
   }
}
?>
```

Exceptions

☑ Respect these rules

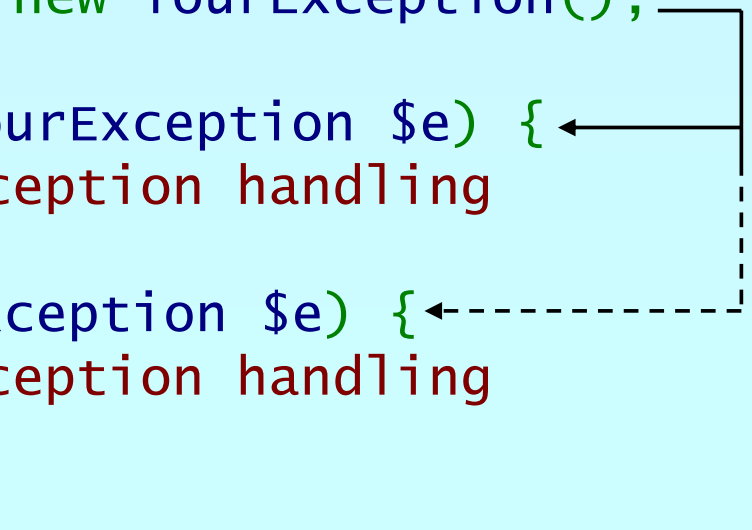
1. Exceptions are exceptions
2. Never use exceptions for control flow
3. Never ever use exceptions for parameter passing

```php
<?php
try {
    // your code
    throw new Exception();
}
catch (Exception $e) {
    // exception handling
}
?>
```

Exception specialization

- ☑ Exceptions should be specialized

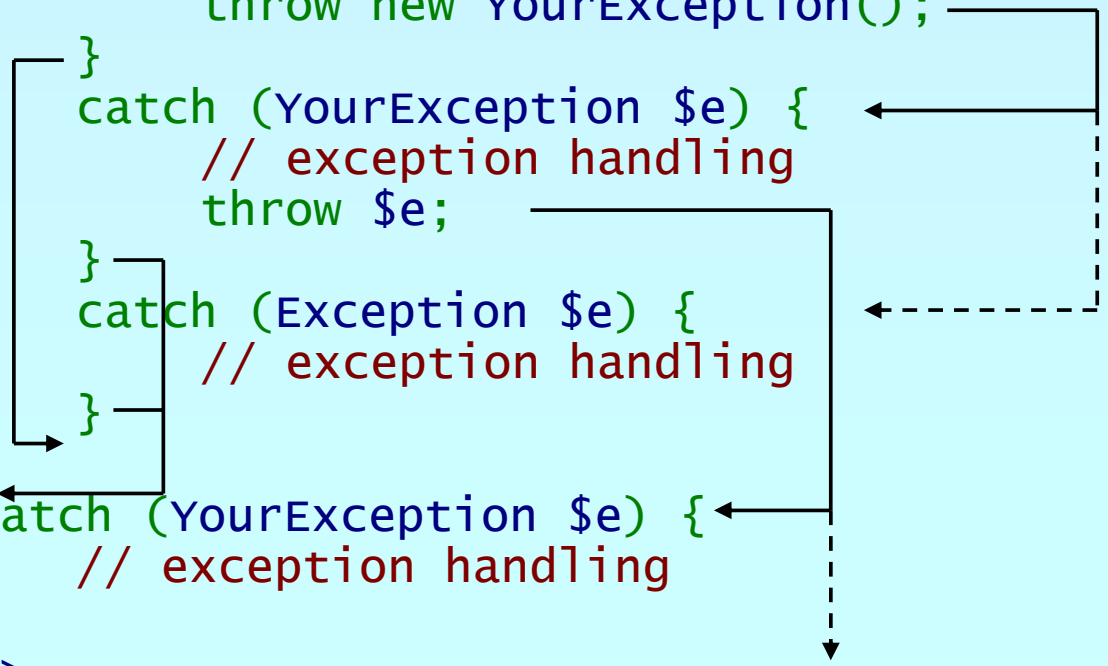
```
<?php
class YourException extends Exception {
}
try {
    // your code
    throw new YourException();
}
catch (YourException $e) {
    // exception handling
}
catch (Exception $e) {
    // exception handling
}
?>
```



Exception specialization

- ☑ Exception blocks can be nested
- ☑ Exceptions can be rethrown

