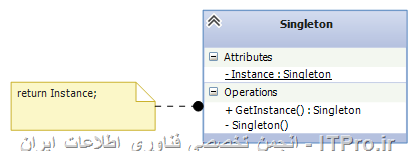
الگوی Singleton

این الگو یکی از الگوهای سازنده (creational) است و در ساختن اشیا از کلاس ها کاربرد دارد. این الگو یکی از معروف ترین الگوهای مطرح شده توسط Gang of Four است.   
**نکته: کلمه کاربر که در این سری مقالات استفاده می شود به کلاس هایی اطلاق می شود که از کلاس مورد بحث استفاده می کنند و منظور تنها کاربر انسانی نیست و ممکن است که یک زیرسیستم یا یک کلاس و یا حتی تابع main برنامه باشد.**

حالتی را در نظر بگیرید که کلاسی دارید که می خواهید در کل برنامه خود فقط یک شی از این کلاس ساخته شود و اجازه ساخته شدن نمونه جدید به استفاده کنندگان از این کلاس داده نشود. این عمل در مواقعی بسیار لازم و ضروری است. به طور مثال حالتی را درنظر بگیرید که در برنامه کلاسی وجود دارد که واسط تنظیمات برنامه است یعنی برنامه هنگام لود شدن تنظیمات خود را در آن بارگزاری می کند. حال اگر این شی یک شی در کل برنامه یکتا نباشد و هر بخش از برنامه که بخواهد تنظیمات را بخواند و یا تغییری در آن ایجاد کند مجبور باشد که یک شی جدید از کلاس تنظیمات بسازد مدیریت این اشیا مشکل می شود و همچنین باعث نابه سامانی در تنظیمات خواهد شد. راه حل این مسئله این است که از همان اول کلاس به گونه ای تعریف شود که اجازه ساختن نمونه جدید به کاربران ندهد و تنها کاربران بتوانند یک نمونه از کلاس به وجود آورند.   
اگر به برخی از قسمت های سیستم عامل ویندوز هم نگاهی بکنیم متوجه می شویم که چنین مکانیزمی در آن تعبیه شده است مثلاً یک سیستم فایل برای مدیریت فایل ها وجود دارد و یا با وجود این که ممکن است چندین پرینتر به کامپیوتر متصل باشد ولی فقط یک printer spooler در سیستم وجود دارد.   
برای به وجود آوردن چنین مکانیزمی شاید اولین راه حل این باشد که یک شی سراسری در کل برنامه تعریف شده و هروقت که برنامه ای بخواهد از این کلاس استفاده کند از شی سراسری تعریف شده استفاده کند. اما مشکلی که این راه حل دارد این است که این راه حل مانع ساخت شی جدید از کلاس نمی شود. یک راه حل بهتر این است که خود کلاس را مسئول این کار قرار دهیم که مواظب باشد که یک نمونه بیشتر نداشته باشد. این کار را الگوی Singleton انجام می دهد.   
  
روش کار این الگو به این صورت است که یک نمونه از خود کلاس در داخل کلاس ساخته می شود و هر وقت که کلاس های دیگر خواستند از این کلاس استفاده کنند تنها می توانند به این نمونه ساخته شده دسترسی داشته باشند و نمی توانند از کلاس نمونه سازی کنند.   
ساختار این الگو به این شکل است:



همانطور که در شکل می بینید یک فیلد به نام instance داریم که از خود کلاس است و این فیلد تنها الگویی که قرار است ساخته شود را در خود نگهداری می کند. سازنده(constructor) این کلاس از نوع private بوده بنابراین نمی توان از این کلاس یک نمونه جدید توسط عملگر new ایجاد کرد. تنها راه ساختن یک نمونه از این کلاس فراخوانی متد GetInstance است.

