عنوان: **پشتیبانی توکار از ایجاد کلاس‌های Singleton از دات نت 4 به بعد**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۰۲ ۱۲:۴۵
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, Threading

روش‌های زیادی برای ایجاد یک وهله‌ی Singleton وجود دارند. وهله‌ای که در طول عمر یک برنامه، تنها یکبار ایجاد شده و حفظ می‌شود. برای مثال شاید متداول‌ترین حالت آن که در بسیاری از کدها دیده می‌شود، تعریف یک متغیر استاتیک در کلاس، غیرعمومی تعریف کردن سازنده‌ی کلاس و وهله سازی این فیلد استاتیک در صورت نال بودن آن است:

```
public class WrongSingleton
    {
        static WrongSingleton _instance;

        WrongSingleton()
        {
        }

        public static WrongSingleton Instance
        {
            get { return _instance ?? (_instance = new WrongSingleton()); }
        }
    }
```

هرچند این روش کار می‌کند اما thread-safe نیست. به این معنا که ممکن است دو ترد در آن واحد به بررسی قسمت ?? _instance بپردازند و چون هنوز نال است، دوبار وهله سازی کلاس، با فراخوانی new WrongSingleton صورت خواهد گرفت و هر ترد در آن لحظه به وهله‌ی متفاوتی دسترسی خواهد داشت.
راه حل‌های زیادی برای رفع این مشکل با اعمال مباحث قفل گذاری تا نکات ریز مربوط به کامپایلر وجود دارند که لیست آن‌ها را در اینجا می‌توانید مطالعه کنید.

از دات نت 4 به بعد با معرفی الگوی Lazy ، امکان پیاده سازی lazy thread safe singletons به صورت توکار در دسترس می‌باشد. نمونه‌ای از آن در کدهای IoC Container معروفی به نام StructureMap بکار رفته‌است :

```
public class Container
    {
        // ...
    }

    public static class ObjectFactory
    {
        private static readonly Lazy<Container> _containerBuilder =
            new Lazy<Container>(defaultContainer, LazyThreadSafetyMode.ExecutionAndPublication);

        public static Container Container
        {
            get { return _containerBuilder.Value; }
        }

        private static Container defaultContainer()
        {
            return new Container();
        }
    }
```

در اینجا کلاس ObjectFactory یک وهله از کلاس Container را در اختیار مصرف کننده قرار می‌دهد؛ با این شرایط:
- چون این وهله توسط کلاس Lazy محصور شده‌است، صرفا در اولین بار دسترسی به آن، نمونه سازی خواهد شد. این مورد سبب کاهش مصرف حافظه‌ی برنامه و همچنین بالا رفتن سرعت برپایی اولیه‌ی آن می‌شود. بسیاری از اشیایی که در یک برنامه تعریف می‌شوند، شاید الزاما جهت ارائه راه حلی برای مساله‌ای خاص، مورد استفاده قرار نگیرند. تعریف آن‌ها به صورت Lazy، سربار نمونه سازی الزامی آن‌ها را حذف خواهد کرد و آن‌را به اولین بار استفاده از شیء مورد نظر، به تعویق خواهد انداخت.
- با استفاده از LazyThreadSafetyMode.ExecutionAndPublication، چندین ترد می‌توانند به خاصیت Container دسترسی پیدا کنند، اما تنها یکی از آن‌ها موفق به وهله سازی این کلاس خواهد شد. البته حالت ExecutionAndPublication، حالت پیش فرض

است و الزاما نیازی به ذکر آن نیست.

اینبار بازنویسی کلاس ابتدای بحث با توجه به نکات ذکر شده به صورت زیر خواهد بود:

```
public sealed class LazySingleton
    {
        private static readonly Lazy<LazySingleton> _instance =
            new Lazy<LazySingleton>(() => new LazySingleton(),
LazyThreadSafetyMode.ExecutionAndPublication);

        private LazySingleton()
        {
        }

        public static LazySingleton Instance
        {
            get { return _instance.Value; }
        }
    }
```

- در آن سازنده‌ی کلاس، خصوصی تعریف شده‌است.
- تنها وهله‌ی در دسترس کلاس، به صورت استاتیک و نمونه سازی کلاس، توسط کلاس Lazy با پارامتر LazyThreadSafetyMode.ExecutionAndPublication انجام می‌شود.
- علت استفاده از lambda در سازنده‌ی کلاس Lazy، امکان دسترسی به اعضای private کلاس، از طریق یک خاصیت static است.