Dependency Injection and Magento

by Tim Bezhashvyly
Openstream Internet Solutions

About Me

Tim Bezhashvyly

Certified Magento Developer

12 years of PHP

3 years of Magento

boy-named-sue

tim-openstream

Dependency injection ..

Dependency injection ..

.. is a technique to reduce coupling.

Dependency injection ..

.. is a technique to reduce coupling.

What is coupling?

Dependency injection ..

.. is a technique to reduce coupling.

What is coupling?
Why it is bad?

Coupled code

```
class A
{
    protected $b;

    public function __constructor()
    {
        $this->b = new B();
    }
}

$a = new A();
```

Fuck Coupling!

How?

Factory!

Factory

```
class A
{
    protected $b;

    public function __constructor($factory)
    {
        $this->b = $factory->getWhatINeed();
    }
}

$factory = new Factory();
$a = new A($factory);
```

Registry!

Registry

```
class A
{
    protected $b;

    public function __constructor($registry)
    {
        $this->b = $registry->b;
    }
}

$registry = new Registry();
$registry->b = new B();
$a = new A($registry);
```

Service Locator!

Service Locator

```
class A
{
    protected $b;

    public function __constructor($serviceLocator)
    {
        $this->b = $serviceLocator->somethingUseful;
    }
}

$serviceLocator = new ServiceLocator();
$serviceLocator->getSomethingUseful('B');
$a = new A($serviceLocator);
```

Service Locator

```
class A
{
    protected $b;

    public function __constructor($serviceLocator)
    {
        $this->b = $serviceLocator->somethingUseful;
    }
}

$serviceLocator- = new ServiceLocator-();
$serviceLocator->getSomethingUseful('B');
$a = new A($serviceLocator);
```

Service Locator

```
class A
{
    protected $b;

    public function __constructor($serviceLocator)
    {
        $this->b = $serviceLocator->somethingUseful;
    }
}

$serviceLocator- = new ServiceLocator-();
$serviceLocator->getModel('B');
$a = new A($serviceLocator);
```

Dependency Injection?

Coupled code

```
class A
{
    protected $b;

    public function __constructor()
    {
        $this->b = new B();
    }
}

$a = new A();
```

Dependency Injection

```
class A
{
    protected $b;

    public function __constructor(B $b)
    {
        $this->b = $b;
    }
}

$b = new B();
$a = new A($b);
```

Full Decoupling!

Benefits of Dependency Injection

- Decouple classes from their dependencies

Benefits of Dependency Injection

- Decouple classes from their dependencies

- More reusable code

Benefits of Dependency Injection

- Decouple classes from their dependencies

- More reusable code

- More readable code

Benefits of Dependency Injection

- Decouple classes from their dependencies

- More reusable code

- More readable code

- Easy unit testing

Benefits of Dependency Injection

- Decouple classes from their dependencies

- More reusable code

- More readable code

- Easy unit testing

- Automatic ability to add new functionality

Dependency Injection

```
class A
{
    protected $b;

    public function __constructor(B $b)
    {
        $this->b = $b;
    }
}

$b = new B();
$a = new A($b);
```

