Dependency injection container by Kai Sassnowski [Laracon EU]

YT link for video

SessionStorage Class:

```php
<?php
class SessionStorage {

   public function __construct () {
       session_start();
   }

   public function set ($key, $value) {
       $_SESSION[$key] = $value;
   }

   public function get ($key) {
       return $_SESSION[$key];
   }
}
```

User Class:

```php
<?php

class User {

   protected $storage;

   public function __construct ()
   {
       $this->storage = new SessionStorage();
   }

   protected function setLanguage ($language) {
       $this->storage->set('language', $language);
   }

   protected function getLanguage () {
       return $this->storage->get('language');
```

```
    }
}
```

- The User class is instantiating its own dependency in its container which causes tight coupling and difficult to test the User class since we need to test the SessionStorage class too.

- Solution: Constructor Injection

```
<?php

class User {

    protected $storage;

    public function __construct (SessionStorage $session)
    {
        $this->storage = $session;
    }

    protected function setLanguage ($language) {
        $this->storage->set('language', $language);
    }

    protected function getLanguage () {
        return $this->storage->get('language');
    }
}
```

```
$storage = new SessionStorage();
$user = new User($storage);
$user->setLanguage('english');
$user->getLanguage();
```

- For simple two classes depending on each other, this might be a good solution. But imagine that a class 1 depending on another class 2 depends on class 3 and class 3 depends on class 4 and class 5 and so forth.

- We had to do this:

```
$object = new Class1(
                Class2(
                        Class3(
                                Class4, Class 5
                        )
                )
        );
```

```php
$svc = new ShippingService(new ProductLocator(),
    new PricingService(), new InventoryService(),
    new TrackingRepository(new ConfigProvider()),
    new Logger(new EmailLogger(new ConfigProvider())));
```

- For this complex dependency injection we need a dependency injection container which helps to abstract the classes that are depended on.

SessionStorage class:

This class depends on Logger class.

```php
<?php
namespace DIC;

class SessionStorage {

   protected $logger;

   public function __construct (Logger $logger) {
       session_start();
       $this->logger = $logger;
   }

   public function set ($key, $value) {
       $this->logger->log(
           "Setting [$key] to [$value]"
       );
       $_SESSION[$key] = $value;
```

```
    }

    public function get ($key) {
        return $_SESSION[$key];
    }
}
```

User class:

This class depends on the SessionStorage class.

```
<?php
namespace DIC;

class User {

    protected $storage;

    public function __construct (SessionStorage $session)
    {
        $this->storage = $session;
    }

    public function setLanguage ($language) {
        $this->storage->set('language', $language);
    }

    public function getLanguage () {
        return $this->storage->get('language');
    }
}
```

Logger class:

This class has no dependencies.

```
<?php
namespace DIC;
```

```php
class Logger {

   public function log ($message) {
       echo "Logging $message to logger";
   }
}
```

