الگوهای طراحی در C# 5.0

مهندس سید منصور عمرانی

انتشارات پندار پارس

سرشناسه : عمرانی، سیدمنصور، 1356 -
عنوان و نام پدیدآور : الگوهای طراحی در C# 5.0/ سیدمنصور عمرانی.
مشخصات نشر : تهران : پندار پارس، 1392.
مشخصات ظاهری : 416ص.: مصور،جدول، نمودار.
شابک : 978-600-6529-42-4 220000 ریال:
وضعیت فهرست نویسی : فیپا
یادداشت : کتابنامه.
موضوع : سی شارپ (زبان برنامه‌نویسی کامپیوتر)
رده بندی کنگره : QA76/73 1392 82ع95س/
رده بندی دیویی : 133/005
شماره کتابشناسی ملی : 3262692

انتشارات پندارپارس

دفتر فروش: انقلاب، ابتدای کارگرجنوبی، کوی رشتچی، شماره 14، واحد 16 **www.pendarepars.com**

تلفن: 66572335 - تلفکس: 66926578همراه: 09122452348 **info@pendarepars.com**

نام کتاب : **الگوهای طراحی در C# 5.0**

ناشر : **انتشارات پندار پارس**

تالیف : **سید منصور عمرانی**

چاپ نخست : **مهر ماه 92**

شمارگان : **1000 نسخه**

طرح جلد : **فرزانه روزبهانی**

لیتوگرافی، چاپ، صحافی : **ترام‌سنج، فرشیوه، خیام**

قیمت : **22000 تومان** **شابک** : 978-600-6529-42-4

درباره‌ی نویسنده‌ی کتاب

سید منصور عمرانی برنامه‌نویس آزادی است که به صورت مستقل در دنیای برنامه‌نویسی وب فعالیت می‌کند. وی دانش‌آموخته‌ی رشته‌ی مهندسی کامپیوتر از دانشگاه علم و صنعت ایران است و با ۹ سال سابقه‌ی فعالیت در دنیای وب در زمینه‌ی فناوری‌های مایکروسافت، همواره انگیزه و علاقه‌ای ویژه به نویسندگی، نشر و آموزش داشته است.

علاقمندی‌های فنی او مبحث کامپایلر، زبان‌ها و سبک‌های برنامه‌نویسی و همچنین مطالعه و تحقیق در زمینه‌ی فناوری‌های وب بوده و داشتن روحیه‌ی تحلیل‌گر و ابداع مدل‌های جدید از خصلت‌های او به شمار می‌رود. او اعتقاد دارد دنیای آینده شاهد تحولات زیادی در زمینه‌ی سبک برنامه‌نویسی خواهد بود. منصور در اوقات فراغت خود به مطالعه، تماشای فیلم، ورزش و در صورت مهیا شدن فرصت مناسب به مسافرت می‌پردازد. اما ابداع مدل‌ها یا ایده‌های نرم‌افزاری جدید، چیز دیگری برای او است!

تقدیر و تشکر

با وجودی که بر روی جلد کتاب تنها نام نویسنده ذکر می‌شود، اما نویسنده‌ی هر کتابی در تهیه‌ی آن، مدیون افراد بیشماری است. در اینجا مایلم از برخی از دوستان و نزدیکانی که در مطالعه و بازبینی برخی از قسمت‌های کتاب نقش داشته‌اند به خاطر نظرات سازنده‌ی آنها تشکر کنم. بدینوسیله از آقایان سید محمد لقمان دستغیب، سعید خلفی‌نژاد، هومن فامیل‌روحانی و سرکار خانم فرناز طالبی قدردانی می‌نمایم.

از جناب آقای حسین یعسوبی، مدیر مسئول و صاحب امتیاز انتشارات پندارپارس به خاطر راهنمایی‌های ارزنده‌شان برای هماهنگی بیشتر متن اثر با زبان فارسی و زحمت‌هایشان در پیگیری و چاپ کتاب سپاسگزارم. همچنین از سرکار خانم فرزانه روزبهانی به خاطر پیاده‌سازی طرح روی جلد تشکر می‌نمایم.

و در پایان، قدردان پدر و مادرم هستم که همواره دلسوزی‌ها و محبت‌هایشان را بدون چشم‌داشتی نثارم کردند.

فهرست

مقدمه

برنامه‌نویسی کار لذتبخشی است، اما خوب برنامه نوشتن چیز دیگری است. چه بسیار برنامه‌هایی که از نظر بصری زیبا هستند و شاید از نظر فنی نیز فناوری‌های پیشرفته و به‌روزی در آنها استفاده شده باشد، اما به دلیل ضعف طراحی، نتوانند نیازمندی‌هایی را که از آنها انتظار می‌رود برآورده کنند. در این فصل می‌خواهیم ببینیم الگوهای طراحی چه چیزی هستند، از کجا شکل گرفته‌اند، چه مزایایی دارند و چرا باید از آنها استفاده کنیم.

طراحی نرم‌افزار

یکی از اصول طراحی نرم‌افزار، مفهومی به نام «قابلیت استفاده‌ی مجدد» یا Reusability است. این مفهوم بدین معنی است که هنگام طراحی نباید تنها به شرایط فعلی توجه کرده و تمرکزمان تنها این باشد که نیازمندی‌های فعلی را پوشش بدهیم. بلکه طراحی را باید به گونه‌ای انجام بدهیم که بتوانیم کُدهایی را که نوشته‌ایم با دردسر کمتر در جاهای دیگر (پروژه‌های دیگر) استفاده کنیم.

چگونه Reusable کار کنیم؟

اگر از طراحان با تجربه بپرسید چگونه می‌شود یک سیستم Reusable نوشت، خواهید شنید چنین کاری اگر بار نخست غیر ممکن نباشد کار بسیار سختی است. به همین دلیل طراحان با تجربه تلاش می‌کنند به جای این که طراحی را در یک دور انجام داده و نهایی کنند، در فرآیندی تکرارپذیر یا Iterative چندین بار آن را به طور عملی استفاده کرده و هر بار آن را بهبود بدهند تا در نهایت به مدل پایدار نهایی دست پیدا کنند.

بهره‌گیری از تجربه‌ی پروژه‌های گذشته می‌تواند راه خوبی باشد. اما چیزی که همه بر سر آن تفوق دارند این است که به جای اختراع چرخ، سپری کردن ماه‌ها وقت برای تحقیق و کشف و ابداع راه‌حل‌های شخصی بهتر است از راه‌حل‌های پذیرفته شده‌ی از پیش موجود استفاده کنیم.