Dependency Injection Container

Dependency Injection Container

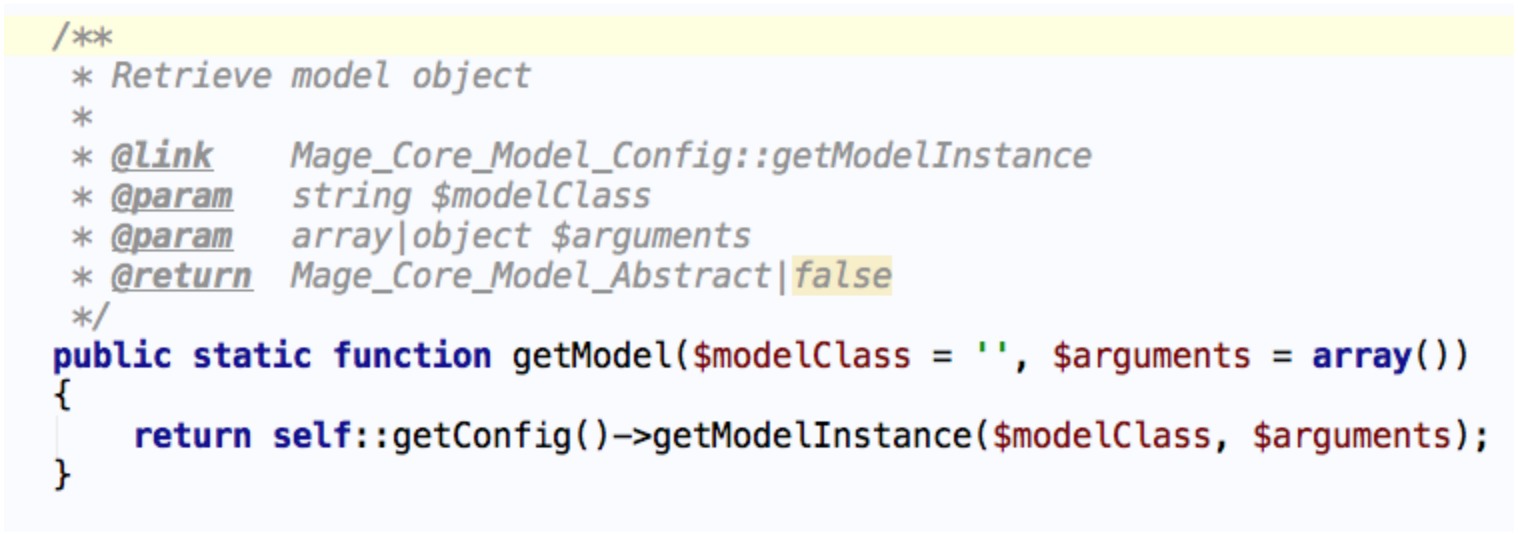
```
class Container
{
    public function getB()
    {
        return new B();
    }
    public function getA()
    {
        $b = $this->getB();
        return new A($b);
    }
}
```

Dependency Injection Container

```
$container = new Container();
$a = $container->getA();
```

