Object Aggregation/Composition

Introduction

Warning
This extension is *EXPERIMENTAL*. The behaviour of this extension?including the names of its functions and any other documentation surrounding this extension?may change without notice in a future release of PHP. This extension should be used at your own risk.

In Object Oriented Programming, it is common to see the composition of simple classes (and/or instances) into a more complex one. This is a flexible strategy for building complicated objects and object hierarchies and can function as a dynamic alternative to multiple inheritance. There are two ways to perform class (and/or object) composition depending on the relationship between the composed elements: *Association* and *Aggregation*.

Examples

Object Aggregation examples

An *Association* is a composition of independently constructed and externally visible parts. When we associate classes or objects, each one keeps a reference to the ones it is associated with. When we associate classes statically, one class will contain a reference to an instance of the other class. For example:

Example #1 - Class association

```
<?php
class MyDateTime {

 function MyDateTime()
 {
     // empty constructor
 }

 function now()
 {
     return date("Y-m-d H:i:s");
 }
}

class Report {
 var $_dt;
 // more properties ...

 function Report()
 {
     $this->_dt = new MyDateTime();
     // initialization code ...
 }

 function generateReport()
 {
     $dateTime = $this->_dt->now();
     // more code ...
 }

 // more methods ...
}

$rep = new Report();
?>
```

We can also associate instances at runtime by passing a reference in a constructor (or any other method), which allow us to dynamically change the association relationship between objects. We will modify the example above to illustrate this point:

Example #2 - Object association

```
<?php
class MyDateTime {
 // same as previous example
}

class MyDateTimePlus {
 var $_format;

 function MyDateTimePlus($format="Y-m-d H:i:s")
 {
     $this->_format = $format;
 }

 function now()
 {
     return date($this->_format);
 }
}

class Report {
 var $_dt;    // we'll keep the reference to MyDateTime here
 // more properties ...

 function Report()
 {
     // do some initialization
 }

 function setMyDateTime(&$dt)
 {
     $this->_dt =& $dt;
 }

 function generateReport()
 {
     $dateTime = $this->_dt->now();
     // more code ...
 }

 // more methods ...
}

$rep = new Report();
$dt = new MyDateTime();
$dtp = new MyDateTimePlus("l, F j, Y (h:i:s a, T)");

// generate report with simple date for web display
$rep->setMyDateTime(&$dt);
echo $rep->generateReport();

// later on in the code ...

// generate report with fancy date
$rep->setMyDateTime(&$dtp);
$output = $rep->generateReport();
// save $output in database
// ... etc ...
```

```
?>
```

Aggregation, on the other hand, implies encapsulation (hidding) of the parts of the composition. We can aggregate classes by using a (static) inner class (PHP does not yet support inner classes), in this case the aggregated class definition is not accessible, except through the class that contains it. The aggregation of instances (object aggregation) involves the dynamic creation of subobjects inside an object, in the process, expanding the properties and methods of that object.

Object aggregation is a natural way of representing a whole-part relationship, (for example, molecules are aggregates of atoms), or can be used to obtain an effect equivalent to multiple inheritance, without having to permanently bind a subclass to two or more parent classes and their interfaces. In fact object aggregation can be more flexible, in which we can select what methods or properties to "inherit" in the aggregated object.

Examples

We define 3 classes, each implementing a different storage method:

Example #3 - storage_classes.inc

```
<?php
class FileStorage {
  var $data;

  function FileStorage($data)
  {
      $this->data = $data;
  }

  function write($name)
  {
      $fp = fopen(name, "w");
      fwrite($fp, $this->data);
      fclose($data);
  }
}

class WDDXStorage {
  var $data;
  var $version = "1.0";
  var $_id; // "private" variable

  function WDDXStorage($data)
  {
      $this->data = $data;
      $this->_id = $this->_genID();
  }

  function store()
  {
```

```
      if ($this->_id) {
          $pid = wddx_packet_start($this->_id);
          wddx_add_vars($pid, "this->data");
          $packet = wddx_packet_end($pid);
      } else {
          $packet = wddx_serialize_value($this->data);
      }
      $dbh = dba_open("varstore", "w", "gdbm");
      dba_insert(md5(uniqid("", true)), $packet, $dbh);
      dba_close($dbh);
  }

  // a private method
  function _genID()
  {
      return md5(uniqid(rand(), true));
  }
}

class DBStorage {
  var $data;
  var $dbtype = "mysql";

  function DBStorage($data)
  {
      $this->data = $data;
  }

  function save()
  {
      $dbh = mysql_connect();
      mysql_select_db("storage", $dbh);
      $serdata = serialize($this->data);
      mysql_query("insert into vars ('$serdata',now())", $dbh);
      mysql_close($dbh);
  }
}

?>
```