```
<?php
class YourException extends Exception { }
try {
    try {
        // your code
        throw new YourException();
    }
    catch (YourException $e) {
        // exception handling
        throw $e;
    }
    catch (Exception $e) {
        // exception handling
    }
}
catch (YourException $e) {
    // exception handling
}
?>
```



Constructor failure

- ☑ Constructors do not return the created object
- ☑ Exceptions allow to handle failed constructors

```php
<?php
class Object {
    function __construct() {
        throw new Exception;
    }
}
try {
    $o = new Object;
}
catch (exception $e) {
    echo "Object could not be instantiated\n";
}
?>
```

Reflection API

- ☑ Can reflect nearly all aspects of your PHP code
  - ☑ Functions
  - ☑ Classes, Methods, Properties
  - ☑ Extensions

```php
<?php
class Foo {
    public $prop;
    function Func($name) {
        echo "Hello $name";
    }
}

reflection_class::export('Foo');
reflection_object::export(new Foo);
reflection_method::export('Foo', 'func');
reflection_property::export('Foo', 'prop');
reflection_extension::export('standard');
?>
```

Iterators

- ☑ Some objects can be iterated
- ☑ Others show their properties

```
<?php

class Object {
    public $prop1 = "Hello";
    public $prop2 = "World\n";
}

foreach(new Object as $prop) {
    echo $prop;
}

?>
```

Iterators

- ☑ Internal Iterators
- ☑ User Iterators

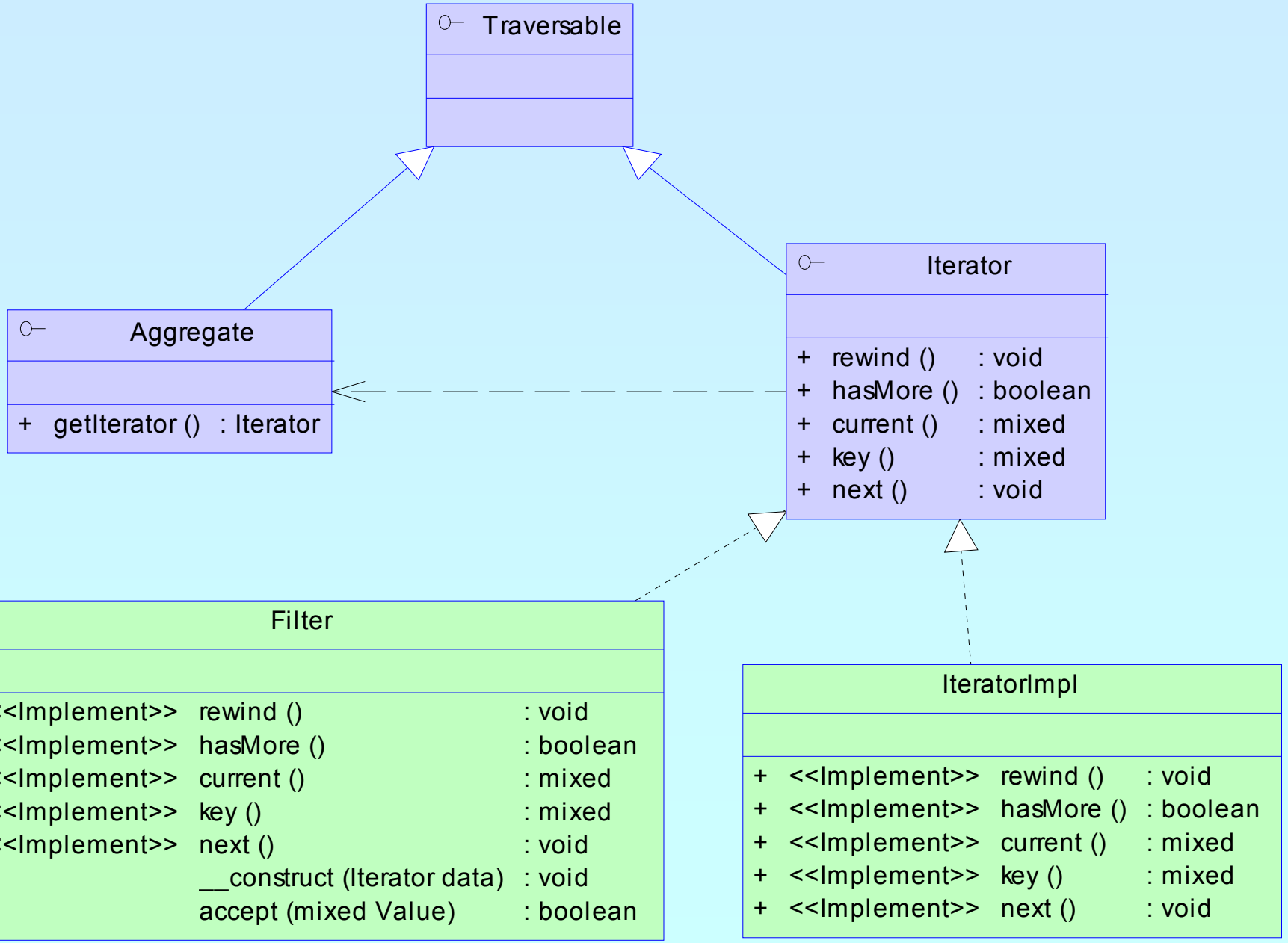
```php
<?php
interface Iterator {
  function rewind();
  function hasMore();
  function current();
  function key();
  function next();
}
?>
```

