6. Services

I will ask you to get the latest code from chapter 5.

```
$ git clone git@github.com:skafandri/symfony-tutorial.git --branch ch5
remote: Counting objects: 638, done.
remote: Total 638 (delta 0), reused 0 (delta 0), pack-reused 638
Receiving objects: 100% (638/638), 403.58 KiB | 173.00 KiB/s, done.
Resolving deltas: 100% (317/317), done.
Checking connectivity... done.
```

6.1 Add quantity to product - warehouse relationship

We will group some warehouse related functionalities as a service. But before, we have something to fix. We have a ManytoMany relation between warehouse and product tables.

- Update **src/AppBundle/Entity/Product.php** and remove everything related to warehouse

```php
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Product
 *
 * @ORM\Table(
 *  name="product",
 *  indexes={
 *      @ORM\Index(name="code_index", columns={"code"}),
 *      @ORM\Index(name="title_index", columns={"title"})
 *  })
 * @ORM\Entity
 */
class Product
{

    const REPOSITORY = 'AppBundle:Product';

    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $id;
```

```php
    /**
     * @var string
     *
     * @ORM\Column(name="code", type="string", length=45, nullable=true)
     */
    private $code;

    /**
     * @var string
     *
     * @ORM\Column(name="title", type="string", length=200, nullable=true)
     */
    private $title;

    /**
     * @var string
     *
     * @ORM\Column(name="description", type="text", nullable=true)
     */
    private $description;

    /**
     * @var \Doctrine\Common\Collections\Collection
     *
     * @ORM\ManyToMany(targetEntity="Category", mappedBy="product")
     */
    private $category;

    /**
     * Constructor
     */
    public function __construct()
    {
        $this->category = new \Doctrine\Common\Collections\ArrayCollection();
    }

    /**
     * Get id
     *
     * @return integer
     */
    public function getId()
    {
        return $this->id;
    }
```

```php
    /**
     * Set code
     *
     * @param string $code
     * @return Product
     */
    public function setCode($code)
    {
        $this->code = $code;

        return $this;
    }

    /**
     * Get code
     *
     * @return string
     */
    public function getCode()
    {
        return $this->code;
    }

    /**
     * Set title
     *
     * @param string $title
     * @return Product
     */
    public function setTitle($title)
    {
        $this->title = $title;

        return $this;
    }

    /**
     * Get title
     *
     * @return string
     */
    public function getTitle()
    {
        return $this->title;
    }
```

```php
    /**
     * Set description
     *
     * @param string $description
     * @return Product
     */
    public function setDescription($description)
    {
        $this->description = $description;

        return $this;
    }

    /**
     * Get description
     *
     * @return string
     */
    public function getDescription()
    {
        return $this->description;
    }

    /**
     * Add category
     * @param \AppBundle\Entity\Category $category
     * @return Product
     */
    public function addCategory(\AppBundle\Entity\Category $category)
    {
        $this->category[] = $category;

        return $this;
    }

    /**
     * Remove category
     *
     * @param \AppBundle\Entity\Category $category
     */
    public function removeCategory(\AppBundle\Entity\Category $category)
    {
        $this->category->removeElement($category);
    }

    /**
     * Get category
     *
     * @return \Doctrine\Common\Collections\Collection
     */
    public function getCategory()
    {
        return $this->category;
    }
```

```
    public function __toString()
    {
        return $this->getTitle();
    }

}
```

- Update **src/AppBundle/Entity/Warehouse.php** and remove everything related to product

```
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Warehouse
 *
 * @ORM\Table(name="warehouse")
 * @ORM\Entity
 */
class Warehouse
{

    const REPOSITORY = 'AppBundle:Warehouse';

    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="name", type="string", length=45, nullable=true)
     */
    private $name;

    /**
     * @var string
     *
     * @ORM\Column(name="address", type="text", nullable=true)
     */
    private $address;
```

```php
    /**
     * Get id
     *
     * @return integer
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * Set name
     *
     * @param string $name
     * @return Warehouse
     */
    public function setName($name)
    {
        $this->name = $name;

        return $this;
    }

    /**
     * Get name
     *
     * @return string
     */
    public function getName()
    {
        return $this->name;
    }

    /**
     * Set address
     *
     * @param string $address
     * @return Warehouse
     */
    public function setAddress($address)
    {
        $this->address = $address;
        return $this;
    }

    /**
     * Get address
     *
     * @return string
     */
    public function getAddress()
    {
        return $this->address;
    }
```

```
    public function __toString()
    {
        return $this->getName();
    }

}
```