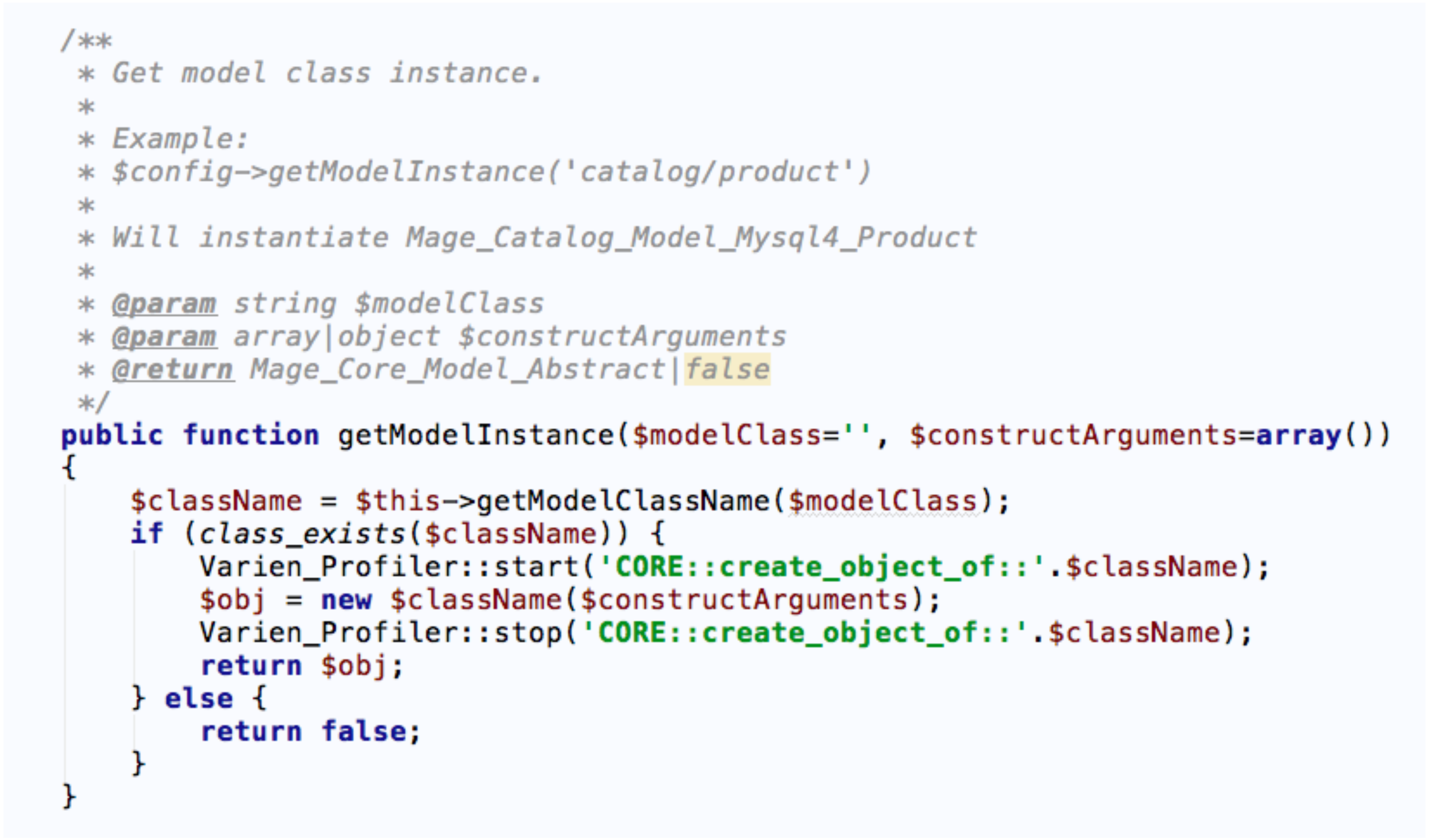
Magento

Magento 1.x.x.x



```php
/**
 * Retrieve model object
 *
 * @link    Mage_Core_Model_Config::getModelInstance
 * @param   string $modelClass
 * @param   array|object $arguments
 * @return  Mage_Core_Model_Abstract|false
 */
public static function getModel($modelClass = '', $arguments = array())
{
    return self::getConfig()->getModelInstance($modelClass, $arguments);
}
```

Magento 1.x.x.x



```php
/**
 * Get model class instance.
 *
 * Example:
 * $config->getModelInstance('catalog/product')
 *
 * Will instantiate Mage_Catalog_Model_Mysql4_Product
 *
 * @param string $modelClass
 * @param array|object $constructArguments
 * @return Mage_Core_Model_Abstract|false
 */
public function getModelInstance($modelClass='', $constructArguments=array())
{
    $className = $this->getModelClassName($modelClass);
    if (class_exists($className)) {
        Varien_Profiler::start('CORE::create_object_of::'.$className);
        $obj = new $className($constructArguments);
        Varien_Profiler::stop('CORE::create_object_of::'.$className);
        return $obj;
    } else {
        return false;
    }
}
```

Magento 1.x.x.x

All singletons and helpers are stored in a registry.

So you can register your own object in a registry using key

_singleton/class_alias

and

_helper/class_alias

DiC in ZF2

DiC in Zend Framework 2

```
class A
{
    protected $b;

    public function __constructor(B $b)
    {
        $this->b = $b;
    }
}

$b = new B();
$a = new A($b);
```

DiC in Zend Framework 2

```
$di = new Zend \ Di \ Di();
$a = $di->get('A');
/*            or           */
$a = $di->newInstance('A');
```

DiC in Zend Framework 2

```
$di = new Zend \ Di \ Di();
$a = $di->get('A', array(...));
/*            or            */
$a = $di->newInstance('A', array(...));
```

Preferences

app/code/core/Mage/Core/etc/config.xml

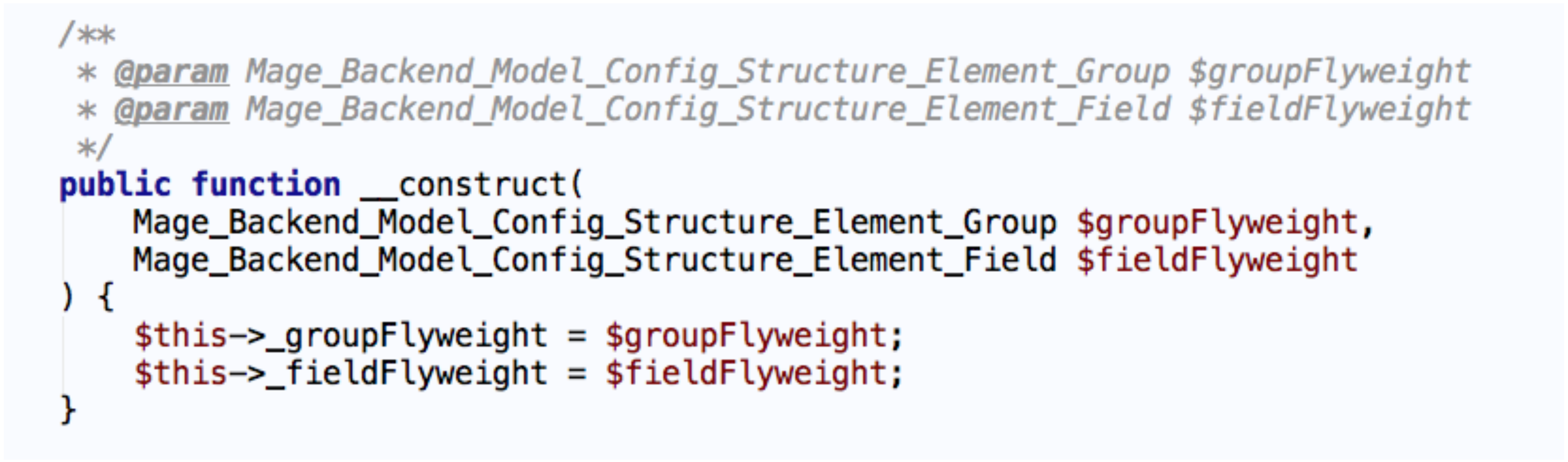
```
<adminhtml>
    ...
    <di>
        <preferences>
            <Mage_Core_Model_Url>Mage_Backend_Model_Url</Mage_Core_Model_Url>
        </preferences>
    </di>
    ...
</adminhtml>
```

