Traits

Code reuse is one of the most important aspects of object-oriented programming. In PHP, you use [inheritance](http://www.zentut.com/php-tutorial/php-interface/)to enable code reuse in different classes that share the same inheritance hierarchy. To achieve code reuse, you move the common functionality of classes to method of the parent class. Inheritance makes the code very tightly coupled therefore makes the code hard to maintain.

To overcome this problem, as of version 5.4.0,  PHP introduced a new unit of code reuse named  trait.Traits allow you to reuse a set of methods freely in many different classes that does not need to be in the same class hierarchy.

Trait is similar to [class](http://www.zentut.com/php-tutorial/php-objects-and-classes/)but it is only for grouping methods in a fine-grained and consistent way. It is not allowed to instantiate a trait on its own.

## PHP traits example

Declaring a new trait is similar to declaring a new class as shown in the following example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php  trait Logger {  function log($msg) {  echo '<pre>';  echo date('Y-m-d h:i:s') . ':' . '(' . \_\_CLASS\_\_ .  ') ' . $msg . '<br/>';  echo '</pre>';  }  } |

To use a trait in a class, you use the use keyword. All the trait’s methods are available in the class where it is used. Calling a method of a trait is similar to calling an instance method.

The following example demonstrates how to use the Logger trait in the BankAccount class:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class BankAccount{  use Logger;    private $accountNumber;    function \_\_construct($accountNumber){  $this->accountNumber = $accountNumber;  $this->log("A new $accountNumber bank account created");  }  } |

We can also reuse the Logger trait in the User class as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | class User{  use Logger;  function \_\_construct() {  $this->log("A new user created");  }  } |

Both BankAccount and User classes reuse methods of the Logger trait, which is very flexible.

We can test our script to see how it works.

|  |  |
| --- | --- |
| 1  2 | $account = new BankAccount('1234567674');  $user = new User(); |

## Using multiple traits

A class can use multiple traits. The following example demonstrates how to use multiple traits in the IDE class. It simulates the [C compilation model](http://www.zentut.com/c-tutorial/c-compilation-model-and-processes/) in PHP for the sake of demonstration.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | <?php    trait Preprocessor{  function preprocess() {  echo 'Preprocess...done'. '<br/>';  }  }  trait Compiler{  function compile() {  echo 'Compile code... done'. '<br/>';  }  }    trait Assembler{  function createObjCode() {  echo 'Create the object code files... done.' . '<br/>';  }  }    trait Linker{  function createExec(){  echo 'Create the executable file...done' . '<br/>';  }  }    class IDE{  use Preprocessor, Compiler, Assembler, Linker;    function run() {  $this->preprocess();  $this->compile();  $this->createObjCode();  $this->createExec();    echo 'Execute the file...done' . '<br/>';  }  }    $ide = new IDE();  $ide->run(); |

## Composing multiple traits

A trait can be composed of other traits by using the use statement in the trait’s declaration. See the following example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | <?php    trait Reader{  public function read($source){  echo sprintf("Read from %s <br/>",$source);  }  }    trait Writer{  public function write($destination){  echo sprintf("Write to %s <br/>",$destination);  }  }    trait Copier{  use Reader, Writer;  public function copy($source,$destination){  $this->read($source);  $this->write($destination);  }  }    class FileUtil{  use Copier;  public function copyFile($source,$destination){  $this->copy($source, $destination);  }  } |

How it works.

* First, we created Reader and Writer traits.
* Second, we created a new trait named Copier that is composed of Reader and Writer traits. In the copy() method of the Copier trait, we called the  read() and write() methods of the Reader and Writer traits.
* Third, we used the Copier trait in the  copyFile() method of the FileUtil class to simulate the [copying file operation](http://www.zentut.com/php-tutorial/php-file-operations/).

## PHP trait’s method conflict resolution

### Overriding trait method

If a class uses multiple traits that share the same method name, HP will raise a fatal error. Fortunately, you can tell PHP which method of  which trait to be used by using inteadof keyword. Let’s take a look at the following example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | <?php  trait FileLogger{  public function log($msg){  echo 'File Logger ' . date('Y-m-d h:i:s') . ':' . $msg . '<br/>';  }  }    trait DatabaseLogger{  public function log($msg){  echo 'Database Logger ' . date('Y-m-d h:i:s') . ':' . $msg . '<br/>';  }  }    class Logger{  use FileLogger, DatabaseLogger{  FileLogger::log insteadof DatabaseLogger;  }  }    $logger = new Logger();  $logger->log('this is a test message #1');  $logger->log('this is a test message #2'); |

Both FileLogger and DatabaseLogger traits have the same  log() method. In the Logger class, we resolved the method name conflict by specifying that the  log() method of the FileLogger trait will be used instead of the DatabaseLogger‘s.

What if you want to use both methods in the FileLogger and DatabaseLogger traits? if so, you can use alias for the method  of the trait within the class that uses the trait.

### Aliasing trait method

By using aliases for the same method name of multiple traits, you can reuse all the methods in those traits. You use the as keyword to alias a method of a trait to a different name within the class that uses the trait. The following example illustrates how to alias trait method to resolve the method name conflict:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class Logger{  use FileLogger, DatabaseLogger{  DatabaseLogger::log as log2DB;  FileLogger::log insteadof DatabaseLogger;  }  }    $logger = new Logger();  $logger->log('this is a test message #1');  $logger->log2DB('this is a test message #2'); |

The method  log() of the DatabaseLogger class has a new name ( log2DB)  in the context of the Logger class.