- Create **src/AppBundle/Entity/ProductStock.php**

```
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * ProductStock
 *
 * @ORM\Table(
 *  name="product_stock",
 *  indexes={
 *      @ORM\Index(name="product_stock_product_id_idx", columns={"product_id"}),
 *      @ORM\Index(name="product_stock_warehouse_id_idx", columns={"warehouse_id"})
 * })
 * @ORM\Entity
 */
class ProductStock
{

    const REPOSITORY = 'AppBundle:ProductStock';

    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $id;

    /**
     * @var Product
     *
     * @ORM\ManyToOne(targetEntity="Product")
     * @ORM\JoinColumns({
     *   @ORM\JoinColumn(name="product_id", referencedColumnName="id")
     * })
     */
    private $product;
```

```php
    /**
     * @var Warehouse
     *
     * @ORM\ManyToOne(targetEntity="Warehouse")
     * @ORM\JoinColumns({
     *   @ORM\JoinColumn(name="warehouse_id", referencedColumnName="id")
     * })
     */
    private $warehouse;

    /**
     * @var integer
     *
     * @ORM\Column(name="quantity", type="integer", nullable=true)
     */
    private $quantity;

    /**
     * Set quantity
     *
     * @param integer $quantity
     * @return ProductStock
     */
    public function setQuantity($quantity)
    {
        $this->quantity = $quantity;

        return $this;
    }

    /**
     * Get quantity
     *
     * @return integer
     */
    public function getQuantity()
    {
        return $this->quantity;
    }

    /**
     * Set product
     *
     * @param Product $product
     * @return ProductStock
     */
    public function setProduct(Product $product = null)
    {
        $this->product = $product;

        return $this;
    }
```

```
    /**
     * Get
     *
     * @return Warehouse
     */
    public function getWarehouse()
    {
        return $this->warehouse;
    }

    /**
     * Set warehouse
     *
     * @param Warehouse $warehouse
     * @return ProductStock
     */
    public function setWarehouse(Warehouse $warehouse = null)
    {
        $this->warehouse = $warehouse;

        return $this;
    }

    /**
     * Get product
     *
     * @return Product
     */
    public function getProduct()
    {
        return $this->product;
    }

}
```

You can update your database schema by running `app/console doctrine:schema:drop --force` then `app/console doctrine:schema:create`

6.2 Warehouse service

A symfony service is a normal class that the instantiation is delegated to the framework. A service is encapsulated in a singleton pattern, and external dependencies are checked before registering a service definition.

Let's start by defining a simple service that just returns the list of warehouses.

- Create **src/AppBundle/Service/WarehouseService.php**

```
<?php

namespace AppBundle\Service;

use Doctrine\ORM\EntityManager;

class WarehouseService
{

    const ID = 'app.warehouse';

    private $container;

    /**
     *
     * @var EntityManager
     */
    private $entityManager;

    public function __construct($container)
    {
        $this->container = $container;
        $this->entityManager = $container->get('doctrine')->getManager();
    }

    public function getAll()
    {
        return $this->entityManager
                        ->getRepository(\AppBundle\Entity\Warehouse::REPOSITORY)
                        ->findAll();
    }

}
```

- Edit **src/AppBundle/Resources/config/services.yml**
  Add the follwoing new item under *services*

```
app.warehouse:
        class: AppBundle\Service\WarehouseService
        arguments: [@service_container]
```

To see the new service in action, let's update the controller's index action to use the new service.

- Edit **src/AppBundle/Controller/WarehouseController.php**
  Change
  `$entities = $em->getRepository('AppBundle:Warehouse')->findAll();`
  to
  `$entities = $this->get(\AppBundle\Service\WarehouseService::ID)->getAll();`
  in indexAction

The previous changes we made to the warehouse entity broke the create and edit forms.
To fix it, simply edit **src/AppBundle/Form/WarehouseType.php** and remove `->add('product')`

The Warehouse service we just create has two dependencies. On explicit dependency on a *service_container* and one implicit dependency on *doctrine*.

I don't recommend using *hidden* (implicit) dependencies for two reasons:

- It is impossible to know on which services a service depends only by looking at the service definition.
- If your service depends on a service that doesn't exist, you will find out only in runtime, and sometimes only when a specific execution path happens within your service.

6.3 Warehouse service without implicit dependencies

Let's clean up our service to remove implicit dependencies. We will forget about injecting a service_container into our services unless is really needed.