Parameters

app/code/core/Mage/Backend/etc/config.xml

```
<global>
    <di>
        <Mage_Backend_Model_Config_Structure_Element_Iterator_Field>
            <parameters>
                <groupFlyweight>Mage_Backend_Model_Config_Structure_
            Element_Group_Proxy</groupFlyweight>
            </parameters>
            <shared>0</shared>
        </Mage_Backend_Model_Config_Structure_Element_Iterator_Field>
    </di>
</global>
```

Mage_Backend_Model_Config_Structure_Element_Iterator_Field



```php
/**
 * @param Mage_Backend_Model_Config_Structure_Element_Group $groupFlyweight
 * @param Mage_Backend_Model_Config_Structure_Element_Field $fieldFlyweight
 */
public function __construct(
    Mage_Backend_Model_Config_Structure_Element_Group $groupFlyweight,
    Mage_Backend_Model_Config_Structure_Element_Field $fieldFlyweight
) {
    $this->_groupFlyweight = $groupFlyweight;
    $this->_fieldFlyweight = $fieldFlyweight;
}
```

app/code/core/Mage/Backend/etc/config.xml

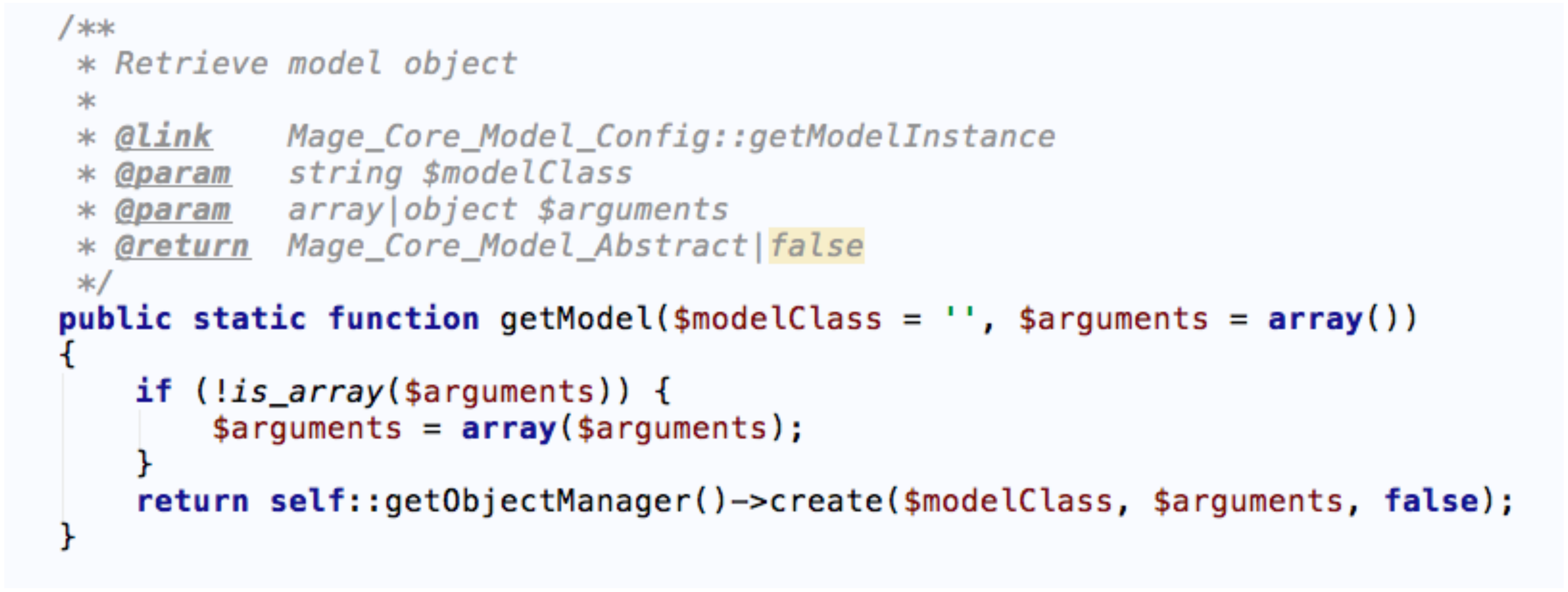
```
<global>
    <di>
        <Mage_Backend_Model_Config_Structure_Element_Iterator_Field>
            <parameters>
                <groupFlyweight>Mage_Backend_Model_Config_Structure_
            Element_Group_Proxy</groupFlyweight>
            </parameters>
            <shared>0</shared>
        </Mage_Backend_Model_Config_Structure_Element_Iterator_Field>
    </di>
</global>
```