We then instantiate a couple of objects from the defined classes, and perform some aggregations and deaggregations, printing some object information along the way:

Example #4 - test_aggregation.php

```
<?php
include "storageclasses.inc";

// some utilty functions

function p_arr($arr)
{
  foreach ($arr as $k => $v)
      $out[] = "\t$k => $v";
  return implode("\n", $out);
```

```
}

function object_info($obj)
{
  $out[] = "Class: " . get_class($obj);
  foreach (get_object_vars($obj) as $var=>$val) {
      if (is_array($val)) {
          $out[] = "property: $var (array)\n" . p_arr($val);
      } else {
          $out[] = "property: $var = $val";
      }
  }
  foreach (get_class_methods($obj) as $method) {
      $out[] = "method: $method";
  }
  return implode("\n", $out);
}


$data = array(M_PI, "kludge != cruft");

// we create some basic objects
$fs = new FileStorage($data);
$ws = new WDDXStorage($data);

// print information on the objects
echo "\$fs object\n";
echo object_info($fs) . "\n";
echo "\n\$ws object\n";
echo object_info($ws) . "\n";

// do some aggregation

echo "\nLet's aggregate \$fs to the WDDXStorage class\n";
aggregate($fs, "WDDXStorage");
echo "\$fs object\n";
echo object_info($fs) . "\n";

echo "\nNow let us aggregate it to the DBStorage class\n";
aggregate($fs, "DBStorage");
echo "\$fs object\n";
echo object_info($fs) . "\n";

echo "\nAnd finally deaggregate WDDXStorage\n";
deaggregate($fs, "WDDXStorage");
echo "\$fs object\n";
echo object_info($fs) . "\n";

?>
```

We will now consider the output to understand some of the side-effects and limitation of object aggregation in PHP. First, the newly created *$fs* and *$ws* objects give the expected output (according to their respective class declaration). Note that for the purposes of object aggregation, *private elements of a class/object begin with an underscore character ("_")*, even though there is not real distinction between public and private class/object elements in PHP.

```
$fs object
Class: filestorage
property: data (array)
  0 => 3.1415926535898
  1 => kludge != cruft
method: filestorage
method: write

$ws object
Class: wddxstorage
property: data (array)
  0 => 3.1415926535898
  1 => kludge != cruft
property: version = 1.0
property: _id = ID::9bb2b640764d4370eb04808af8b076a5
method: wddxstorage
method: store
method: _genid
```

We then aggregate *$fs* with the WDDXStorage class, and print out the object information. We can see now that even though nominally the *$fs* object is still of FileStorage, it now has the property *$version*, and the method **store()**, both defined in WDDXStorage. One important thing to note is that it has not aggregated the private elements defined in the class, which are present in the *$ws* object. Also absent is the constructor from WDDXStorage, which will not be logical to aggegate.

```
Let's aggregate $fs to the WDDXStorage class
$fs object
Class: filestorage
property: data (array)
  0 => 3.1415926535898
  1 => kludge != cruft
property: version = 1.0
method: filestorage
method: write
method: store
```

The process of aggregation is cumulative, so when we aggregate *$fs* with the class DBStorage, generating an object that can use the storage methods of all the defined classes.

```
Now let us aggregate it to the DBStorage class
$fs object
Class: filestorage
property: data (array)
  0 => 3.1415926535898
  1 => kludge != cruft
property: version = 1.0
property: dbtype = mysql
```

```
method: filestorage
method: write
method: store
method: save
```

Finally, the same way we aggregated properties and methods dynamically, we can also deaggregate them from the object. So, if we deaggregate the class WDDXStorage from *$fs*, we will obtain:

```
And deaggregate the WDDXStorage methods and properties
$fs object
Class: filestorage
property: data (array)
  0 => 3.1415926535898
  1 => kludge != cruft
property: dbtype = mysql
method: filestorage
method: write
method: save
```

One point that we have not mentioned above, is that the process of aggregation will not override existing properties or methods in the objects. For example, the class FileStorage defines a *$data* property, and the class WDDXStorage also defines a similar property which will not override the one in the object acquired during instantiation from the class FileStorage.