راه حل آشنای copy-paste

برای هر یک از ما به عنوان برنامه‌نویس بارها اتفاق افتاده هنگام روبرو شدن با یک مشکل، به سرعت تکه کُد یکی از پروژه‌ی گذشته را در پروژه‌ی فعلی copy-paste کرده یا برای راه‌حل آن به اینترنت متوسل شویم. سپس با دستکاری نمونه کُدی که پیدا کرده‌ایم، مشکل را بر طرف می‌کنیم. یعنی به جای پیدا کردن دوباره‌ی راه‌حل از کُدی استفاده می‌کنیم که می‌دانیم درست کار می‌کند. سوال مهمی که در اینجا مطرح می‌شود این است که ...

آیا می‌توانیم همین کار را در خصوص طراحی انجام بدهیم؟

با وجودی که پاسخ این پرسش، منفی است و هیچ‌گاه نمی‌شود طراحی را با copy-paste در یک پروژه‌ی دیگر استفاده کرد، اما احساسی به ما می‌گوید چیزی در پروژه‌های گذشته وجود دارد که می‌توانیم آن را منتقل کنیم. نامش را تجربه می‌گذاریم، اما تجربه یک واژه‌ی بسیار گسترده، گنگ و مبهم است.

اگر بتوانیم راه‌حل‌های استفاده شده در پروژه یا پروژه‌های گذشته را از دل نیازمندی‌ها و کسب و کار پیچیده‌ی آنها، انبوه فایل‌ها، کُدها و کتابخانه‌های نوشته شده بیرون بکشیم، مسئله تقریبا حل می‌شود. اما چه کسی می‌تواند ادعا کند توانایی چنین کاری را دارد؟ چیزی که نیاز داریم ابزاری است که بدون نیاز به این کار، راه‌حل مسائلی را که در پروژه‌های آینده با آنها مواجه می‌شویم در اختیار ما قرار بدهد. به گونه‌ای که هم بتواند آنها را به روشنی بیان کند تا به سادگی قابل درک باشد و هم بتوان آنها را به راحتی در یک پروژه‌ی دیگر به کار گرفت.

یک حس غریب آزاردهنده

در شرایطی که نه می‌دانیم این ابزار چیست یا کجا و چگونه باید دنبال آن بگردیم، چاره‌ای نداریم جز این که به دانسته‌های پیشین و تجربه‌ی پروژه‌های گذشته اکتفا کنیم. اما حتی پس از این که با ترس و اضطراب، دست به طراحی زده و پروژه را پیش بردیم، آزارهای ناشی از احساس بی‌اعتمادی به طراحی دست بردار نیست. در واقع مطمئن نیستیم طراحی ما واقعا بتواند به نیازهای آینده پاسخ بدهد.

به همین دلیل پس از نوشتن برنامه اگر کسی از ما بپرسد چه احساسی دارید، آیا واقعا از نتیجه‌ی کار راضی هستید یا کار را با کثیف‌کاری و سرهم‌بندی تمام کردید، چگونه توانستید به طور هم‌زمان نیازمندی‌های برنامه را فراهم کرده و خوب کُد بنویسید، آیا توانستید همه‌ی ویژگی‌ها و خصوصیت‌های یک برنامه‌ی خوب (انعطاف‌پذیری، پشتیبان‌پذیری، اعتمادپذیری، قابلیت استفاده‌ی مجدد، توسعه پذیری و ...) را محقق کنید، آیا برنامه‌ی شما در برابر تغییرات نیازمندی‌های پروژه دوام می‌آورد، در برابر این پرسش‌ها جوابی نداریم.

تجربه، آموزگاری که نخست آزمون می‌گیرد

می‌گویند تجربه آموزگاری است که نخست آزمون می‌گیرد و سپس درس می‌دهد. آزمون‌هایش هم بسیار دردناک است، به گونه‌ای که هیچ کس تا عمر دارد تجربه‌ی آن را فراموش نمی‌کند! به همین دلیل با همه‌ی سختی‌ها، پس از اجرای چند پروژه به هر حال یک چیزهایی دستگیرمان می‌شود.

در حقیقت هرچه تجربه‌مان بیشتر می‌شود آن راه‌حل‌های گنگ و تار و آن ابزار پنهانی که در هاله‌ی ابهام قرار دارد، کم‌کم شفاف‌تر شده و این بار هنگام انتقال تجربه‌ی پروژه‌های گذشته، حدودا می‌دانیم سراغ چه چیزهایی یا چه بخش‌هایی از کُد باید برویم، چه تغییراتی باید اعمال کنیم، چه چیزهایی را باید منتقل کنیم و چه چیزهایی را نباید منتقل کنیم. همچنین احساس می‌کنیم دیدگاهی در طراحی به دست آورده‌ایم. اما باز هم به طور دقیق از توصیف آن ناتوان هستیم. به همین دلیل منتقل کردن کُدها هنوز دردسر بسیار زیادی دارد و باید کلی آنها را تمیز کنیم.

تجربه‌ی فردی کافی نیست

حتی اگر بر اساس تجربه برای خودمان راه‌حل‌هایی پیدا کنیم، مشکل بزرگ روش آزمون و خطای ما (به شرط درست بودنش) این است که تنها بر اساس تجربه‌ی خودمان به دست آمده است. تردیدی وجود ندارد که اگر به نحوی بتوانیم از تجربیات دیگران هم استفاده کنیم نتیجه‌ی بسیار بهتری خواهیم گرفت. اما چنین کاری چندان راحت نیست و وقت و انرژی زیادی صرف می‌کند. به هر حال همین تجربه‌ی فردی نیز غنیمت است. اما اگر همین تجربه را هم نداشته باشیم چه باید بکنیم؟ چه چیزی می‌تواند طراحی ما را ضمانت کرده و در این وادی تنهایی به ما کمک کند.

از آن سو اگر کار طراحان حرفه‌ای را دیده باشید خواهید دید بعد از دریافت نیازمندی‌های پروژه، به سرعت کلاس‌های مورد نیاز را طراحی می‌کنند و طراحی آنها نیز واقعا خوب است. در جایی که ما به عنوان یک تازه‌کار در نیازمندی‌های پروژه غرق شده و گهگاه تمایل داریم به روش‌های غیر شیءگرای گذشته‌ی خودمان که به آنها خو گرفته‌ایم رو بیاوریم، طراحان با تجربه چیزی را می‌دانند یا می‌بینند که برنامه‌نویسان بی‌تجربه و تازه‌کار از آن خبر ندارند.

