Basic Usage

Basic Usage

At it's core, League\Container allows you to register services, with or without their dependencies for later retrieval. It is a registry of sorts that used correctly can allow you to implement the Dependency Injection design pattern.

More...

Classes & dependencies

We have a class, `Foo` , that depends on another class, `Bar` .

```php
<?php

namespace Acme;

class Foo
{
    /**
     * @type Acme\Bar
     */
    public $bar;

    /**
     * Construct.
     *
     * @param \Acme\Bar $bar
     */
    public function __construct(Bar $bar)
    {
        $this->bar = $bar;
    }
}

class Bar
{
    // ...
}
```

There are several ways to now register this service with the container.

Aliases

The container can be configured to register `Foo` against an alias with it's dependencies as a prototype so that every time it is retrieved, it will be a new instance of `Foo` with `Bar` injected in to the constructor.

```php
<?php

use Acme\{Foo, Bar};
use League\Container\Container;

$container = new Container;

$container
    ->add('alias_for_foo', Foo::class)
```

```php
$container->add('alias_for_bar', Bar::class);

$foo1 = $container->get('alias_for_foo');
$foo2 = $container->get('alias_for_foo');

var_dump($foo1 instanceof Foo);      // true
var_dump($foo1 === $foo2);           // false
var_dump($foo1->bar instanceof Bar); // true
var_dump($foo1->bar === $foo2->bar); // false
```

Aliases can be useful when implementing interfaces.

```php
<?php

namespace Acme;

interface FooInterface
{
    // ...
}

class Foo implements FooInterface
{
    // ...
}
```

Now you can register `Foo` against it's interface as an alias `FooInterface`, meaning that whenever you retrieve `FooInterface` from the container, it will be the concrete implementation `Foo` that is returned.

```php
<?php

use Acme\{Foo, FooInterface};
use League\Container\Container;

$container = new Container;

$container->add(FooInterface::class, Foo::class);
```

```php
<?php

$container = new League\Container\Container;

// register the service as a prototype against the fully qualified classname
$container->add('Acme\Service\SomeService');

// now to retrieve this service we can just retrieve the classname
// each time we `get` the service it will be a new instance
$service1 = $container->get('Acme\Service\SomeService');
$service2 = $container->get('Acme\Service\SomeService');

var_dump($service1 instanceof Acme\Service\SomeService); // true
var_dump($service1 === $service2); // false
```

There may be occasions where you wish the service to be the same instance each time you retrieve it. There are two ways to achieve this, declare it as shared, or register a ready built instance of an object.

```php
<?php

$container = new League\Container\Container;

// register the service as shared against the fully qualified classname
$container->share('Acme\Service\SomeService');

// you retrieve the service in exactly the same way, however, each time you
// call `get` you will retrieve the same instance
$service1 = $container->get('Acme\Service\SomeService');
$service2 = $container->get('Acme\Service\SomeService');

var_dump($service1 instanceof Acme\Service\SomeService); // true
var_dump($service1 === $service2); // true
```

```php
// register the service as an instance against an alias
$container->add('service', new Acme\Service\SomeService);

// you retrieve the service in exactly the same way, however, each time you
// call `get` you will retrieve the same instance
$service1 = $container->get('service');
$service2 = $container->get('service');

var_dump($service1 instanceof Acme\Service\SomeService); // true
var_dump($service1 === $service2); // true
```

© Container The PHP League | Design: HTML5 UP | Jekyll integration: Andrew Banchich | Additional tinkering: Phil Bennett