Object Aggregation Functions

aggregate_info

aggregate_info -- Gets aggregation information for a given object

Description

array **aggregate_info** (object `$object`)

Gets the aggregation information for the given *`object`*.

Parameters

`object`

Return Values

Returns the aggregation information as an associative array of arrays of methods and properties. The key for the main array is the name of the aggregated class.

Examples

Example #5 - Using aggregate_info()

```
<?php

class Slicer {
   var $vegetable;

   function Slicer($vegetable)
   {
       $this->vegetable = $vegetable;
   }

   function slice_it($num_cuts)
   {
       echo "Doing some simple slicing\n";
       for ($i=0; $i < $num_cuts; $i++) {
           // do some slicing
       }
   }
}

class Dicer {
   var $vegetable;
   var $rotation_angle = 90;   // degrees

   function Dicer($vegetable)
```

```
    {
        $this->vegetable = $vegetable;
    }

    function dice_it($num_cuts)
    {
        echo "Cutting in one direction\n";
        for ($i=0; $i < $num_cuts; $i++) {
            // do some cutting
        }
        $this->rotate($this->rotation_angle);
        echo "Cutting in a second direction\n";
        for ($i=0; $i < $num_cuts; $i++) {
            // do some more cutting
        }
    }

    function rotate($deg)
    {
        echo "Now rotating {$this->vegetable} {$deg} degrees\n";
    }

    function _secret_super_dicing($num_cuts)
    {
        // so secret we cannot show you ;-)
    }
}

$obj = new Slicer('onion');
aggregate($obj, 'Dicer');
print_r(aggregate_info($obj));
?>
```

The above example will output:

```
Array
(
    [dicer] => Array
        (
            [methods] => Array
                (
                    [0] => dice_it
                    [1] => rotate
                )

            [properties] => Array
                (
                    [0] => rotation_angle
                )

        )

)
```

As you can see, all properties and methods of the Dicer class have been aggregated into our new object, with the exception of the class constructor and the method **_secret_super_dicing**

See Also

- aggregate()
- aggregate_methods()
- aggregate_methods_by_list()
- aggregate_methods_by_regexp()
- aggregate_properties()
- aggregate_properties_by_list()
- aggregate_properties_by_regexp()
- deaggregate()

aggregate_methods_by_list

aggregate_methods_by_list -- Selective dynamic class methods aggregation to an object

Description

void **aggregate_methods_by_list** (object `$object`, string `$class_name`, array `$methods_list` [, bool `$exclude`])

Aggregates methods from a class to an existing object using a list of method names.

The class constructor or methods whose names start with an underscore character (_), which are considered private to the aggregated class, are always excluded.

Parameters

`object`

`class_name`

`methods_list`

`exclude`
: The optional parameter *`exclude`* is used to decide whether the list contains the names of methods to include in the aggregation (i.e. *`exclude`* is **FALSE**, which is the default value), or to exclude from the aggregation (*`exclude`* is **TRUE**).

Return Values

No value is returned.

See Also

- aggregate()
- aggregate_info()
- aggregate_methods()
- aggregate_methods_by_regexp()
- aggregate_properties()
- aggregate_properties_by_list()
- aggregate_properties_by_regexp()
- deaggregate()

aggregate_methods_by_regexp

aggregate_methods_by_regexp -- Selective class methods aggregation to an object using a regular expression

Description

void **aggregate_methods_by_regexp** (object `$object`, string `$class_name`, string `$regexp` [, bool `$exclude`])

Aggregates methods from a class to an existing object using a regular expression to match method names.

The class constructor or methods whose names start with an underscore character (_), which are considered private to the aggregated class, are always excluded.

Parameters

`object`

`class_name`

`regexp`

`exclude`
: The optional parameter *`exclude`* is used to decide whether the regular expression will select the names of methods to include in the aggregation (i.e. *`exclude`* is **FALSE**, which is the default value), or to exclude from the aggregation (*`exclude`* is **TRUE**).

Return Values

No value is returned.

See Also

- aggregate()
- aggregate_info()
- aggregate_methods()
- aggregate_methods_by_list()
- aggregate_properties()
- aggregate_properties_by_list()
- aggregate_properties_by_regexp()

- deaggregate()

aggregate_methods

aggregate_methods -- Dynamic class and object aggregation of methods

Description

void **aggregate_methods** (object `$object`, string `$class_name`)

Aggregates all methods defined in a class to an existing object, except for the class constructor, or methods whose names start with an underscore character (_) which are considered private to the aggregated class.