رمز و راز کار طراحان حرفه‌ای چیست؟

واقعا طراحان حرفه‌ای از چه چیزی آگاه هستند؟ چه چیزی یک برنامه‌نویس را به رهبر تیم برنامه‌نویسی تبدیل می‌کند؟ چرا برخی برنامه‌نویسان بهتر و سریع‌تر از دیگران به این سطح رسیده و می‌توانند طراحی را بدین شکل انجام بدهند؟ با وجودی که به سادگی می‌توانیم بگوییم برخی افراد این گونه متولد شده‌اند و خوب کُدنویسی در سرشت آنها است، برای تبدیل شدن به یک طراح حرفه‌ای و موفق، رمز و راز ویژه‌ای وجود ندارد.

واقعیت این است که طراحان موفق همان ابزار کلیدی‌ای را دارند که دنبالش هستیم. آن ابزار نیز چیزی جز راه‌حل‌های درست و اثبات شده نیست. تجربه را نمی‌توان با خواندن کتاب یا مقاله به دست آورد، اما خوشبختانه آن راه‌حل‌های درست را می‌توان با مطالعه به دست آورد. این کتاب برای همین نوشته شده است تا بتوانید وارد قلمرو برنامه‌نویسان حرفه‌ای شوید.

راه‌حل انتزاعی

تمام داستان الگوهای طراحی حول چیزی به نام «تعریف انتزاعی راه‌حل» می‌چرخد. یعنی راه‌حل مسائل مختلف به صورت مستقل از پروژه و به شکلی عمومی تعریف می‌شود. از این رو با وجود تفاوت کسب و کار، نیازمندی‌ها، مشکلات و دامنه‌ی پروژه‌های مختلف، بیشتر مواقع می‌توان راه‌حل‌ها را دوباره در هر پروژه‌ای استفاده کرد. به همین دلیل در بیشتر برنامه‌های شیءگرای حرفه‌ای، مدل‌ها و الگوهای همسانی در خصوص طراحی کلاس‌ها و روابط بین آنها دیده می‌شود.

الگوهای طراحی چیستند؟

به طور خلاصه یک الگو یک راه‌حل اثبات شده‌ی انتزاعی است که برای حل مشکل ویژه‌ای ابداع شده است که بارها و بارها به شکل‌های گوناگون در پروژه‌های مختلف رخ می‌دهد. ارزشمندی الگوهای طراحی این است که دستاورد تجربیات یک یا چند نفر نیستند. بلکه از تجربیات صدها برنامه‌نویس و طراح حرفه‌ای در طول سال‌ها برنامه‌نویسی به دست آمده‌اند. همچنین چیزی نیستند که از ابتدا ابداع یا اختراع شده باشند. بلکه بازتاب طراحی‌ها و کُدنویسی‌های مجددی هستند که برنامه‌نویسان برای کسب بیشترین انعطاف‌پذیری، توسعه‌پذیری و قابلیت استفاده‌ی مجدد با آنها درگیری داشته‌اند. الگوهای طراحی نمودی موجز، خلاصه و عملی از راه‌حل‌هایی هستند که برنامه‌نویسان از تجربیات موفق خود به دست آورده‌اند.

منشاء پیدایش الگوها

با وجودی که الگوهای طراحی، دستاورد تجربیات برنامه‌نویسان بیشماری هستند، در سال ۱۹۹۴ برای نخستین بار مجموعه‌ای از مشهورترین آنها در کتابی تحت عنوان Design Patterns: Elements or Reusable Object-Oriented Software گردآوری شد که تقریبا به عنوان انجیل الگوهای طراحی شناخته می‌شود. این کتاب توسط چهار نفر به نام‌های اریک گاما، ریچارد هلم، رالف جانسون و جان ولیسیدس نوشته شد که به «چهار گانگستر» یا Gang of Four معروف هستند.

آیا الگو همان انتزاع است؟

بین الگو و انتزاع رابطه‌ی نزدیکی وجود دارد. برای نمونه درخت دودویی یک مفهوم انتزاعی است که برای حل برخی مسائل استفاده می‌شود. اما انتزاع، گسترده‌تر از الگو است و الگو را می‌توان نمونه‌ای از انتزاع دانست. برای نمونه مرتب‌سازی یک مفهوم انتزاعی است و می‌تواند برای حل مشکلات مختلفی به کار برود. این مسئله، مستقل از یک مشکل به خصوص است و از نظر پیاده‌سازی نیز چیزی را به برنامه‌نویس تحمیل نمی‌کند، بلکه تنها راه را نشان می‌دهد. به همین دلیل در یک فروشگاه الکترونیک می‌توان آن را برای مشکلی مانند «پیدا کردن ارزان‌ترین کالاها» یا «پیدا کردن آخرین سفارش‌ها» استفاده کرد. با وجودی که این دو مسئله با یکدیگر فرق دارد، اما در راه‌حل آنها به یک شکل از مفهوم انتزاعی «مرتب‌سازی» استفاده می‌شود. الگوهای طراحی نیز همین نقش را در خصوص طراحی بازی می‌کنند. یعنی راه‌حل طراحی یک مسئله را به صورت انتزاعی بیان می‌کنند.

دستاورد الگوهای طراحی

استفاده از الگوها، باعث می‌شود طراحی ساده‌تر شده و ساختار به دست آمده انعطاف‌پذیرتر، زیباتر و قابل استفاده‌ی مجددتر شود. کسی که با الگوهای طراحی آشنا باشد می‌تواند بدون نیاز به کشف دوباره‌ی راه‌حل‌ها، بی‌درنگ آنها را برای مشکلات مختلف به کار ببرد، ضمانت و اعتماد طراحی خود را افزایش بدهد و در وقت، هزینه و انرژی خود صرفه‌جویی کند. حتی الگوها توصیف و مستندسازی و نگهداری سیستم‌ها را نیز بهبود می‌دهند.