Aliases

app/code/core/Mage/Backend/etc/config.xml

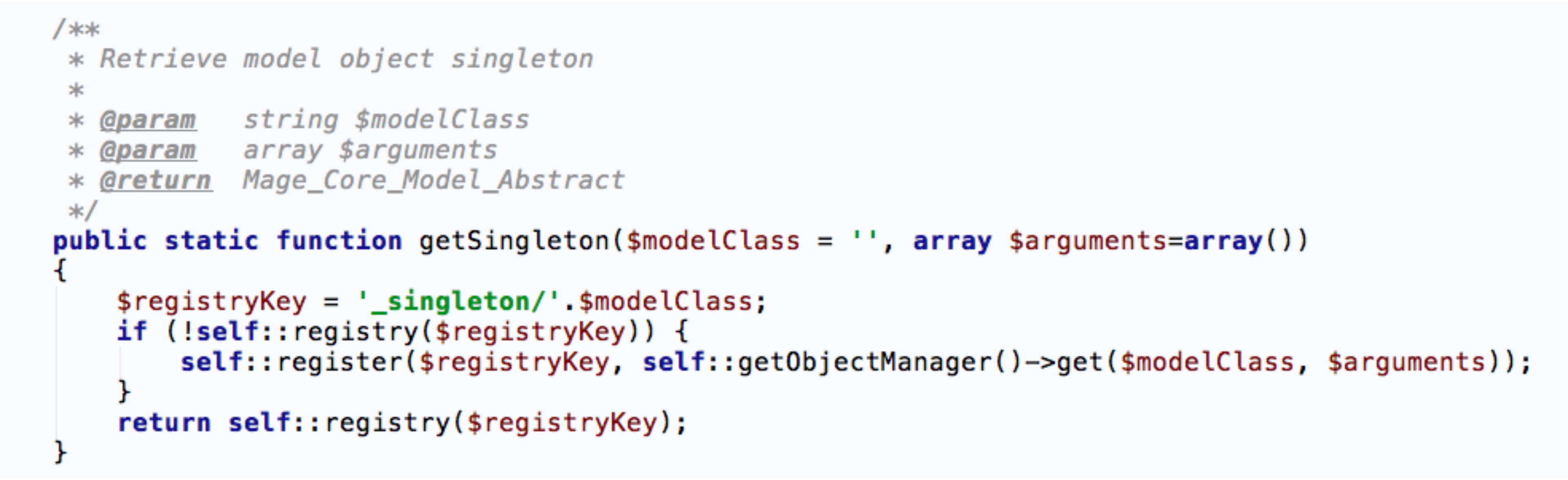
```
<di>
    <alias>
        <class_from>class_to</class_from>
    </alias>
</di>
```

Mage::getModel()



```php
/**
 * Retrieve model object
 *
 * @link    Mage_Core_Model_Config::getModelInstance
 * @param   string $modelClass
 * @param   array|object $arguments
 * @return  Mage_Core_Model_Abstract|false
 */
public static function getModel($modelClass = '', $arguments = array())
{
    if (!is_array($arguments)) {
        $arguments = array($arguments);
    }
    return self::getObjectManager()->create($modelClass, $arguments, false);
}
```

Mage::getSingleton()



```php
/**
 * Retrieve model object singleton
 *
 * @param   string $modelClass
 * @param   array $arguments
 * @return  Mage_Core_Model_Abstract
 */
public static function getSingleton($modelClass = '', array $arguments=array())
{
    $registryKey = '_singleton/'.$modelClass;
    if (!self::registry($registryKey)) {
        self::register($registryKey, self::getObjectManager()->get($modelClass, $arguments));
    }
    return self::registry($registryKey);
}
```