- Edit **src/AppBundle/Resources/config/services.yml** and inject `@doctrine` instead of `@service_container` to **app.warehouse** service definition.
- Edit **src/AppBundle/Service/WarehouseService.php** we will also add another functionality

```php
<?php

namespace AppBundle\Service;

use AppBundle\Entity\ProductStock;
use AppBundle\Entity\Warehouse;
use Doctrine\Bundle\DoctrineBundle\Registry;
use Doctrine\ORM\EntityManager;

class WarehouseService
{

    const ID = 'app.warehouse';

    private $doctrine;

    /**
     *
     * @var EntityManager
     */
    private $entityManager;

    public function __construct(Registry $doctrine)
    {
        $this->doctrine = $doctrine;
        $this->entityManager = $doctrine->getManager();
    }

    public function getAll()
    {
        return $this->entityManager
                        ->getRepository(Warehouse::REPOSITORY)
                        ->findAll();
    }

    public function getProductStocks($productId)
    {
        return $this->entityManager->
                        getRepository(ProductStock::REPOSITORY)
                        ->findBy(array('product' => $productId));
    }

}
```

In order to see the new method `getProductStocks` in action, we will display the product stocks in the products listing.
Before that, we need to remove the warehouse from the product form type.

- Edit **src/AppBundle/Form/ProductType.php** and remove `->add('warehouse')`

We can proceed with the controller and views now.

- Edit **src/AppBundle/Controller/WarehouseController.php** add the follwing action

```
public function productStocksAction($productId)
{
    $productStocks = $this->get(\AppBundle\Service\WarehouseService::ID)
                          ->getProductStocks($productId);

    return $this->render('AppBundle:Product:product_stock.html.twig', array(
                'productStocks' => $productStocks,
    ));
}
```

- Create **src/AppBundle/Resources/views/Product/product_stock.html.twig**

```
{% for stock in productStocks %}
    {{ stock.warehouse.name }} :  {{ stock.quantity }}
{% endfor %}
```

- Edit **src/AppBundle/Resources/views/Product/index.html.twig**
  Add `<th>Stocks</th>` after `<th>Description</th>`
  Add `<td>{{ render(controller('AppBundle:Warehouse:productStocks', { 'productId': entity.id })) }}</td>` after `<td>{{ entity.description }}</td>`

6.4 Catalog service

We want to start implementing a catalog service. We will need doctrine service for this class as well. Is also probable that other services might have the same need in the future. We will move the commun logic into a super class.

- Create **src/AppBundle/Service/AbstractDoctrineAware.php**

```php
<?php

namespace AppBundle\Service;

use Doctrine\Bundle\DoctrineBundle\Registry;
use Doctrine\ORM\EntityManager;

class AbstractDoctrineAware
{
    const ID = 'app.doctrine_aware';

    /**
     * @var Registry
     */
    protected $doctrine;

    /**
     * @var EntityManager
     */
    protected $entityManager;

    public function __construct(Registry $doctrine)
    {
        $this->doctrine = $doctrine;
        $this->entityManager = $doctrine->getManager();
    }
}
```

- Edit **src/AppBundle/Service/WarehouseService.php**

```php
<?php

namespace AppBundle\Service;

use AppBundle\Entity\ProductStock;
use AppBundle\Entity\Warehouse;

class WarehouseService extends AbstractDoctrineAware
{

    const ID = 'app.warehouse';

    public function getAll()
    {
        return $this->entityManager
                        ->getRepository(Warehouse::REPOSITORY)
                        ->findAll();
    }
}
```

- Create **src/AppBundle/Service/CatalogService.php**

```php
<?php

namespace AppBundle\Service;

use AppBundle\Entity\Category;

class CatalogService extends AbstractDoctrineAware
{

    const ID = 'app.catalog';

    public function getCategories()
    {
        return $this->entityManager
                        ->getRepository(Category::REPOSITORY)
                        ->findAll();
    }

}
```

- Edit **src/AppBundle/Resources/config/services.yml**

```
services:
    app.listener.soft_delete:
        class: AppBundle\Event\Listener\SoftDelete
        tags:
            - { name: doctrine.event_listener, event: onFlush }

    app.doctrine_aware:
        class: AppBundle\Service\AbstractDoctrineAware
        arguments: [@doctrine]
        abstract: true

    app.warehouse:
        class: AppBundle\Service\WarehouseService
        parent: app.doctrine_aware

    app.catalog:
        class: AppBundle\Service\CatalogService
        parent: app.doctrine_aware
```

Done, let's update the category controller to use the catalog service. Change indexAction as follwing

```
public function indexAction()
{
    $entities = $this->get(CatalogService::ID)->getCategories();

    return $this->render('AppBundle:Category:index.html.twig', array(
        'entities' => $entities,
    ));
}
```

and `use AppBundle\Service\CatalogService;`

6.5 Optional references

We will want to inject a *logger* service to AbstractDoctrineAware, but only if avilable.