همچنین الگوها برای بیان راه‌حل مشکلات پیچیده بسیار موثر هستند و طراحی درست شیءگرا و پایبندی به اصول صحیح شیءگرایی را ترویج می‌دهند. زیرا خودشان بر اساس اصول طراحی شیءگرا ابداع شده‌اند. با در دست داشتن دانش الگوهای طراحی می‌توانید بدون درگیری با جزئیات سطح پایین پیاده‌سازی، کُدها را به سرعت درک کنید، مدل ابداع شده توسط طراح و رهبر تیم را به سرعت بفهمید، با دیگر اعضای تیم، تعامل و ارتباط مناسب‌تری داشته باشید و به شکل کاراتری به ایفای نقش خود بپردازید. در نتیجه تمام تیم بهتر برنامه‌نویسی می‌کنند، بازدهی بیشتری خواهند داشت، کیفیت کُد بالاتر خواهد رفت و نگهداری و توسعه و قابلیت استفاده‌ی مجدد به شکل بهتری فراهم خواهد شد.

آیا الگوها تنها به مهندسی نرم‌افزار مربوط می‌شوند؟

مفهوم الگو چیزی نیست که تنها در دنیای نرم‌افزار کاربرد داشته باشد بلکه در سایر زمینه‌های علمی نیز به طور مشابه –با عنوانی دیگر– استفاده شده و می‌شود. در حقیقت نخستین بار یک معمار به اسم کریستوفر الکساندر بود که در سال ۱۹۷۰ ایده‌ی الگو را برای ایجاد مجموعه‌ای از واژگان مشترک یا یک زبان در مبحث طراحی (ساختمان، عمارت، نقشه‌کشی، شهرسازی، ...) معرفی کرد.

او چنین نوشت:

> المان‌های این زبان، موجودیت‌هایی به نام الگو هستند و هر کدام آنها مشکلی را توصیف می‌کند که بارها و بارها در محیط پیرامون ما رخ می‌دهد. هر الگو مغز و قلب راه‌حل را به گونه‌ای توضیح می‌دهد که بتوان آن را میلیون‌ها بار به کار برد، بدون آن که حتی دو نمونه از آنها مانند هم باشد.

جملات آلکساندر به همان اندازه‌ای که برای طراحی، نقشه‌کشی و شهرسازی قابل بهره‌برداری است برای دنیای نرم‌افزار نیز قابل استفاده است. از این رو با وجودی که الگوهای هر زمینه‌ی علمی به همان حیطه‌ی فنی تعلق دارد، اما از نظر مفهومی، روش الگوسازی در همه‌ی زمینه‌ها قابل استفاده است. همچنین برخی الگوها از زمینه یا context خود مستقل هستند و می‌توان آنها را از یک context به یک context دیگر منتقل و استفاده کرد.

سطح کاری الگوها

پس از درک الگوهای طراحی، دیگر دیدگاه برنامه‌نویسی شما مانند گذشته نخواهد بود. بصیرت و بینشی به دست می‌آورید که باعث می‌شود طراحی‌های شما انعطاف‌پذیرتر، قابل استفاده‌ی مجددتر و قابل فهم‌تر شود. از آنجایی که سطح الگوها بالاتر از مفاهیم شیءگرا است، پس از مطالعه‌ی الگوهای طراحی و هنگام برخورد با مشکلات، دیگر به این فکر نمی‌کنید برای حل مسئله باید از کدام قابلیت زبان مانند وراثت، کلاس مجرد، کلاس استاتیک و ... استفاده کنید. بلکه می‌گویید از چه الگویی باید استفاده کنم. تصمیم‌گیری و انتخاب حذف نمی‌شود، بلکه سطح آن به طور چشمگیری بالاتر می‌رود.

خلاصه‌ی مزایای الگوهای طراحی

به طور خلاصه مزایای استفاده از الگوها را می‌توان چنین ذکر کرد:

- طراحی بهتر
- کدنویسی زیباتر
- صرفه‌جویی در زمان و هزینه
- افزایش قابلیت استفاده‌ی مجدد
- افزایش اطمینان و ضمانت طراحی
- کاهش هزینه‌ی نگهداری و توسعه
- افزایش شیءگرایی
- تعامل بهتر با اعضای تیم
- درک ساده‌تر کتابخانه‌ها، مدل‌ها و چهارچوب‌های کاری دیگر
- به دست آوردن یک زبان مشترک و قابل فهم برای طراحی

آیا استفاده از الگوهای طراحی همیشه الزامی است؟

همانند افرادی که هیچگاه از الگوها استفاده نمی‌کنند، برخی افراد هم هستند که تلاش می‌کنند همیشه به هر بهایی از الگو استفاده کنند. اما چنین کاری خلاف مقصود الگوها یعنی ساده‌سازی است. راه درست استفاده از الگوها این است که نخست مشکل خود را به خوبی بشناسید، سپس ببینید آیا راه‌حل از پیش آماده‌ای برای آن وجود دارد یا خیر.

به یاد داشته باشید همیشه و همه جا مجبور نیستید از الگوهای طراحی استفاده کنید. اگر با مشکل ساده‌ای مواجه شدید که در هیچ یک از الگوها قرار نمی‌گرفت خودتان را مجبور نکنید به هر نحوی شده یک الگو به آن بخورانید یا آن را آنقدر انتزاعی کنید که به شکل نمونه‌ای از یک الگو در بیاید. با این کارها کُد خودتان را پیچیده می‌کنید. در واقع همان گونه که الگوها می‌توانند مشکلات پیچیده را ساده کنند می‌توانند مشکلات ساده را هم پیچیده کنند.

همچنین فراموش نکنید که میزان کاربرد الگوها با یکدیگر متفاوت است. قرار نیست در یک پروژه همه‌ی الگوها را به کار ببرید. استفاده از الگوها به صورت مسئله و مشکلات پروژه بستگی دارد. اگر پس از اتمام این کتاب، چندین پروژه را پشت سر گذاشتید اما از برخی الگوها استفاده نکردید در درستی آنها شک نکنید. هر الگویی کاربرد ویژه‌ی خودش را دارد.

به ویژه توجه داشته باشید گاهی برای حل یک مسئله ممکن است راه‌حل‌های مختلفی وجود داشته باشد. از این رو تصور نکنید هنگام روبرو شدن با فلان مشکل الزاما باید از فلان الگوی طراحی استفاده کنید. انتخاب یک الگوی طراحی مناسب به فاکتورها و عوامل مختلفی بستگی دارد. برای انتخاب مناسب نیز در درجه‌ی نخست باید صورت مسئله و نیازمندی‌های آن را به روشنی بشناسید و بر شرایط پروژه و خواسته‌های آن آگاهی و اشراف کافی داشته باشید.