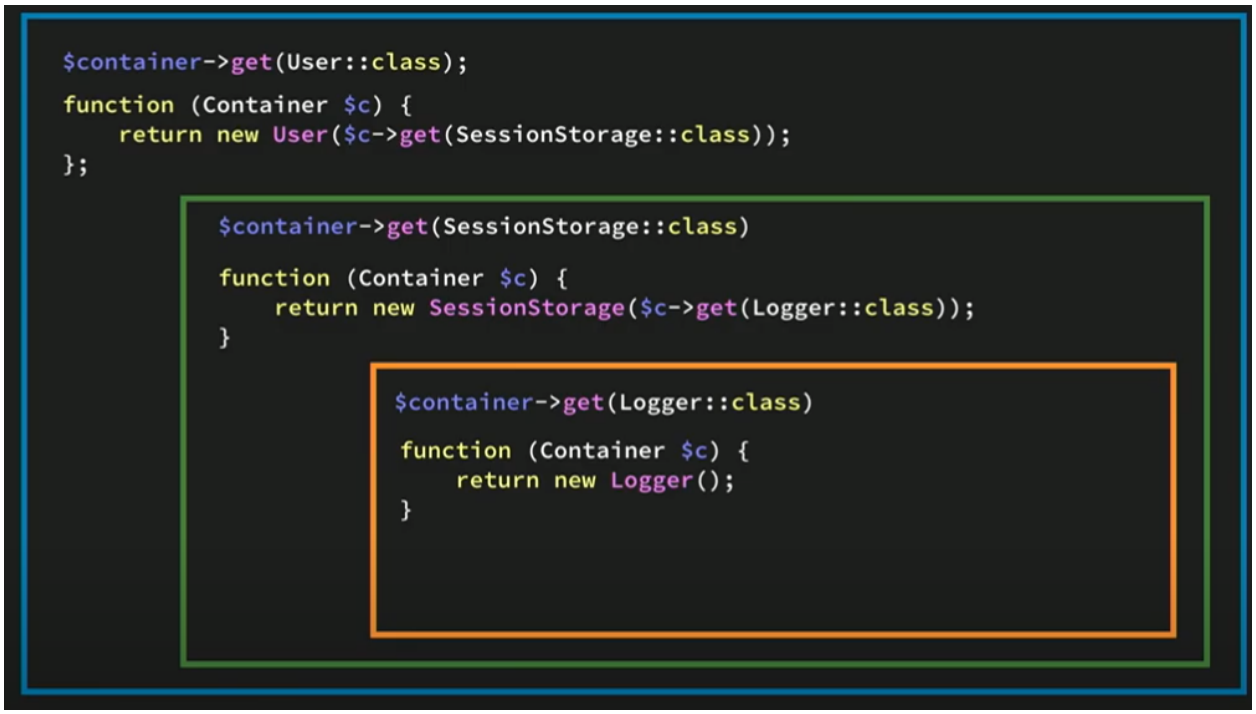
Container class:

This class is a dependency container.

```php
<?php
namespace DIC;

class Container {
   private $bindings = [];

   public function set ($abstract, callable $factory) {
       $this->bindings[$abstract] = $factory;
   }

   public function get ($abstract) {
       return $this->bindings[$abstract]($this);
   }
}

$container = new Container();

$container->set(User::class, function (Container $c) {
           return new User($c->get(SessionStorage::class));
});

$container->set(SessionStorage::class, function (Container $c) {
           return new SessionStorage($c->get(Logger::class));
});

$container->set(Logger::class, function (Container $c) {
           return new Logger();
});
```

```php
$user = $container->get(User::class);
$user->setLanguage('English');
$user->getLanguage();
```

Overview of dependencies using a container:



Every DI container does this or similar:

User(SessionStorage(Logger()));

But at the core this is the recursive building of dependencies.

Advanced DI container with auto wiring:

Autowiring - The ability of the container to automatically build a class without needing an explicit binding or explicit factory registered in order for the container to be able to give back an instance of it.

Reflection based autowiring:

1. Build a ReflectionClass
2. Build its dependencies
    - Get the constructor
    - Get its parameters
    - Look at the types
    - Recursively build the dependencies
3. Create new instance

Container class:

```php
<?php

namespace DIC;

use ReflectionClass;
use RuntimeException;

class Container {
   private $bindings = [];

   public function set ($abstract, callable $factory) {
       $this->bindings[$abstract] = $factory;
   }

   public function get ($abstract) {

       if (isset($this->bindings[$abstract])) {
           return $this->bindings[$abstract]($this);
       }

       # build a reflection class
       $reflection = new ReflectionClass($abstract);

       # build its dependencies
       $dependencies = $this->buildDependencies($reflection);

       # new instance with these args
       return $reflection->newInstanceArgs($dependencies);
```

```
    }


    private function buildDependencies ($reflection) {

        # get the constructor
        if (!$constructor = $reflection->getConstructor()) {
            return [];
        }

        # get its parameters
        $params = $constructor->getParameters();

        return array_map(function ($param) {
            # look at the types
            if (!$type = $param->getType()) {
                throw new RuntimeException();
            }

            # recursively build the dependencies
            return $this->get($type);
        }, $params);
    }
}
```

Index.php file:

```
$container = new Container();
$user = $container->get(User::class);
$user->setLanguage('English');
$user->getLanguage();
```