```php
<?php
class Filter implements Iterator {
  function __construct(Iterator $input)...
  function rewind()...
  function accept($value)...
  function hasMore()...
  function current()...
  function key()...
  function next()...
}
?>
```

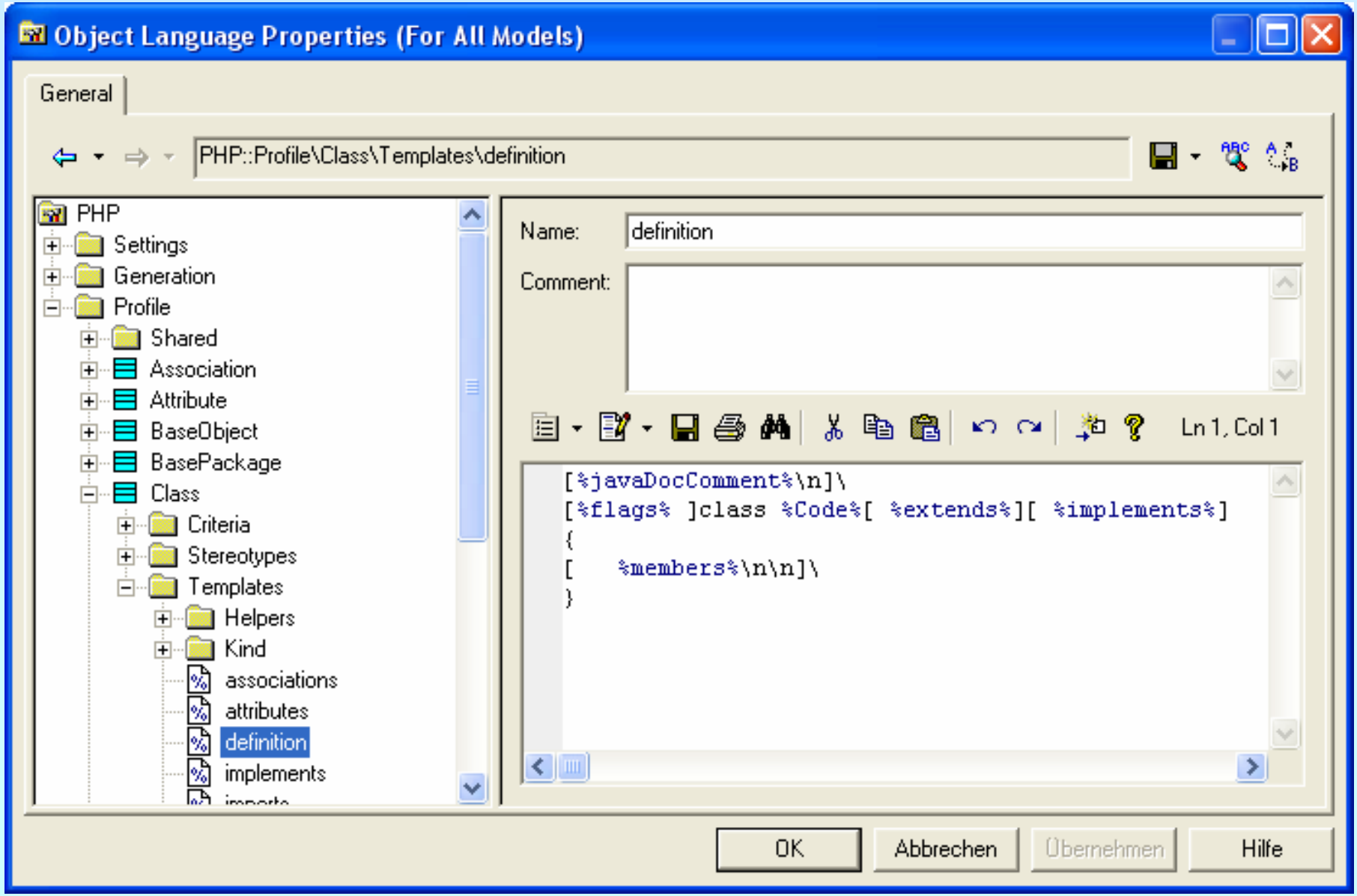
```php
<?php
$it = get_resource();
foreach($it as $key=>$val) {
  // access data
}
?>
```

