
Design Matters LP 1
Part-Three : Forever Single
Dcoder Team Presents

Information

Article : Design Matters LP 1

Authors : Mhmv and Joulook

Prerequisite : Object-Oriented Concepts

Published By : Dcoder Team

Follow us on Telegram : https://t.me/de_coder

The Cover of Article is Adapted From “Design Patterns Explained Simply By Alexander Shvets” Cover Page

Part-Three : Forever Single

Single Life Better Life

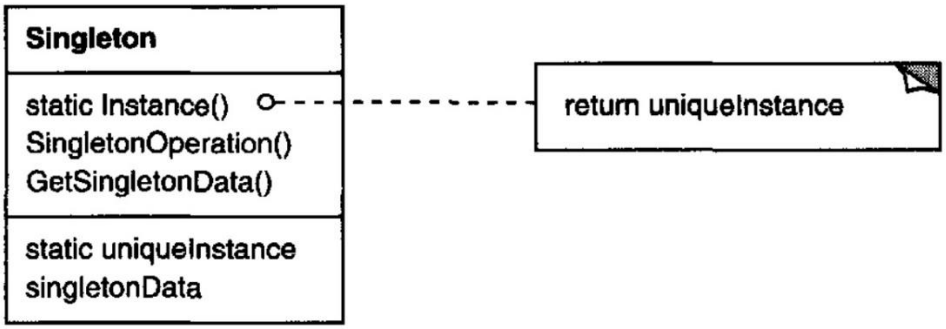
محال ممکن است که در دنیای Object-Oriented تا به حال با موضوع Instantiation مواجه نشده باشید . از همان ابتدای یاد گیری این دنیا، شما Class ساخته اید و از آن به هر تعداد و اندازه ای که می خواستید Object ساخته اید و به عبارت فنی Instance های مختلفی از Class های دلخواه تان ساخته اید. اما تا به حال این مسئله به ذهنتان خطور کرده است که ممکن است موقعیتی پیش بیاید که ما تنها باید فقط یک Object از یک Class بسازیم و باید جلوی این که بشود از یک Class بیش از یک Object ساخت را بگیریم ؟ اصلا آیا به نظر شما موقعیتی پیش می آید که به این محدودیت برسیم ؟ و آیا Object-Oriented این امکان را می تواند برای ما فراهم کند ؟ و به راستی باید این سوال را از خود مطرح کنیم که ما چه اندازه توانسته ایم Object-Oriented را درک کنیم و آن را بفهمیم ؟

Real World Examples

اگر بخواهیم چند مثال از دنیای واقعی بزنیم که در آن ها دقیقا باید یک Object از Class مربوطه داشته باشیم می توانیم به این موارد اشاره کنیم : ممکن است در یک سیستم و یا یک مجموعه ای ما چندین دستگاه Printer داشته باشیم ، اما فقط و فقط یک Printer Spooler داریم . یا در یک Operating System فقط یک File System داشته باشیم یا فقط یک Window Manager داشته باشیم یا زمانی که میخواهیم تنها یک نقطه دسترسی به یک Database Engine داشته باشیم . گروه GOF برای حل تمامی این مسائل Design Pattern ای را تعریف کردند که نام آن Singleton می باشد و جزو Design Pattern های Creational دسته بندی شده است حال به تعریف دقیق این Design Pattern می پردازیم

Get Familiar with Singleton Design Pattern

این Design Pattern تضمین می کند که یک Class فقط یک Instance دارد و یک Global Point برای دسترسی به آن تهیه می کند



در شکل فوق می توانید Structure این Design Pattern را در قالب یک Class Diagram مشاهده نمایید

How Implement This Philosophy

حال چگونه میتوانیم از این Design Pattern برای حل یک مسئله که Problem Domain آن منطبق بر این Design Pattern است استفاده کنیم و آن را پیاده سازی کنیم ؟ برای این کار روش های مختلفی وجود دارد که ما به بررسی برخی از این روش ها روی یک مثال می پردازیم

Lazy Initialization

فرض کنید ما برای شبکه ای خاص می خواهیم تنها و تنها یک Admin داشته باشیم برای اینکه فقط یک Object از کلاس Admin داشته باشیم می آییم ابتدا داخل کلاس Admin یک Private Variable قرار میدهیم که یک Object را دسترس پذیر با استفاده از یک Method می کند. و برای بهینگی کار قسمت new کردن Object را هم در Method قرار میدهیم تا زمانی Object بتواند new شود که آن Method به اصطلاح Call می شود

```
public class Admin {
	private static Admin instance;

	public static Admin getInstance(){
		if(this.instance == null){
			this.instance = new Admin();
		}
		return this.instance;
	}

}
```

حال باید جلوی ساخت Object توسط Class را به طور کل بگیریم و برای این کار باید Constructor این Class را Private کنیم

```
public class Admin {
	private static Admin instance;

	private Admin(){

	}

	public static Admin getInstance(){
		if(this.instance == null){
			this.instance = new Admin();
		}
		return this.instance;
	}

}
```

خب تا به اینجای کار همه چیز خوب پیش می رود و به نظر می رسد که اگر ما از این Class استفاده بکنیم به هیچ وجه مشکلی برای ما پیش نخواهد آمد و همیشه یک Object در طول Life-Time برنامه مان از کلاس Admin خواهیم داشت... اما اینطور نیست!