الگوهای طراحی در چه سطحی یا چه نوعی از برنامه‌ها استفاده می‌شوند؟

الگوها به لایه یا سطح ویژه‌ای تعلق ندارند. برخی الگوها مانند Facade، Adapter، Prototype یا Decorator به تنهایی قابل استفاده هستند. اما برخی الگوها مانند Factory Method، Abstract Factory، Singleton و مانند آن عمدتا در ترکیب با یک یا چند الگوی دیگر استفاده می‌شوند. همچنین الگوها مستقل از نوع برنامه هستند و فرقی نمی‌کند برنامه‌ی شما یک برنامه‌ی رومیزی است، یک برنامه‌ی تحت وب است یا یک برنامه‌ی کنسول یا وب‌سرویس.

الگوهای طراحی به چه زبان یا سکویی نیاز دارند

در حالت کلی الگوهای طراحی مستقل از زبان برنامه‌نویسی و سکوی نرم‌افزار هستند. تنها چیزی که الگوهای طراحی به آن نیاز دارند شیءگرایی است. به همین دلیل الگوهای طراحی را با همه‌ی زبان‌های شیءگرای استاندارد می‌توان پیاده‌سازی کرد و دانشی را که در خصوص آنها به دست می‌آورید به سادگی می‌توانید به یک زبان دیگر، یک سکوی دیگر و یک چهارچوب کاری دیگر منتقل کنید.

با این وجود قابلیت‌های خاص زبان‌های برنامه‌نویسی و سکوهای نرم‌افزاری به طور چشمگیری پیاده‌سازی الگوها را تحت تاثیر قرار می‌دهد. برای نمونه پیاده‌سازی الگوها در یک زبان داینامیک یا بدون نوع (typeless) با یک زبان استاتیک یا نوع‌دار (strongly typed) فرق دارد. در واقع پیاده‌سازی یک الگو زمانی کمال پیدا می‌کند که با قابلیت‌های سکوی برنامه‌نویسی به خوبی آشنا باشید. به همین دلیل در آغاز این کتاب نخست قابلیت‌های زبان #C را مرور خواهیم کرد. همچنین هنگام توضیح الگوها تا جای ممکن تلاش کرده‌ایم قابلیت‌هایی از زبان #C را که به پیاده‌سازی الگو کمک می‌کند بیان کنیم.

الگوهای طراحی برای چه کسانی هستند؟

اگر به کُدهای خود عشق می‌ورزید، کسی هستید که هر خط کُد برایش ارزش دارد، کسی که به کیفیت کُد اهمیت می‌دهد و دنبال سرهم‌بندی کردن نیست، این کتاب برای شما است. این کتاب به شما کمک می‌کند کُدهایی صحیح‌تر، زیباتر، توسعه‌پذیرتر و مفیدتر بنویسید. اگر دنبال چیزی می‌گردید که سطح کاری کُدهای شما را بالاتر ببرد این کتاب نیاز شما را بر آورده می‌کند.

الگوها چیز شگفت‌انگیزی نیستند. در واقع برای حل مشکل برخی الگوها اگر کمی منطقی فکر کنید ممکن است خودتان هم به همان راه‌حل یا راه‌حل مشابهی برسید. به همین دلیل ممکن است قبلا الگوها یا شکل مشابهی از آنها را، دانسته یا نادانسته، استفاده کرده باشید. در این حالت الگوی مطرح شده باعث تایید دانسته‌های شما می‌شود و به شما می‌گوید راه را اشتباه نرفته‌اید.

الگوها چه چیزی را برآورده نمی‌کنند

الگوهای طراحی به خودی خود برای شما معجزه نمی‌کنند. این شما هستید که باید مشکلات برنامه‌ی خود را به خوبی بشناسید، بتوانید به درستی آنها را تعمیم داده (انتزاعی کنید) و سپس الگوی مناسبی برای حل آنها پیدا کنید. از این رو پس از خواندن کتاب انتظار نداشته باشید در پروژه‌ی خود بتوانید یک مدل خارق العاده ابداع کنید.

استفاده‌ی درست از الگوها به شناخت درست صورت مسئله و مهمتر از آن به تجربه بستگی دارد. هرچقدر تجربه‌ی شما بیشتر باشد، مهارت شما در استفاده از الگوها بالاتر خواهد رفت. پس از خواندن کتاب سعی کنید الگوها و کاربرد آنها را به خاطر بسپارید و هنگام مواجه شدن با مسائل مختلف ببینید کدام را می‌توانید استفاده کنید.

آیا الگوهای دیگری نیز وجود دارد؟

همان گونه که گفتیم الگوهای طراحی نخستین بار اواخر سال ۱۹۹۴ در کتاب Design Patterns: Elements of Reusable Object-Oriented Software معرفی شد. آن الگوها که به اختصار به آنها الگوهای GoF می‌گوییم امروزه به عنوان رایج‌ترین و اصلی‌ترین الگوهای طراحی شناخته می‌شوند. الگوهایGoF نخست به زبان C++ و SmallTalk (زبان‌های شیءگرای آن زمان) پیاده‌سازی شده بودند و سپس به دیگر زبان‌ها مانند جاوا، VB و C# نیز نوشته شدند.

پس از پذیرش مفهوم الگو، الگوهای دیگری نیز توسط دیگران معرفی شد. اگرچه برخی از الگوهای جدید هم جالب هستند اما به اندازه‌ی الگوهای GoF شهرت ندارند. همچنین افزون بر *طراحی*، در سایر زمینه‌ها مانند معماری نرم‌افزار، واسط کاربر، همروندی، امنیت، پایداری و غیره نیز الگوهای دیگری توسط دیگر برنامه‌نویسان معرفی شد.

چیزی که قصد دارم در این قسمت تاکید کنم این است که در این کتاب تنها به *الگوهای GoF* می‌پردازیم و با الگوهای ابداع شده‌ی دیگران و همچنین الگوهای سایر زمینه‌ها کاری نداریم.

تقسیم‌بندی الگوهای طراحی

مولفین GoF در کتاب خود ۲۳ الگوی معرفی شده را به سه دسته‌ی کلی تقسیم کردند:

- الگوهای ساختاری (Structural Patterns)
  موضوع این الگوها روابط بین اشیاء و مشارکت آنها برای تشکیل اشیاء پیچیده‌تر است.
- الگوهای ایجادی (Creational Patterns)
  این الگوها به ساخت اشیاء و ارجاع به آنها مربوط می‌شوند.
- الگوهای رفتاری (Behavioural Patterns)
  این الگوها به ارتباط بین اشیاء به ویژه از دیدگاه مسئولیت و الگوریتم مربوط می‌شوند.