- Edit **src/AppBundle/Resources/config/services.yml** and update app.doctrine_aware definition

```
app.doctrine_aware:
        class: AppBundle\Service\AbstractDoctrineAware
        arguments: [@doctrine, @?logger]
        abstract: true
```

- Update **src/AppBundle/Service/AbstractDoctrineAware.php**

```php
<?php

namespace AppBundle\Service;

use Doctrine\Bundle\DoctrineBundle\Registry;
use Doctrine\ORM\EntityManager;
use Symfony\Bridge\Monolog\Logger;

class AbstractDoctrineAware
{

    const ID = 'app.doctrine_aware';

    /**
     *
     * @var Registry
     */
    protected $doctrine;

    /**
     *
     * @var EntityManager
     */
    protected $entityManager;

    /**
     *
     * @var Logger
     */
    protected $logger;

    public function __construct(Registry $doctrine, Logger $logger = null)
    {
        $this->doctrine = $doctrine;
        $this->entityManager = $doctrine->getManager();
        if ($logger) {
            $this->logger = $logger;
        } else {
            $this->logger = new Logger('');
        }
    }

}
```

- Update *getProductStocks* from **src**/**AppBundle**/**Service**/**WarehouseService.php**

```
public function getProductStocks($productId)
{
    $stocks = $this->entityManager->
            getRepository(ProductStock::REPOSITORY)
            ->findBy(array('product' => $productId));
    if (empty($stocks)) {
        $this->logger->addNotice(
                sprintf('No stocks found for product %s', $productId)
            );
    }

    return $stocks;
}
```

6.6 Fixtures

6.6.1 DoctrineFixturesBundle

We want to create some mock data that is as close as possible to the production database. The best way is to actually use a copy of the production data (after removing or obfuscationg sensitive informations like user emails or passwords). That's however a luxury not everyone can have. Since we are developing a new application, we will create some data fixtures.

To isntall **DoctrineFixturesBundle** add `"doctrine/doctrine-fixtures-bundle": "2.2.0"` to the *require* section, in **composer.json** then run `composer update`.
To enable the bundle within the application, edit **app**/**AppKernel.php** and add a new item to the $bundles array in registerBundles()
`new Doctrine\Bundle\FixturesBundle\DoctrineFixturesBundle(),`

6.6.2 Category fixtures

We will write a data fixture to load the database with a simple category tree.

- Create **src/AppBundle/DataFixtures/ORM/CategoryFixtures.php**

```php
<?php
namespace AppBundle\DataFixtures\ORM;

use AppBundle\Entity\Category;
use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\Persistence\ObjectManager;

class CategoryFixtures extends AbstractFixture
{

    private $manager;
    private $categories = array(
        'computers' => array('servers', 'desktop',
            'laptops', 'components', 'periferals'),
        'phones and tablets' => array('phones', 'tablets', 'accessories'),
        'appliances' => array('coffe machines',
            'washing machines', 'blenders', 'juicers'),
        'video games' => array('consoles', 'games', 'accessories')
    );

    public function load(ObjectManager $manager)
    {
        $this->manager = $manager;
        $this->createAndPersistCategories();
        $this->manager->flush();
    }

    private function createAndPersistCategories()
    {
        foreach ($this->categories as $parent => $children) {
            $parentCategory = $this->createAndPersistCategory($parent);
            foreach ($children as $label) {
                $this->createAndPersistCategory($label, $parentCategory);
            }
        }
    }

    private function createAndPersistCategory($label, $parentCategory = null)
    {
        $category = new Category();
        $category->setLabel($label)->setParentCategory($parentCategory);
        $this->manager->persist($category);
        return $category;
    }
}
```

To load the new fixtures, simply run

```
$ app/console doctrine:fixtures:load
Careful, database will be purged. Do you want to continue Y/N ?y
  > purging database
  > loading AppBundle\DataFixtures\ORM\CategoryFixtures
```

If you have some records in category table the previous command may fail with an *Integrity constraint violation* exception. You should empty the category table and try to load the fixtures again.