پیشنهاد ما به دوستانی که درس Operating System را در دانشگاه های خود گذرانده اند این است که با دید سیستم عاملی بار دیگر برگردند کد بالا رو ببینند و روی آن فکر کنند که در چه صورتی Code فوق نمی تواند مفهوم Singleton را پیاده سازی کند و سپس به ادامه خواندن این مقاله بپردازند

Time to Rise The Side of Truth

شاید بتوان گفت اینجا یکی از جاهایی است که می توان دانش سیستم عاملی خود را محک زد و به این قضیه پی برد که براستی آن استادی که به شما سیستم عامل یاد داده است تا چه اندازه کارش را درست انجام داده و یا آن کتابی که شما به عنوان مرجع درس سیستم عامل انتخاب کرده اید تا چه اندزه توانسه آن چیز هایی که نیاز بوده را به شما منتقل کند

و شاید اینجا جای خوبی باشد باشد که جواب آن دسته از افراد بی منطقی که می گویند خواندن درس های آکادمیک دانشگاه به طور درست و اصولی هیچ کمکی به یادگیری دانشجو برای کار در بازار کار نمی کند را داد .

Now... What is the issue?

بگذریم به ادامه بحث می پردازیم... مشکل زمانی بروز پیدا می کند که ما بخوایم برنامه مان را به صورت Multi-Thread بنویسم . فرض کنید دو Thread با نام های T1 و T2 داریم . ابتدا T1 متدِ getInstance را Call می کند بعد از این که شرط if را چک کرد و جواب true از درست بودن شرط گرفت Interrupt بخورد و سپس T2 همان Method را Call کند به طور Commit آن را انجام دهد . وقتی T2 آن Method را Call کرد و سپس به طور کامل انجام داد ، Object ما ساخته شده است حال Interrupt از روی T1 برداشته می شود چون قبلا شرط را بررسی کرده بود دیگر آن را بررسی نمی کند و می آید دومین Object را از کلاس Admin می سازد و فلسفه Singleton را ضایع می کند . در کتاب ها ی مرجع درس Operating System راجع به بررسی این معضلات و پیشنهاد راه کار در فصل هایی تحت عنوان INTERPROCESS COMMUNICATION و یا IPC پرداخته می شود که صحبت کردن راجع به آن در این مقاله جایگاهی ندارد اما Java برای حل این مشکل راه کاری دارد و آن این است که بیایم از کلمه کلیدی synchronized برای Method مورد نظر استفاده کنیم اتفاقی که می افتد این است که زمانی که یک Thread یک Synchronized Method

را Call می کند هیچ Thread دیگری نمی تواند تا تمام نشدن کار Thread ای که Method را Call کرده است بتواند آن Method را Call کند و مشکل ما در این قسمت حل می شود

```
public class Admin {
	private static Admin instance;
	private Admin(){
	}
	public static synchronized Admin getInstance(){
		if(this.instance == null){
			this.instance = new Admin();
		}
		return this.instance;
	}
}
```

راه کار های دیگری مانند double-checked locking وجود دارد که میتواند راه کار بهینه تری باشد و می توانید برای یادگیری بیشتر راجع به این موضوع به کتاب هایی که در بخش Bibliography معرفی می شود مراجعه بکنید

Eager initialization

در این روش کافیست مانند روش بالا عمل کنیم و به جای اینکه بیایم Object را در یک Method ، new کنیم در همان قسمت Declaration این کار را انجام دهیم و چیزی شبیه به code زیر را بنویسیم

```
public class Admin {
	private static Admin instance = new Admin();
	private Admin(){
	}
	public static Admin getInstance(){
		return this.instance;
	}
}
```

The Other Ways

روش های دیگری برای پیاده سازی این Design Pattern وجود دارد و از آنجایی که هدف ما ترقیب شما برای مطالعه کتب رفرنس و زبان اصلی می باشد از شرح آن روش ها خودداری می کنیم و یادگیری آن روش ها را بر عهده شما می گزاریم و میتوانید با مطالعه کتاب های معرفی شده در بخش Bibliography و یا جسجتو در سایت هایی که به زبان اصلی توضیح دادند آن ها را یاد بگیرید

این هم لیستی از دیگر روش هایی که می شود این Design Pattern را پیاده سازی کرد :

1. Static block initialization
2. Bill Pugh Singleton Implementation
3. Using Reflection to destroy Singleton Pattern
4. Enum Singleton
5. Serialization and Singleton

Bounds

تیم Dcoder برای شما یک پروژه تهییه کرده است به سناریو زیر که میتوانید سورس کد پیاده سازی این سناریو به دو زبان Java و C# را از کانال ما دریافت کنید

فرض کنید یک Database Engine داریم به نام ADABAS و میخواهیم از آن در یک سیستم ثبت ثبت دقیقه ای آب و هوا استفاده کنیم . و میخواهیم این DBMS تنها یک Admin داشته باشد ، Class Diagram این سناریو را میتوانید در شکل زیر ببینید

Bibliography

1. **Design Patterns Elements of Reusable Object-Oriented Software -Gmma and Helm and Johnson and Vlissides -Addison Wesley**

Download Link : https://t.me/debrary/525

2. **Java Design Patterns -Rohit Joshi -Exelixis Media**

Download Link : https://t.me/debrary/529

جهت دریافت قسمت های بعدی به کانال زیر مراجعه کنید

https://t.me/de_coder