Yagni

"شما به آن نیاز نخواهید داشت" یک اصل است که از برنامه نویسی (XP) ناشی می شود که بیان می کند یک برنامه نویس نباید تا زمانی که ضروری تشخیص داده شود، عملکردی اضافه کند. اشکال دیگر عبارت عبارتند از: "شما به آن نیاز نخواهید داشت" و "شما به آن نیاز ندارید"

ران جفریس، یکی از بنیانگذارانXP ، این فلسفه را توضیح داد: "همیشه چیزها را زمانی که واقعا به آنها نیاز دارید اجرا کنید، هرگز زمانی که فقط پیش بینی می کنید [به آنها] نیاز خواهید داشت." توسعه دهندگان باتجربه قدردانی کنند که معماری برای نیازها / برنامه های آینده چقدر به ندرت مثبت می شود".

YAGNIیک اصل در پشت تمرین XP است که "ساده ترین کاری را انجام دهید که ممکن است کار کند " (DTSTTCPW) قرار است از آن در ترکیب با چندین روش دیگر مانند بازسازی پیوسته، آزمایش واحد خودکار پیوسته و یکپارچه سازی مداوم استفاده شود. اگر بدون بازسازي مداوم استفاده شود، مي‌تواند منجر به کد نامرتب و دوباره کاري گسترده شود که به عنوان بدهي فني شناخته مي‌شود.

Kiss

K : Keep I : it

S : super S: simple

KISS مخفف است به معنای ساده نگه دارید. من دوست ندارم از کلمه احمق استفاده کنم زیرا می دانم خوانندگان من احمق نیستند، به همین دلیل است که آن را به Keep It Super Simple ترجمه می کنم.

KISS یک اصطلاح مهم در برنامه نویسی است زیرا در ضمیر ناخودآگاه شما قرار می دهد که هر فرآیندی که ایجاد می کنید باید تا حد امکان ساده و همچنین به همان اندازه کارآمد باشد.

وقتی می گوییم «ساده»، منظورمان چیست؟

ساده در این زمینه لزوماً به معنای آسان نیست، بلکه صرفاً به معنای تولید همان نتایج یا نتیجه بهتر با تلاش یا پیچیدگی کمتر است.

در مفهوم KISS، ما نه فقط کمتر می خواهیم و نه بیشتر، ما فقط می خواهیم فقط به اندازه ای که لازم است داشته باشیم

اگر در حال ساخت محصولی هستید که یک شی را از نقطه A به نقطه B منتقل می کند، این کار را تا حد امکان کارآمد انجام دهید و در عین حال سادگی در پشت ذهن خود دارید.

DRY (Don’t Repeat Yourself) Principle in Java with Examples

DRY صرفاً یک رویکرد یا می توان گفت دیدگاهی متفاوت برای برنامه نویسان است. DRY مخفف عبارت Don’t Repeat Yourself است. در جاوا، به این معنی است که یک کد را به طور مکرر ننویسید. فرض کنید در بسیاری از مکان‌های برنامه خود کد مشابهی دارید، پس به این معنی است که از رویکرد DRY پیروی نمی‌کنید. شما همان کد را در مکان های مختلف تکرار می کنید. از این رو، راه حل با استفاده از مفهوم DRY با قرار دادن روش ها به جای همه کدهای تکراری و تعریف کد در یک روش به دست می آید. پس با فراخوانی متدها به اصل DRY می رسیم. مفهوم DRY برای بهتر کردن کد با کاهش افزونگی کد و تشویق قابلیت استفاده مجدد آن بسیار مهم است.

Solid

اصول SOLID پنج اصل طراحی کلاس شی گرا هستند. آنها مجموعه ای از قوانین و بهترین شیوه ها هستند که باید هنگام طراحی ساختار کلاس رعایت شوند.

این پنج اصل به ما کمک می کند تا نیاز به الگوهای طراحی خاص و به طور کلی معماری نرم افزار را درک کنیم. بنابراین من معتقدم که این موضوعی است که هر توسعه دهنده باید یاد بگیرد.