ویژگی های این الگو

این الگو باعث می شود که دسترسی به نمونه های کلاس کنترل شود و تعداد خاص از نمونه ها ایجاد شود.   
با استفاده از این الگو نیاز به ساختن متغیر های سراسری در فضای namespace کمتر شده و از اشغال فضای namespace توسط متغیرهای سراسری جلوگیری می کند.   
  
کد پیاده سازی این الگو در زبان C# به شکل زیر است 

public sealed class Singleton

{

private Singleton()

{

}

private readonly static Singleton instance=new Singleton();

public static Singleton UniqueInstance { get { return instance; } }

}

همانطور که درکد مشاهده می شود یک بار یک نمونه از کلاس ساخته می شود و تنها راه دسترسی به کلاس استفاده از ویژگی UniqueInstance می باشد که نمونه ساخته شده را برمی گرداند. در این نوع پیاده سازی از همان ابتدا یک نمونه از کلاس ساخت شده است و استفاده می شود ولی می توان این نمونه را وقتی ساخت که درخواستی برای اولین باربه کلاس ارسال شود که به این شیوه **Lazy instantiate** گفته می شود. کد این روش به شکل زیر است. 

public class LazySingleton

{

private LazySingleton(){}

private static LazySingleton instance;

public static LazySingleton UniqueInstance

{

get { return instance!=null?instance:new LazySingleton();}

}

}

کدی که آورده شده است تا حدی می تواند هدف این الگو را که ساخته شدن یک نمونه از کلاس باشد را برآورده کند ولی اگر برنامه دارای چند thread باشد ممکن است در مواردی بیشتر از یک نمونه از کلاس ساخته شود که در اصطلاح گفته می شود برنامه **Thread Safe** نیست که به جای کد بالا از کد پایین استفاده می کنیم. 

public class Singleton

{

private Singleton(){}

class SingletonCreator

{

static SingletonCreator()

{

}

internal static readonly Singleton UniqueInstance = new Singleton();

}

public static Singleton UniqueInstance

{

get { return SingletonCreator.UniqueInstance; }

}

}

در کد بالا از یک کلاس داخلی برای ساخت نمونه از کلاس استفاده می شود که علاوه بر Thread safe بودن خاصیت Lazy instatiate هم دارد.   
حال اگر نیاز باشد که از یک خانواده کلاس ها تنها یک نمونه ایجاد شود پیاده سازی این الگوی کمی متفاوت تر خواهد شد. برای مثال تصور کنید که یک کلاس میوه دارید که از آن کلاس های دیگری مانند سیب و پرتقال مشتق شده اند و لازم است که در برنامه فقط یک عدد میوه وجود داشته باشد یعنی اگر یک سیب وجود دارد دیگر نتوان میوه ی دیگری(چه سیب و چه پرتقال) ایجاد کرد. به منظور پیاده سازی چنین روشی به شکل زیر عمل می کنیم. 

public class SingleFruit

{

public string name;

protected SingleFruit()

{ }

protected static SingleFruit Instance ;

public static SingleFruit UniqueInstance

{

get { return Instance; }

}

}

public class SingleApple : SingleFruit

{

private SingleApple()

{

name = "Apple";

}

public static SingleFruit UniqueInstance

{

get { return Instance != null ? Instance :Instance= new SingleApple(); }

}

}

public class SingleOrange : SingleFruit

{

private SingleOrange()

{

name = "Orange";

}

public static SingleFruit UniqueInstance

{

get { return Instance != null ? Instance :Instance= new SingleOrange(); }

}

}

با توجه به کد بالا اگر به کلاس های فرزندان مقداردهی نشده باشد استفاده از کلاس پدر باعث خطای NullReference خواهد شد. و همچنین این کد Thread safe نمی باشد.   
  
  
الگوهای مرتبط با الگوی Singleton الگوهای Façade و Abstract Factory و بسیاری از الگوهای دیگر هستند که لازم است در برنامه از آنها فقط یک نمونه ساخته شده باشند که در زمان معرفی این الگوها توضیحات لازم ارایه خواهد شد.