```php
<?php
$it = get_resource();
$it = new Filter($it, $filter_params);
foreach($it as $key=>$val) {
  // access filtered data only
}
?>
```

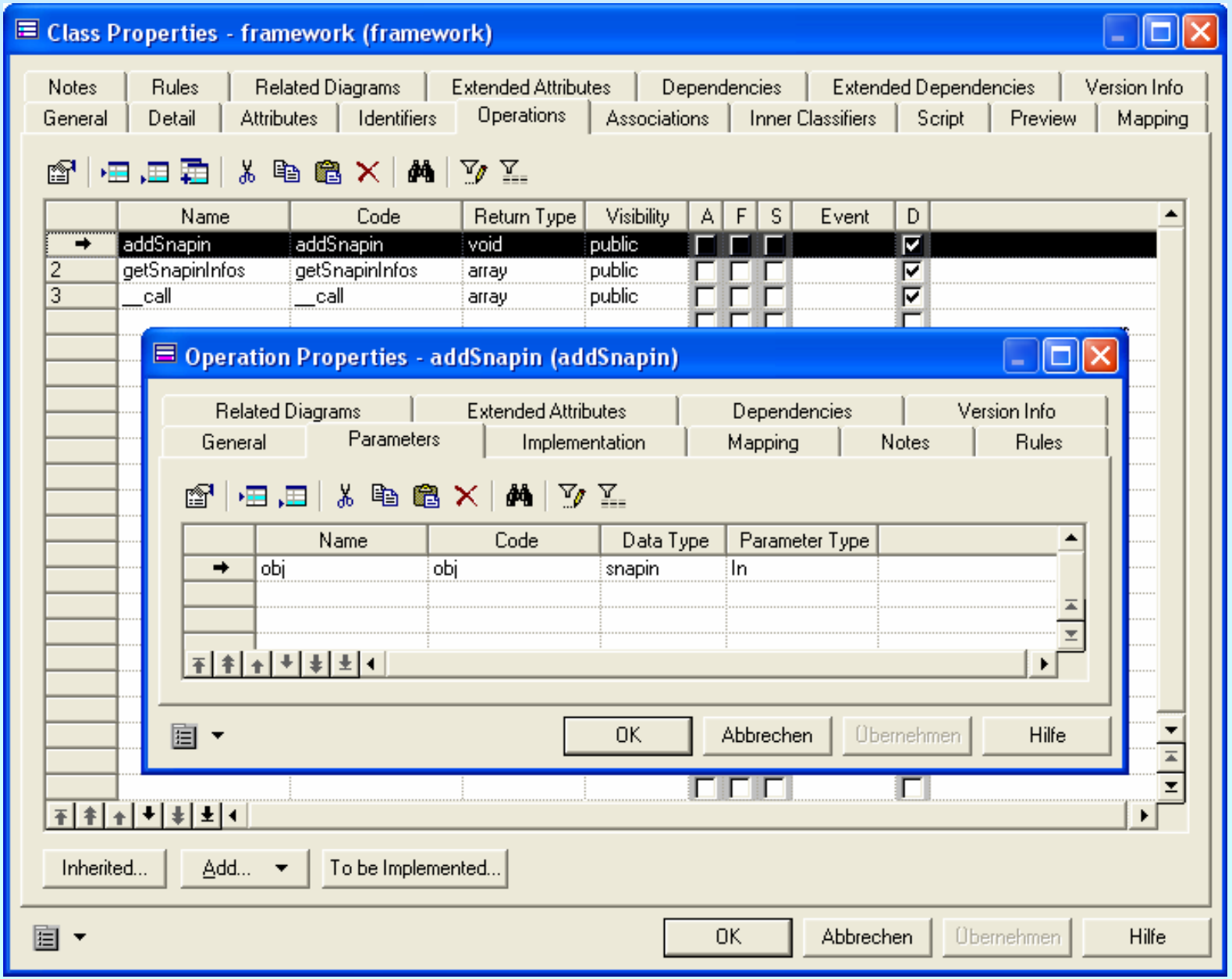
PHP & UML


Traversable
Aggregate
+ getIterator () : Iterator
Iterator
+ rewind () : void
+ hasMore () : boolean
+ current () : mixed
+ key () : mixed
+ next () : void
Filter
+ <<Implement>> rewind () : void
+ <<Implement>> hasMore () : boolean
+ <<Implement>> current () : mixed
+ <<Implement>> key () : mixed
+ <<Implement>> next () : void
+ __construct (Iterator data) : void
+ accept (mixed Value) : boolean
IteratorImpl
+ <<Implement>> rewind () : void
+ <<Implement>> hasMore () : boolean
+ <<Implement>> current () : mixed
+ <<Implement>> key () : mixed
+ <<Implement>> next () : void

PHP & UML


Object Language Properties (For All Models)
General
PHP::Profile\Class\Templates\definition
PHP
Settings
Generation
Profile
Shared
Association
Attribute
BaseObject
BasePackage
Class
Criteria
Stereotypes
Templates
Helpers
Kind
associations
attributes
definition
implements
Name:
definition
Comment:
Ln 1, Col 1
[%javaDocComment%\n]\
[%flags%]class %Code%[%extends%][%implements%]
{
