Service Container

PHP Test

Implement class that will act as a service container for managing providers to have a way of dependency injection.
Provider type can be scalar, object, array, resource or closure.
Registering provider that already exists is not considered as error.

1. Implement class Container

Having two methods:

```
Container::set($name, $provider, $singleton = false);
Container::get($name, $params = array());


$container = new Container();

$container->set("book", "Lord of the flies");
$container->set("number", 317);

$container->set("now", function() {
        return date("F j, Y, g:i a");
});

$container->set("hello", function($firstName, $lastName) {
        return "Hello {$firstName} {$lastName}";
});

echo $container->get("book"); // Prints "Lord of the flies"
echo $container->get("number"); // Prints 317
echo $container->get("now"); // Prints now date ("August 23, 2015, 7:28 am")
echo $container->get("hello", array("John", "Doe")));  // Prints "Hello John Doe"
```

2. Implement accessing and changing providers as property

Closure providers in this case can not accept parameters.

```
$container->book = "Lord of the flies 2";
echo $container->book; // Prints "Lord of the flies 2"
echo $container->now; // Prints now date ("August 23, 2015, 7:28 am")
```

3. Implement accessing and changing providers as array

Closure providers in this case can not accept parameters.

```
$container["book"] = "Lord of the flies 3";
echo $container[“book”]; // Prints "Lord of the flies 3"
echo $container[“now”]; // Prints now date ("August 23, 2015, 7:28 am")
```

4. Implement accessing providers as function

Closure providers can accept parameters.

```
echo $container->book(); // Prints "Lord of the flies"
echo $container->number(); // Prints 317
echo $container->now(); // Prints now date ("August 23, 2015, 7:28 am")
echo $container->hello("John", "Doe"); // Prints "Hello John Doe"
```

5. Implement singleton access

Having third parameter singleton as true in Container::set method should always
return the same instance for result. If the provider is not closure, it is treated as constant.

```
$container->set("db", function($dsn, $user, $pass) {
	return new \PDO($dsn, $user, $pass);
}, true);

$db = $container->get("db"); // Always returns the same instance

$container->set("MAX_BUFFER_SIZE", 200, true);
$container->set("hash", function() {
	return md5(gethostname() . time());
}, true);

$value = $container->MAX_BUFFER_SIZE;
echo $value; // Prints 200

$container->MAX_BUFFER_SIZE = 300;
echo $value; // Still prints 200

$value = $container->hash();
echo $value; // Prints "c5fbeb164b784672ae118d0442aa7be6"

$value = $container->hash();
echo $value; // Still prints "c5fbeb164b784672ae118d0442aa7be6"
```

6. Implement provider interface

Provider interface makes services reusable.

```
Container::register(Provider $provider);

$container->register(new UserServiceProvider());
$container->get("UserService")->getUser(317);
$container->get("UserApplicationService")->getUserApplications(317);


interface Provider {
	public function register(Container $container) {}
}

class UserServiceProvider implements Provider {
	public function register(Container $container) {
		$db = $container->get("db");

		$container->set("UserService", function() use($db) {
			return new UserService($db);
		});

		$container->set("UserApplicationService", function() use($db) {
			return new UserApplicationService($db);
		});
	}
}

class UserServiceBase {
	public function __construct(\PDO $pdo) { ... }
}

class UserService extends UserServiceBase {
	public function getUser($userId) { ... }
}

class UserApplicationService extends UserServiceBase{
	public function getUserApplications($userId) { ... }
}
```

Marko Prelic
23. August 2015.