کلاس‌های مقداری

این الگوی پیاده‌سازی برای جلوگیری از پراکنده شدن مقادیر متنی ثابت در سورس‌ها استفاده شده است. با استفاده از این الگو می‌توان تعدادی مقادیر مرتبط را به گونه‌ای در یک کلاس مرکزی تعریف کرد که کاربردی مانند انواع داده‌ای شمارشی (enumeration) داشته باشد. با توجه به این که انواع داده‌ای شمارشی فقط برای مقادیر عددی صحیح مورد استفاده هستند، با استفاده از این الگوی پیاده‌سازی می‌توان قابلیت مشابهی را برای انواع داده‌ای دیگر (متن، عدد اعشاری، تاریخ و...) با استفاده از امکانات شیئ‌گرای موجود در زبان C# پیاده‌سازی کرد.

هدف اصلی در استفاده از این الگو در کدهای موجود، آماده کردن زمینه‌های لازم برای پشتیبانی از واسط‌های کاربری چندزبانه و – در سناریوهای خاص – داده‌های چندزبانه بوده است. بیشترین موارد به‌کارگیری این الگو در کدهای موجود، در پروژه‌های زیر بوده است:

- SPPC.Framework.Common
- SPPC.Tadbir.Common

الگوی دورریختنی (disposable)

این الگوی پیاده‌سازی برای کنترل بهتر منابع حافظه سیستمی در کدهای اجرایی استفاده می‌شود. بسیاری از کلاس‌های موجود در پلتفرم دات‌نت، با استفاده از کدهای مدیریت‌نشده (unmanaged code) موجود پیاده‌سازی شده‌اند. بنابراین مصرف منابع حافظه آنها خارج از کنترل امکانات هوشمند مدیریت حافظه در دات‌نت (garbage collection) قرار می‌گیرد. برای کدهایی که از این امکانات هوشمند استفاده می‌کنند نیز می‌توان از این الگو استفاده کرد.

به‌عنوان یک روال کار امن و توصیه‌شده، بهتر است هنگام پیاده‌سازی یک کلاس جدید، هرگاه حداقل یکی از شرایط زیر صادق بود از این الگوی پیاده‌سازی استفاده شود:

۱. کلاس جدید یک یا چند عضو داده‌ای (data member) دارد که به‌صورت مستقیم یا غیرمستقیم اینترفیس IDisposable را پیاده‌سازی می‌کند.
۲. کلاس جدید یا یکی از کلاس‌های پایه آن، اینترفیس IDisposable را پیاده‌سازی می‌کند.

تعدادی از کلاس‌های موجود در فریم‌ورک جدید، این الگو را پیاده‌سازی کرده‌اند که شکل پیاده سازی در تمام موارد یکسان است. به‌عنوان مثال، کلاس TypeContainer در پروژه SPPC.Tadbir.Unity یک عضو داده‌ای از نوع IUnityContainer دارد و این اینترفیس مستقیما از اینترفیس IDisposable مشتق شده است. بنابراین الگوی disposable در این کلاس پیاده‌سازی شده است.

الگوی طراحی Singleton

یکی از الگوهای طراحی کلاسیک است که در سال ۱۹۸۴ توسط اریش گاما و همکارانش در کتاب بسیار معروفشان مطرح شد. این الگو در مواردی استفاده می‌شود که به‌دلایلی بخواهیم از یک کلاس یک و فقط یک نمونه در سراسر برنامه داشته باشیم. نوع پیاده‌سازی این الگو در بیشتر مواقع کاملا یکسان و قابل پیش‌بینی است: تابع سازنده با دسترسی خصوصی (private) تعریف شده و تنها نمونه کلاس در قالب یک عضو داده‌ای همان کلاس تعریف شده و پس از ایجاد و آماده‌سازی در تابع سازنده، در اختیار کلاس‌های خارجی قرار می‌گیرد.

این الگوی طراحی تا مدتی بسیار محبوب و مورد استفاده بود. ولی در سالهای اخیر، استفاده از این الگو به‌دلایل متعدد بسیار کمتر و از محبوبیت اولیه آن کاسته شده است. یکی از مهمترین دلایل این کاهش محبوبیت، مشکلات مربوط به پیاده‌سازی تست‌های واحد در مواردی است که کلاس‌های پیاده‌سازی شده با این الگو به‌عنوان وابستگی (dependency) در سایر کلاس‌ها استفاده می‌شوند. در این موارد، پیاده‌سازی کلاس‌های مقلد (mock class) برای این نوع کلاس‌ها مشکل خواهد بود. با این حال، هنوز در موارد خیلی خاص از این الگو استفاده می‌شود.

در کدهای موجود استفاده محدود و متمرکزی از این الگو شده است. به‌عنوان مثال، تعدادی از کلاس‌های کمکی موجود در فولدر Security در پروژه SPPC.Tadbir.Interfaces این الگوی طراحی را پیاده‌سازی می‌کنند. این کلاس‌های کمکی برای بالا بردن خوانایی کد هنگام تخصیص دسترسی امنیتی داخل کد، پیاده‌سازی شده‌اند.

کلاس‌های اطلاعاتی ساده (POCO)

این الگوی پیاده‌سازی بیشتر در ارتباط با ابزارهای نگاشت دیتابیسی (Object/Relational Mapper) مطرح و استفاده می‌شود. برای پیاده‌سازی کلاس‌هایی که توسط این ابزارها برای نگاشت اطلاعات جداول دیتابیسی استفاده می‌شوند، الگوهای مختلفی بکار می‌رود. یکی از آنها، الگوی کلاس‌های POCO (Plain-Old CLR Objects) است. مطابق این الگو، کلاس‌های متناظر با جداول اطلاعاتی، فقط اطلاعات فیلدهای جدول دیتابیسی را نگهداری کرده و تا حد امکان، عملیات کاربردی (مانند کنترل قواعد کسب‌وکار یا محاسبات) در آنها پیاده‌سازی نمی‌شود.

تمام کلاس‌های موجود در لایه‌های مدل و مدل نمایشی با استفاده از این الگو تولید یا پیاده‌سازی شده‌اند.