چرا تالیف

با وجود کتاب بسیار اثرگذاری مانند Design Patterns: Elements of Object-Oriented Software که مرجع واقعی الگوهای طراحی است ممکن است بپرسید چرا نویسنده به جای ترجمه اقدام به تالیف نموده است. دلایل مختلفی برای این کار وجود دارد و مهمترین دلیلش متفاوت بودن زبان برنامه‌نویسی کتاب GoF با C# و بالا بودن سطح آن کتاب از نظر فنی است. اگر اینترنت را بررسی کنید خواهید دید با وجود کتاب‌های بیشماری که در زمینه‌ی الگوهای طراحی برای سایر زبان‌ها به ویژه جاوا نگاشته شده است، در زمینه‌ی C# خلاء روشنی به چشم می‌خورد.

شاید کتاب جیمز کوپر در سال ۲۰۰۲ یا کتاب جودیت بیشاپ در سال ۲۰۰۸ را بتوان یکی از معدود کتاب‌های الگوهای طراحی زبان C# دانست. مساله‌ی دیگری که وجود دارد این است که نویسنده در مطالعه‌ی کتاب‌ها و مستندات مختلف الگوهای طراحی GoF، نوعا مشاهده نموده افراد تنها به بیان الگوها می‌پردازند و چندان به پیامدهای آن توجه نمی‌کنند. به همین دلیل نویسنده قصد داشته کتابی تهیه نماید که برنامه‌نویسان C# را به شکل عملی‌تری به ویژه بر اساس جدیدترین قابلیت‌های زبان C# با الگوهای طراحی آشنا نماید و همچنین دیدگاهی تحلیلی بر الگوهای طراحی داشته باشد. دیدگاهی که کمتر به آن پرداخته شده است. در عین حال نویسنده به خواننده بسیار توصیه می‌نماید کتاب اصلی الگوهای طراحی نوشته‌ی گروه GoF را به جهت ارزشمندی فوق‌العاده‌ی آن مطالعه نماید.

مخاطبان کتاب

مخاطبان این کتاب به طور ویژه طراحان نرم‌افزار هستند. با این حال این کتاب برای تمام برنامه‌نویسان شیءگرا که دوست دارند خوب کُد بنویسند بسیار مفید است. همچنین این کتاب برای دانشجویان رشته‌ی مهندسی نرم‌افزار نیز مناسب است.

چه چیزی در این کتاب آموزش داده می‌شود؟

در این کتاب تمام ۲۳ الگوی GoF را توضیح می‌دهیم. اما افزون بر آن ...

- با قابلیت‌های زبان #C آشنا می‌شوید
- با نمودارهای کلاس UML آشنا می‌شوید
- با تعدادی از اصول و تکنیک‌های طراحی آشنا می‌شوید
- و چند مثال و برنامه‌ی عملی خواهید دید که از الگوها برای پیاده‌سازی آنها استفاده شده است

برخی از افراد عقیده دارند برخی از الگوها بیشتر جنبه‌ی نظری یا تجملاتی دارند. در این کتاب می‌خواهیم ببینیم آیا واقعا چنین است یا خیر. رویه‌ی ما در این کتاب تا حدی تحلیلی است. از این رو هنگام بیان هر الگو، ابعاد مختلف آن را شکافته و پیامدهای آن را بررسی می‌کنیم. این مسئله کمک می‌کند درک عمیق‌تری نسبت به الگو پیدا کنید و بهتر بتوانید تصمیم‌گیری کنید آیا در برخورد با یک مشکل به خصوص، الگویی که برای آن ارائه شده واقعا به درد شما می‌خورد یا خیر.

چه چیزی در این کتاب آموزش داده نمی‌شود؟

به منظور جلوگیری از هرگونه اشتباه برداشت نسبت به این کتاب، باید توضیح بدهیم این کتاب چه چیزهایی را نیز آموزش نمی‌دهد. با وجودی که این کتاب به الگوهای طراحی می‌پردازد اما باید بدانید این کتاب هیچ یک از موارد زیر نیست:

- آموزش NET. یا زبان #C
- آموزش برنامه‌نویسی شیءگرا (OOP)
- تکنیک‌ها و ترفندهای برنامه‌نویسی
- طراحی دیتابیس
- آموزش تحلیل و طراحی
- آموزش UML یا RUP
- برنامه‌نویسی چابک (Agile)
- مدیریت پروژه یا مهندسی نرم‌افزار

این کتاب چیزی درباره‌ی چگونگی تحلیل، مدل‌سازی، شناخت منطق کسب و کار پروژه، تشخیص موجودیت‌ها و ارتباطات آنها ارائه نمی‌دهد. این کتاب تنها مجموعه‌ای از الگوهای انتزاعی بسیار رایج است که هر کدام به طور دقیق به یک مسئله‌ی ویژه می‌پردازد.

برای استفاده از این کتاب به چه چیزی نیاز دارید؟

برای استفاده از کتاب فرض می‌شود به طور نسبی با زبان #C و اصول برنامه‌نویسی شیءگرا آشنا هستید. بیشتر مثال‌های کتاب نیز برنامه‌های ساده‌ی کنسول یا خط فرمان هستند که به سادگی می‌توانید آنها را در Visual Studio اجرا کنید. از نظر نسخه‌ی Visual Studio نیز تقریبا بیشتر مثال‌ها را می‌توانید در Visual Studio 2008 اجرا کنید. ولی بعضی از مثال‌ها را که با قابلیتی از زبان #C در4.0 NET. یا 4.5 NET. نوشته‌ایم باید در Visual Studio 2010 یا نسخه‌های بعدی آن اجرا کنید.

در فصل نخست کتاب، مفاهیم برنامه‌نویسی شیءگرا و خلاصه‌ای از مهمترین قابلیت‌های زبان #C را مرور کرده‌ایم. در صورتی که با برنامه‌نویسی شیءگرا و زبان #C آشنا هستید نیازی به خواندن فصل ۱ ندارید، اما مرور سریع آن خالی از لطف نیست. زیرا احتمالا نکات جالبی در آن پیدا خواهید کرد. با این وجود خواندن یکباره‌ی فصل ۱ تا حدی خسته‌کننده است و این فصل را می‌توان مرجع قابلیت‌های زبان #C محسوب کرد.