6.6.3 Refactoring category fixtures

Before jumping into writing data fixtures for products, we should consider three facts:

- It will share same code with the category fixtures class.
  We will move some common logic into a commun parent class.
- We will need to access the categories created in the category fixtures from the product fixtures.
  We will add references to the categories and get them from product fixtures.
- We will need to ensure that the category fixtures are run before product fixtures.
  We will implement *OrderedFixtureInterface* and set the desired execution order.
- Doctrine is not able to delete the categories in the correct order to avoid integrity constraint violation. We will enforce the deletion before loading the fixtures. We will add this functionality to a new custom CategoryRepository.

- Edit **src/AppBundle/Entity/Category.php**
  Change `@ORM\Entity` to `@ORM\Entity(repositoryClass="AppBundle\Repository\CategoryRepository")` in the class annotation.

- Create **src/AppBundle/Repository/CategoryRepository.php**

```php
<?php

namespace AppBundle\Repository;

use AppBundle\Entity\Category;
use Doctrine\ORM\EntityRepository;

class CategoryRepository extends EntityRepository
{

    public function deleteAll()
    {
        $this->deleteCategories();
        $this->deleteCategories(false);
    }

    private function deleteCategories($hasParent = true)
    {
        $queryBuilder = $this->createQueryBuilder('category')
                ->delete(Category::REPOSITORY, 'category');

        if ($hasParent) {
            $queryBuilder->where('category.parentCategory IS NOT NULL');
        }
        $queryBuilder->getQuery()->execute();
    }

}
```

- Edit **src/AppBundle/Entity/Category.php** and update the `$product` annotation as following

```php
/**
 * @var \Doctrine\Common\Collections\Collection
 *
 * @ORM\ManyToMany(targetEntity="Product", inversedBy="category")
 * @ORM\JoinTable(name="category_has_product",
 *   joinColumns={
 *     @ORM\JoinColumn(
        name="category_id",
        referencedColumnName="id",
        onDelete="cascade"
        )
 *   },
 *   inverseJoinColumns={
 *     @ORM\JoinColumn(
        name="product_id",
        referencedColumnName="id",
        onDelete="cascade"
        )
 *   }
 * )
 */
private $product;
```

To update your database, with the new foreign key constraint, you can use the command :

```
$ app/console doctrine:schema:update --force
Updating database schema...
Database schema updated successfully! "32" queries were executed
```

- Create **src/AppBundle/DataFixtures/ORM/AbstractDataFixture.php**

```php
<?php

namespace AppBundle\DataFixtures\ORM;

use Doctrine\Common\DataFixtures\AbstractFixture;
use Doctrine\Common\DataFixtures\OrderedFixtureInterface;
use Doctrine\Common\Persistence\ObjectManager;

abstract class AbstractDataFixture extends AbstractFixture
implements OrderedFixtureInterface
{

    protected $manager;

    public function load(ObjectManager $manager)
    {
        $this->manager = $manager;
        $this->preLoad();
        $this->createAndPersistData();
        $this->manager->flush();
    }

    protected function preLoad()
    {

    }

    abstract protected function createAndPersistData();
}
```

- Refactor **src/AppBundle/DataFixtures/ORM/CategoryFixtures.php**

```php
<?php

namespace AppBundle\DataFixtures\ORM;

use AppBundle\Entity\Category;
use AppBundle\Repository\CategoryRepository;

class CategoryFixtures extends AbstractDataFixture
{
```

```
    private $categoriesCount = 0;
    private $categories = array(
        'computers' => array('servers', 'desktop',
            'laptops', 'components', 'periferals'),
        'phones and tablets' => array('phones', 'tablets', 'accessories'),
        'appliances' => array('coffe machines', 'washing machines',
            'blenders', 'juicers'),
        'video games' => array('consoles', 'games', 'accessories')
    );

    protected function createAndPersistData()
    {
        foreach ($this->categories as $parent => $children) {
            $parentCategory = $this->createAndPersistCategory($parent);
            foreach ($children as $label) {
                $this->createAndPersistCategory($label, $parentCategory);
            }
        }
    }

    private function createAndPersistCategory($label, $parentCategory = null)
    {
        $this->categoriesCount ++;
        $category = new Category();
        $category->setLabel($label)->setParentCategory($parentCategory);
        $this->manager->persist($category);
        $this->setReference(
            sprintf('category_%s', $this->categoriesCount),
            $category
          );

        return $category;
    }

    protected function preLoad()
    {
        /* @var $categoryRepository CategoryRepository */
        $categoryRepository = $this->manager->getRepository(Category::REPOSITORY);
        $categoryRepository->deleteAll();
    }

    public function getOrder()
    {
        return 1;
    }

}
```