این مقاله هر آنچه را که برای اعمال اصول SOLID در پروژه های خود نیاز دارید به شما آموزش می دهد.

ما با نگاهی به تاریخچه این اصطلاح شروع خواهیم کرد. سپس با ایجاد یک طرح کلاس و بهبود گام به گام به جزئیات دقیق – چرایی و چگونگی هر اصل – می‌پردازیم.

پس یک فنجان قهوه یا چای بردارید و بیایید همان جا بپریم!

زمینه

اصول SOLID برای اولین بار توسط دانشمند معروف کامپیوتر رابرت جی مارتین (با نام مستعار عمو باب) در مقاله خود در سال 2000 معرفی شد. اما مخفف SOLID بعدها توسط مایکل فیرز معرفی شد.

عمو باب همچنین نویسنده کتاب های پرفروش Clean Code و Clean Architecture است و یکی از شرکت کنندگان " "Agile Allianceاست.

بنابراین، جای تعجب نیست که همه این مفاهیم کدگذاری تمیز، معماری شی گرا و الگوهای طراحی به نحوی با یکدیگر مرتبط و مکمل باشند.

همه آنها به یک هدف عمل می کنند:

"برای ایجاد کد قابل فهم، خوانا و قابل آزمایش که بسیاری از توسعه دهندگان بتوانند به طور مشترک روی آن کار کنند."بیایید یک به یک به هر اصل نگاه کنیم. پس از مخفف SOLID، آنها عبارتند از:

اصل مسئولیت واحد

اصل باز-بسته

اصل جایگزینی لیسکوف

اصل جداسازی رابط

اصل وارونگی وابستگی

اصل مسئولیت واحد

اصل مسئولیت واحد بیان می کند که یک کلاس باید یک کار را انجام دهد و بنابراین باید تنها یک دلیل برای تغییر داشته باشد.

برای بیان این اصل به صورت فنی تر: فقط یک تغییر بالقوه (منطق پایگاه داده، منطق ورود به سیستم و غیره) در مشخصات نرم افزار باید بتواند بر مشخصات کلاس تأثیر بگذارد.

این به این معنی است که اگر یک کلاس یک محفظه داده است، مانند یک کلاس Book یا یک کلاس Student، و دارای فیلدهایی در رابطه با آن موجودیت است، تنها زمانی باید تغییر کند که مدل داده را تغییر دهیم.

پیروی از اصل مسئولیت واحد مهم است. اول از همه، از آنجا که تیم های مختلف می توانند بر روی یک پروژه کار کنند و به دلایل مختلف یک کلاس را ویرایش کنند، این می تواند منجر به ایجاد ماژول های ناسازگار شود.

دوم، کنترل نسخه را آسان تر می کند. به عنوان مثال، فرض کنید یک کلاس persistence داریم که عملیات پایگاه داده را مدیریت می کند، و ما شاهد تغییر در آن فایل در commit های GitHub هستیم. با پیروی از SRP، متوجه می شویم که مربوط به ذخیره سازی یا موارد مرتبط با پایگاه داده است.

تضادهای ادغام نمونه دیگری است. زمانی که تیم های مختلف یک فایل را تغییر می دهند ظاهر می شوند. اما اگر از SRP پیروی شود، تضادهای کمتری ظاهر می‌شوند - فایل‌ها یک دلیل واحد برای تغییر خواهند داشت، و تضادهایی که وجود دارند راحت‌تر حل می‌شوند.

دام های رایج و ضد الگوها

در این بخش به برخی از اشتباهات رایج که اصل مسئولیت واحد را نقض می‌کنند، نگاه می‌کنیم. سپس در مورد روش هایی برای رفع آنها صحبت خواهیم کرد.

ما به عنوان مثال به کد یک برنامه ساده فاکتور کتابفروشی نگاه خواهیم کرد. بیایید با تعریف یک کلاس کتاب برای استفاده در فاکتور شروع کنیم.