Parameters

`object`

`class_name`

Return Values

No value is returned.

See Also

- aggregate()
- aggregate_info()
- aggregate_methods_by_list()
- aggregate_methods_by_regexp()
- aggregate_properties()
- aggregate_properties_by_list()
- aggregate_properties_by_regexp()
- deaggregate()

aggregate_properties_by_list

aggregate_properties_by_list -- Selective dynamic class properties aggregation to an object

Description

void **aggregate_properties_by_list** (object `$object`, string `$class_name`, array `$properties_list` [, bool `$exclude`])

Aggregates properties from a class to an existing object using a list of property names.

The properties whose names start with an underscore character (_), which are considered private to the aggregated class, are always excluded.

Parameters

`object`

`class_name`

`properties_list`

`exclude`
: The optional parameter *`exclude`* is used to decide whether the list contains the names of class properties to include in the aggregation (i.e. *`exclude`* is **FALSE**, which is the default value), or to exclude from the aggregation (*`exclude`* is **TRUE**).

Return Values

No value is returned.

See Also

- aggregate()
- aggregate_methods()
- aggregate_methods_by_list()
- aggregate_methods_by_regexp()
- aggregate_properties()
- aggregate_properties_by_regexp()
- aggregate_info()
- deaggregate()

aggregate_properties_by_regexp

aggregate_properties_by_regexp -- Selective class properties aggregation to an object using a regular expression

Description

void **aggregate_properties_by_regexp** (object `$object`, string `$class_name`, string `$regexp` [, bool `$exclude`])

Aggregates properties from a class to an existing object using a regular expression to match their names.

The properties whose names start with an underscore character (_), which are considered private to the aggregated class, are always excluded.

Parameters

`object`

`class_name`

`regexp`

`exclude`
: The optional parameter *`exclude`* is used to decide whether the regular expression will select the names of class properties to include in the aggregation (i.e. *`exclude`* is **FALSE**, which is the default value), or to exclude from the aggregation (*`exclude`* is **TRUE**).

Return Values

No value is returned.

See Also

- aggregate()
- aggregate_methods()
- aggregate_methods_by_list()
- aggregate_methods_by_regexp()
- aggregate_properties()
- aggregate_properties_by_list()

- aggregate_info()
- deaggregate()

aggregate_properties

aggregate_properties -- Dynamic aggregation of class properties to an object

Description

void **aggregate_properties** (object `$object`, string `$class_name`)

Aggregates all properties defined in a class to an existing object, except for properties whose names start with an underscore character (_) which are considered private to the aggregated class.

Parameters

`object`

`class_name`

Return Values

No value is returned.

See Also

- aggregate()
- aggregate_methods()
- aggregate_methods_by_list()
- aggregate_methods_by_regexp()
- aggregate_properties()
- aggregate_properties_by_list()
- aggregate_properties_by_regexp()
- deaggregate()

aggregate

aggregate -- Dynamic class and object aggregation of methods and properties

Description

void **aggregate** (object `$object`, string `$class_name`)

Aggregates methods and properties defined in a class to an existing object. Methods and properties with names starting with an underscore character (_) are considered private to the aggregated class and are not used, constructors are also excluded from the aggregation procedure.

Parameters

`object`

`class_name`

Return Values

No value is returned.

See Also

- aggregate_info()
- aggregate_methods()
- aggregate_methods_by_list()
- aggregate_methods_by_regexp()
- aggregate_properties()
- aggregate_properties_by_list()
- aggregate_properties_by_regexp()
- deaggregate()

aggregation_info

aggregation_info -- Alias of aggregate_info()

Description

This function is an alias of: aggregate_info().

deaggregate

deaggregate -- Removes the aggregated methods and properties from an object

Description

void **deaggregate** (object `$object` [, string `$class_name`])

Removes the methods and properties from classes that were aggregated to an object.

Parameters

`object`

`class_name`
If the optional *`class_name`* parameters is passed, only those methods and properties defined in that class are removed, otherwise all aggregated methods and properties are eliminated.

Return Values

No value is returned.

See Also

- aggregate()
- aggregate_methods()
- aggregate_methods_by_list()
- aggregate_methods_by_regexp()
- aggregate_properties()
- aggregate_properties_by_list()
- aggregate_properties_by_regexp()
- deaggregate()