From now on, we will load the fixtures using the *amend* option.

```
$ app/console doctrine:fixtures:load --append
  > loading [1] AppBundle\DataFixtures\ORM\CategoryFixtures
```

> I intentionally introduced the *refactoring* above. You should continuously refine and clean your code design, to keep it easily maintainable. A good thing is that the more experience you get refactoring your code, the less frequent you will need to refactor. You will also develop a *code smell* eye, you will be able to spot immediately a class that is acting on many abstraction layers, and how to split it in better hierarchy. This was a short word about code refactoring, if you want to read more about the subject, I recommend two invaluable books *Refactoring: Improving the Design of Existing Code* by *Martin Fowler* and *Clean Code A Handbook of Agile Software Craftsmanship* by *Robert C. Martin*

6.6.4 Product fixtures

Let's first prepare the logic that deletes all the products.

- Create **src/AppBundle/Repository/ProductRepository.php**

```
<?php

namespace AppBundle\Repository;

use AppBundle\Entity\Product;
use Doctrine\ORM\EntityRepository;

class ProductRepository extends EntityRepository
{

    public function deleteAll()
    {
        $this->createQueryBuilder('product')
                ->delete(Product::REPOSITORY)
                ->getQuery()
                ->execute();
    }

}
```

- Edit **src/AppBundle/Entity/Product.php**
  Change `* @ORM\Entity`
  into
  `* @ORM\Entity(repositoryClass="AppBundle\Repository\ProductRepository")`
  in the class annotation.

For the product fixtures we have a new requirement. We should be able to set the number of products per category as a parameter.

- Create **src/AppBundle/DataFixtures/ORM/ProductFixtures.php**

```
<?php
namespace AppBundle\DataFixtures\ORM;
```

```php
namespace AppBundle\DataFixtures\ORM;

use AppBundle\Entity\Category;
use AppBundle\Entity\Product;
use AppBundle\Repository\ProductRepository;
use OutOfBoundsException;
use Symfony\Component\DependencyInjection\ContainerAwareInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

class ProductFixtures extends AbstractDataFixture implements ContainerAwareInterface
{

    private $productsPerCategory;

    private function createAndPersistProducts(Category $category)
    {
        for ($i = 1; $i <= $this->productsPerCategory; $i++) {
            $product = new Product();
            $product->setCode(sprintf('code_%s_%s', $category->getId(), $i))
                    ->setTitle(
                      sprintf('title %s %s %s',$category->getId(),$i,uniqid())
                    )
                    ->setDescription(sprintf('product description %s', $i));
            $category->addProduct($product);
            $this->manager->persist($product);
        }
    }

    protected function createAndPersistData()
    {
        for ($i = 1; true; $i++) {
            try {
                /* @var $category Category */
                $category = $this->getReference(sprintf('category_%s', $i));
                $this->createAndPersistProducts($category);
            } catch (OutOfBoundsException $exception) {
                break;
            }
        }
    }

    protected function preLoad()
    {
        /* @var $productRepository ProductRepository */
        $productRepository = $this->manager->getRepository(Product::REPOSITORY);
        $productRepository->deleteAll();
    }

    public function setContainer(ContainerInterface $container = null)
    {
        $fixturesConfig = $container->getParameter('fixtures');
        $this->productsPerCategory = $fixturesConfig['products_per_category'];
    }

    public function getOrder()
```

```
    {
        return 2;
    }

}
```

- Edit **app/config/parameters.yml.dist**
  Add a new key under *parameters* in **app/config/parameters.yml**

```
fixtures:
        products_per_category: #number
```

The previous file is not interpreted by the framework. Is very helpful to contain the expected parameter keys so when one developer adds a new parameter, it becomes visible to his team members as soon as he pushes his changes to a commun remote.

Add a new key under *parameters* in **app/config/parameters.yml**

```
fixtures:
        products_per_category: 100
```

You can load the new fixtures in your database.

```
$ app/console doctrine:fixtures:load --append
  > loading [1] AppBundle\DataFixtures\ORM\CategoryFixtures
  > loading [2] AppBundle\DataFixtures\ORM\ProductFixtures
```

6.6.5 Homework

1. Create a JsonRpcServer service that will expose some methods of some services.
   The methods and services to expose should be configurable
2. Add to warehouse service a method
   `moveProductStock($productId, $quantity, $fromWarehouseId, $toWarehouseId)`