افزون بر این، فصلی را هم به مرور UML اختصاص داده‌ایم تا آن دسته از خوانندگانی که با نمودارهای UML آشنایی نداشته یا آشنایی کمی دارند، هنگام روبرو شدن با الگوها مشکل پیدا نکنند. در صورتی که با UML آشنا هستید می‌توانید از فصل نیز ۲ عبور کنید. اما توصیه می‌کنیم این فصل را هم به صورت سریع مرور کنید.

سپس فصلی را به اصول طراحی اختصاص داده‌ایم. البته این فصل خودش موضوع یک کتاب جداگانه است و یادگیری اصول طراحی در چنین فصل مختصری میسر نیست. اما از آنجایی که در توضیح و نقد و بررسی الگوهای طراحی کرارا به این اصول ارجاع کرده‌ایم چاره‌ای نداشتیم که دست‌کم این اصول را به صورت کلی مرور کنیم تا خواننده هنگام مطالعه‌ی الگوهای طراحی و روبرو شدن با اصطلاحاتی مانند تزریق وابستگی، اصل Soc، قانون جایگزینی لیسکف و همانند آن مشکل پیدا نکند.

شیوه‌ی این کتاب برای آموزش الگوها چیست؟

مطالبی که در مورد هر الگو بیان کرده‌ایم از چند قسمت تشکیل می‌شود:

- **هدف:** به طور بسیار خلاصه بیان می‌کند هدف الگو چیست و چه چیزی را برآورده می‌کند.
- **انگیزه:** در این قسمت مقدمه‌ای بیان می‌کنیم تا ذهن خواننده برای بیان راه‌حل آماده شود. در واقع سعی می‌کنیم نخست مشکل را به صورت عملی توضیح بدهیم و بگوییم انگیزه‌ی ابداع الگوی مورد بحث چیست و الگوی مورد ارائه، چه چیزی را باید فراهم کند.
- **توضیح:** در این بخش به طور دقیق الگو را توضیح می‌دهیم.
- **ساختار:** در این قسمت نمای بصری راه‌حل الگو را به شکل نمودار کلاس UML نشان می‌دهیم.
- **شرکت کنندگان:** در این قسمت اجزای نمودار قسمت «ساختار» را توضیح می‌دهیم.
- **نمونه‌ی عملی در NET.:** در صورتی که یک الگو به طور عملی در چهارچوب کاری NET. استفاده شده باشد، مورد کاربرد آن را در این قسمت بیان می‌کنیم.
- **پیاده‌سازی:** در این قسمت پیاده‌سازی الگو را به زبان #C بیان می‌کنیم. تلاش کرده‌ایم تا جای ممکن به صورت انتزاعی عمل کنیم و کُد پیاده‌سازی شده، با نمودار UML قسمت «ساختار» هماهنگی داشته باشد تا خواننده بهتر الگو را یاد بگیرد.
- **نکته‌ها و پیامدها:** در این قسمت تلاش کرده‌ایم تا جای ممکن هر نکته‌ای را که در رابطه با الگوی مورد بحث وجود داشته باشد بیان کرده و پیامدهای آن را از جهات مختلف بررسی کنیم.
- **نقاط قوت:** در این قسمت نقاط قوت و مزایای الگو را به طور خلاصه ذکر می‌کنیم.
- **نقاط ضعف:** در این قسمت نقاط ضعف الگو را به طور خلاصه ذکر می‌کنیم.
- **کاربرد:** در این قسمت به طور خلاصه بیان می‌کنیم بهتر است چه زمانی از الگوی ارائه شده استفاده کنید.
- **الگوهای مرتبط:** و در این قسمت الگوهای مرتبط با الگوی توضیح داده شده را ذکر می‌کنیم.

دلگرمی پایانی

همان گونه که گفتیم این کتاب یک کتاب آموزش برنامه‌نویسی نیست. بلکه یک کتاب طراحی است و مخاطبان ویژه‌ای دارد. از این رو ممکن است برخی از مطالب آن برای شما گنگ و پیچیده به نظر برسد. با این وجود این کتاب یک رساله‌ی دکترای فنی و سطح بالا هم نیست. الگوهای طراحی توسط هر کسی که با شیءگرایی آشنا باشد قابل درک است.

اگر در نخستین بار همه‌ی مطالب کتاب را به طور کامل نفهمیدید نگران نشوید. همچنین فراموش نکنید این کتابی نیست که یک بار بخوانید و سپس برای همیشه در قفسه‌ی کتابخانه بگذارید. بلکه مرجعی است که ممکن است بارها سراغش بیایید. برای افزایش بهره‌وری از کتاب سعی کنید مفهوم هر الگو را درک کنید و الگوها را بر اساس مثال و پیاده‌سازی به خاطر بسپارید. در پایان کتاب جدولی ذکر کرده‌ایم و لیست الگوهای طراحی را به طور خلاصه در آن آورده‌ایم تا هر زمان بتوانید با مراجعه به این جدول، آموخته‌های خود را مرور کنید.

کُدهای پیوست

می‌توانید کُدهای پیوست کتاب را از پایگاه وب انتشارات پندارپارس به آدرس www.pendarepars.com دانلود نمایید. برای این کار ابتدا در پایگاه وب انتشارات، صفحه‌ی مشخصات کتاب را جستجو کنید. در آن صفحه می‌توانید پایین تصویر و مشخصات کتاب از برگه‌ی «سورس کُد و ضمائم»، لینک دانلود فایل فشرده‌ی کُدهای پیوست را پیدا کنید.

تماس با نویسنده

در صورتی که انتقاد یا نظری نسبت به کتاب داشتید یا ایراد، کاستی، نقص یا حتی خطای فنی در کتاب مشاهده کردید در آگاه کردن نویسنده از نظر خود دریغ نورزید. آدرس تماس نویسنده mansoor.omrani@gmail.com می‌باشد. در پایان وی امیدوار است این کتاب برای برنامه‌نویسان #C مفید بوده و مورد پسند واقع شود.

