Namespaces

Se utilizan para encapsular “items” (funciones, clases, constantes)
En php tiene dos propositos:

1. Evitar las colisiones entre nombres permitiendo, a su vez, la utilización de librerías externas sin necesidad de preocuparnos por los nombres de sus clases.
2. Permitir el uso de Alias para los nombres de las clases, a fin de evitar nombres largos y complicados o feos de leer en el código.

Los elementos afectados por los namespaces son:

- Clases
- Clases abstractas
- Traits
- Interfaces
- Funciones
- Constantes

Un namespace puede ser definido en varios archivos (justamente, indicando que esos archivos tienen cierta relación entre sí).
http://php.net/manual/en/language.namespaces.rationale.php

Declaracion

```
<?php

namespace MyProject;
```

Debe ser la primer sentencia en el principio del archivo.
Se pueden declarar varios namespaces por archivo pero no es una práctica recomendada.

http://php.net/manual/en/language.namespaces.definition.php
http://php.net/manual/en/language.namespaces.nested.php

Uso

Hay tres formas de utilizar los namespaces para utilizar sus “items”:

Unqualified
La clase (por ejemplo) se utiliza sin namespace, por lo tanto, PHP buscará la clase en el mismo namespace que contiene la clase actual.

Qualified
La clase (por ejemplo) se utiliza con un sub-namespace (es decir, una parte del namespace), por lo tanto, PHP buscará la clase en el mismo namespace que contiene la clase actual, agregando el sub-namespace.

Fully-Qualified
La clase (por ejemplo) se utiliza con todo el namespace completo a la hora de llamarla.

http://php.net/manual/en/language.namespaces.basics.php

Alias/Importing

La sentencia **use** nos permite hacer uso de los namespaces para indicar que vamos a utilizar una clase (o alguna de los otros "items"), sin necesidad de tener que escribir el fully-qualified name cada vez que queremos, por ejemplo, crear una nueva instancia.

Además, podemos hacer uso de la sentencia **as** que nos permite crear un alias para la clase en cuestión y cambiar su nombre en nuestro código, en el archivo que implementa el **use**.

Podemos elegir escribir la sentencia **use** indicando un namespace completo, parcial o incluír la clase o función que vamos a utilizar.

http://php.net/manual/en/language.namespaces.importing.php

http://php.net/manual/en/language.namespaces.faq.php

Traits

http://php.net/traits

Los traits son un mecanismo que permite la reutilización de código cuando no se permite la herencia múltiple en clases.

Son similares a una clase, pero solo están pensados para agrupar funcionalidad.

```
<?php
trait ATrait {
    function getType() { /*1*/ }
    function getDescription() { /*2*/ }
}

class A extends B {
    use ATrait;
    /* ... */
}
```

Al heredar métodos, el trait pisa lo heredado del padre y la clase que hace uso del trait, pisa lo declarado por el trait. No así con las propiedades. No puede haber repetición de nombres entre las propiedades declaradas por un trait y la clase que lo utiliza.

```
<?php
class Base {
    public function sayHello() {
        echo 'Hello ';
    }
}

trait SayWorld {
    public function sayHello() {
```

```
        parent::sayHello();
        echo 'World!';
    }
}

class MyHelloWorld extends Base {
    use SayWorld;
}

$o = new MyHelloWorld();
$o->sayHello();
```

Otro ejemplo:

```
<?php
trait HelloWorld {
    public function sayHello() {
        echo 'Hello World!';
    }
}

class TheWorldIsNotEnough {
    use HelloWorld;
    public function sayHello() {
        echo 'Hello Universe!';
    }
}

$o = new TheWorldIsNotEnough();
$o->sayHello();
```

Utilizando la sentencia **use**, vemos que podemos utilizar un trait dentro de una clase. Si el trait pertenece a un **namespace** en particular, deberemos indicarlo en la sentencia **use** dentro de la clase, escribir la sentencia **use** fuera de la clase, que nos permitirá abreviar el nombre en la sentencia **use** interna.

Podemos utilizar mas de un trait por clase separandolos por comas al escribir la sentencia **use**; o podemos escribir varias sentencias **use** separadas por punto y coma.

A su vez, podemos utilizar (con la misma sintaxis) uno o más traits adentro de otro.

Puede suceder que los métodos y propiedades de un trait colisionen con los de otro. Para ver la solución a ese problema:
http://php.net/manual/en/language.oop5.traits.php#language.oop5.traits.conflict

Los traits soportan el uso de métodos abstractos, es decir, que la clase que utilice el trait, estará obligada a implementar ese método.