[%members%\n\n]\
}
OK
Abbrechen
Übernehmen
Hilfe

PHP & UML


Class Properties - framework (framework)
Notes
Rules
Related Diagrams
Extended Attributes
Dependencies
Extended Dependencies
Version Info
General
Detail
Attributes
Identifiers
Operations
Associations
Inner Classifiers
Script
Preview
Mapping
Name
Code
Return Type
Visibility
A
F
S
Event
D
addSnapin
addSnapin
void
public
getSnapinInfos
getSnapinInfos
array
public
__call
__call
array
public
Operation Properties - addSnapin (addSnapin)
Related Diagrams
Extended Attributes
Dependencies
Version Info
General
Parameters
Implementation
Mapping
Notes
Rules
Name
Code
Data Type
Parameter Type
obj
obj
snapin
In
OK
Abbrechen
Übernehmen
Hilfe
Inherited...
Add...
To be Implemented...
OK
Abbrechen
Übernehmen
Hilfe

Typehinting

- PHP 5 allows to easily force a type of a parameter
  - Beta 1 and beta 2 allow NULL with typehints
  - Beta 3 will have a syntax to decide about NULL

```
<?php
class Object {
   public function compare(Object $other) {
      // Some code here
   }
}
?>
```

New extensions

- ☑ New extensions
  - ☑ FFI
  - ☑ DOM
  - ☑ MySQLi
  - ☑ PDO
  - ☑ PIMP
  - ☑ SimpleXML
  - ☑ SPL
  - ☑ SQLite
  - ☑ XML + XSL