بخش اول. مفاهیم اولیه

فصل ۱. مروری بر شیءگرایی و زبان C#

مروری بر شیءگرایی

پایه و اساس چهارچوب کاری .NET. و زبان C# را شیءگرایی تشکیل می‌دهد. برنامه‌نویسی شیءگرا[1] سبکی از برنامه‌نویسی است که در آن برنامه بر اساس نمونه‌هایی از مفاهیم و موجودیت‌ها ساخته می‌شود. این نمونه‌ها دو ویژگی دارند:

- می‌توانند حاوی اطلاعات باشند
- می‌توانند کار انجام بدهند

از نظر فنی هر نمونه می‌تواند از دو چیز تشکیل شود:

- فیلد و خصوصیت
- توابع و رویه‌های اجرایی

به آن موجودیت‌ها « کلاس » و به این نمونه‌ها « شیء » گفته می‌شود. گاهی به برنامه‌نویسی شیءگرا برنامه‌نویسی مبتنی بر کلاس[2] نیز گفته می‌شود.

برنامه‌نویسی شیءگرا بر اساس سه قابلیت مهم استوار است:

- کپسوله‌سازی[3]
کپسوله‌سازی مکانیزمی است که طی آن می‌توان اجزای تشکیل دهنده‌ی شیء را درون آن بسته‌بندی و همچنین دسترسی به آنها را محافظت کرد تا هر کسی نتواند به آنها دسترسی داشته باشد.
- وراثت[4]
مکانیزمی است که طی آن می‌توان بین کلاس‌ها، رابطه‌ی « هست–یک » یا Is-A ایجاد کرد. برای نمونه « گربه یک حیوان » است در برنامه‌نویسی شیءگرا یک جور رابطه‌ی وراثت است. در رابطه‌ی وراثت، یک کلاس می‌تواند خصلت‌ها و رفتار کلاس دیگری به نام کلاس پدر[5]، سوپر کلاس[6] یا کلاس پایه[7] را ارث ببرد. به کلاس وارث نیز کلاس فرزند[8]، کلاس مشتق شده[9] یا زیرکلاس[10] گفته می‌شود.
- چندریختی[11]
قابلیتی است که طی آن می‌توان اشیائی از نوع‌های مختلف ایجاد کرد که با وجود مشابه بودن واسطشان، رفتار متفاوتی داشته باشند. از آنجایی که واسط اشیاء ایجاد شده مانند یکدیگر است می‌توان آنها را بر اساس واسطشان بدون آگاهی از نوع واقعی‌شان استفاده کرد.

---

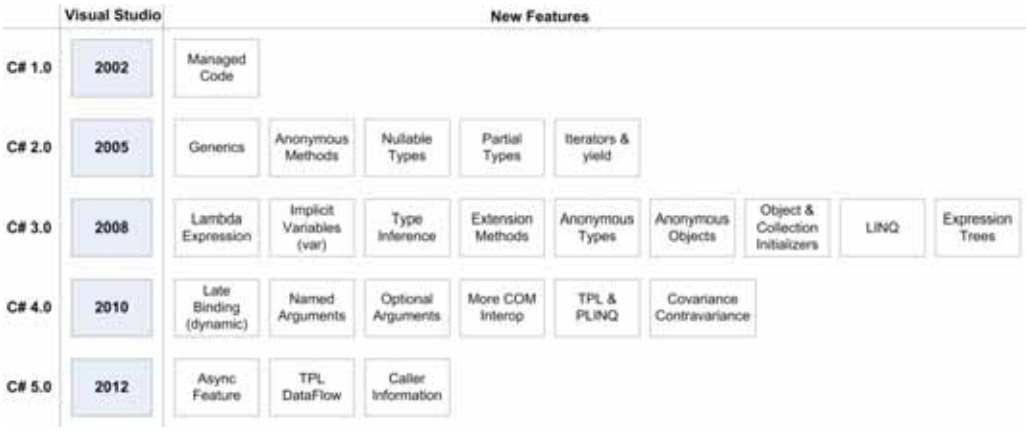
[1] Object-Oriented Programming
[2] Class-Based Programming
[3] Encapsulation
[4] Inheritance
[5] parent class
[6] super class
[7] base class
[8] child class
[9] derived class
[10] sub-class
[11] polymorphism

به کپسوله‌سازی گاهی *پنهان‌سازی اطلاعات*[12] نیز گفته می‌شود اما همه بر این مسئله توافق ندارند. به گونه‌ای که information hiding یک اصل و کپسوله‌سازی یکی از تکنیک‌های برآورده کردن این اصل بیان می‌شود. زیرا پنهان‌سازی اطلاعات در حالت کلی به جداسازی و پنهان کردن بخش‌های مستعد تغییر، بر پایه‌ی یک واسط عمومی و ثابت گفته می‌شود. بدین ترتیب با پنهان کردن آنها تحت پوشش واسط در برگیرنده می‌توان از قسمت‌های دیگر برنامه در برابر تغییرات آنها محافظت کرد.

با این وجود این تعریف نیز چندان درست نیست. زیرا برای حفاظت قسمتی از برنامه در برابر تغییرات یک قسمت دیگر، می‌توان از تکنیکی مانند تزریق وابستگی استفاده کرد، بدون آن که نیازی باشد چیزی پنهان شود. در حالی که کپسوله‌سازی به دسترسی نیز توجه دارد و سعی می‌کند حفاظت اطلاعات بسته‌بندی شده را نیز فراهم کند.

بررسی قابلیت‌های زبان #C

به طور خلاصه می‌توان سیر تحول زبان #C را در شکل زیر نشان داد.



شکل ۱-۱ سیر تحول زبان #C و قابلیت‌های آن

C# 1.0

نوع یا Type

چیزی است که ماهیت داده‌ها یا اطلاعات مختلف برنامه را تعریف می‌کند و از جنس آن می‌توان متغیر تعریف کرد.

متغیر

از دیدگاه سورس کُد، نامی است که برای یک داده یا فقره اطلاع مشخص می‌شود. زبان‌های NET. زبان‌هایی نوع‌دار یا strongly typed هستند. در چنین زبان‌هایی بدون مشخص کردن نوع یک متغیر نمی‌توان آن را تعریف کرد.

```
int myIntVariable;
double myDoubleVariable;
string myStringVariable;
```

وقتی متغیری از جنس یک نوع (Type) تعریف می‌شود نوع آن تا ابد غیر قابل تغییر خواهد بود. برای نمونه متغیری که از نوع string تعریف شده تا ابد از جنس string باقی می‌ماند. البته در C# 4.0 قابلیت نوع داده‌ی پویا نیز به #C افزوده شد. توسط این قابلیت می‌توان متغیری تعریف کرد که نوع آن قابل تغییر باشد. با این حال این قابلیت حالت محدودی از پویایی به معنی واقعی آن است و زبان‌های NET. کماکان strongly typed محسوب می‌شوند.

[12] Information Hiding