UML، چرا و چگونه؟

محمدمهدی برقی و سجاد سلطانیان

کمتر هنرمندی را می‌توان یافت که قبل از رسم یک اثر هنری، نمونه اولیه آن را در ذهن و یا روی کاغذ نیاورده باشد. همچنان که هیچ خیاطی نیست که بدون الگو و پیاده سازی طرح روی کاغذ دست به کار شود و تقریبا هیچ معماری وجود ندارد که بدون نقشه کامل و اصولی خشت اول را بنهد. چون قطعا تا ثریا کج می‌رود!

در این نوشته سعی شده است تا به صورت تجربه‌محور و علمی به بیان چرایی داشتن یک UML خوب بپردازیم و در ادامه، راهکارهایی برای رسیدن به یک نمونه خوب و اصولی از آن معرفی کنیم. به امید اینکه خشت اول را اصولی تر از همیشه بگذارید و یک پروژه خوب را تجربه کنید.

پروژه در یک نگاه

نه تنها در طول این پروژه بلکه در تمامی پروژه‌هایی که در آینده با آن مواجه می‌شوید، لازم است چندین بار آن‌ها را برای افراد مختلف، از کارفرما گرفته تا راهنمای پروژه و هم‌تیمی‌هایتان توضیح دهید.

ارائه یک پروژه، بدون داشتن UML درست و اصولی، شبیه بیان آرزو و خیالاتمان از نتایجی است که می‌خواهیم حاصل شود؛ بدون راه حل عملی و فکر شده! یادمان نرود، تا زمانی که یک ایده فقط در ذهن ما باشد، فقط در ذهن ما است! و تا وقتی آن را از ذهن به کاغذ نیاوریم متولد نخواهد شد. در واقع، یک UML کامل، این امکان را به شما می‌دهد که ایده و طرح خود را به صورت ملموس و قابل درکی برای خود و دیگران توضیح دهید.

جایی برای اندیشیدن

شاید بتوان گفت که از جمله مهم‌ترین ویژگی‌های فاز صفر و طراحی نمودار UML، اعمال تغییرات، به سادگی جابه‌جایی چند شکل و کلمه است، نه تغییر صدها خط کد!

این موضوع باعث می‌شود تا این مرحله از طراحی به بهترین محل برای ایده پردازی و بحث بین اعضای تیم برای چالش‌های پروژه باشد و بتوانند ساعت‌ها با صرف کم‌ترین هزینه، به بهترین ساختار برای پیاده‌سازی پروژه برسند.

این ویژگی را اصلا دست کم نگیرید چون هر چه پیش‌تر برویم، تغییر دادن ساختار و طراحی پروژه دشوارتر خواهد شد؛ تا جایی که به یک امر ناممکن تبدیل می‌شود و هر تغییر به ظاهر کوچک ممکن است به قیمت بازنویسی و پاک کردن ده‌ها و صدها خط کد تمام شود.

چه حکمتی است که در آغاز، نگاه من به سرانجام است؟

نمودار UML، به شما یک نگرش و دید کلی از پروژه و کد نهایی آن می‌دهد حتی بدون آنکه یک خط کد زده باشید! در طراحی UML، ما قرار است، قسمت اصلی فکر کردن و بررسی ایده‌های مختلف را روی کاغذ انجام بدهیم و در فازهای بعدی، کار ما تبدیل این طراحی به کد خواهد بود.

و بهترین اتفاق ممکن برای پروژه شما این است که بدانید در نهایت قرار است به کجا برسد و عناصر مختلف آن چگونه با یکدیگر ارتباط برقرار کنند.

که چه؟

داشتن یک UML کامل علاوه بر تمام مزایایی که گفته شد دو ویژگی بسیار مهم نیز دارد. تا وقتی دید کلی و جامعی به پروژه نداشته باشیم، تقسیم وظایف معمولا به دو صورت انجام می‌شود: فرایند محور، به این معنا که مثلا یک نفر بخش لاگین و یک نفر قسمت گیم پلی را پیاده‌سازی کند، یا در حالت دیگر به صورت کلاس محور، به این معنا که مثلا یک نفر کل کلاس یوزر و متعلقات آن را و دیگری کلاس نیروهای داخل بازی را بزند!

طبیعی است که هر دو روش معقول نیست و مشکلاتی را ناشی می‌شوند.

اولین مشکل آن است که قبل از طراحی UML، دشواری و بار کاری هر قسمت مشخص نیست و لذا معیار مناسبی برای تقسیم کارها نداریم.

دلیل دوم، وابستگی قسمت‌های مختلف پروژه و اتصال آن‌ها به یکدیگر است. به طوری که مثلا کلاس user باید با ده جای دیگر مرتبط باشد و تا وقتی که این ارتباط بین اجزای برنامه، مشخص نشده باشند، پیاده‌سازی مستقل و مجزای کلاس‌ها و توابع، مثل قرارگیری تعداد زیادی آجر و مشتی سیمان است که دیواری را تشکیل نداده‌اند!

مزیت دیگر ترسیم UML آن است که با داشتن یک دید جامع از کل پروژه، اعضای تیم به نیازهای هر قسمت و هر فرایند از بازی مطلع هستند و می‌دانند که تکه کدی که در حال نوشتن آن هستند قرار است در کجا و چگونه مورد استفاده قرار بگیرد. در نتیجه داشتن این آگاهی، برای قسمت‌های مختلف برنامه به نحوی برنامه‌ریزی و کدنویسی می‌کنند که در آینده و در فرایندهای آتی دچار مشکل نشوند.

و اما بعد...

تا اینجای کار سعی در تفهیم و بیان اهمیت وافر طراحی UML و فاز صفر پروژه داشتیم که حاصل تجربه و سوز دل‌های نسل‌های برنامه نویسی پیشرفته بودند.

از اینجا به بعد سعی شده تا از بعد علمی به مسئله بپردازیم و از دید یک مهندس نرم‌افزار، ویژگی‌ها و راه‌های طراحی یک UML خوب را بیان کنیم.

باز هم باگ!

در دنیای کامپیوتر هر چیزی باگی دارد و باگ در نمودار UML آن است که نسبت به سناریوهای برنامه و برخی قسمت‌های آن غافل باشید. می‌توان گفت مهم ترین وظیفه شما، تصور و شبیه‌سازی سناریوهای مختلف یک برنامه است تا نیازمندی‌های آن را شناسایی کنید و برای هر کدام از نیازها چاره‌ای بیندیشید.

در ادامه به بررسی برخی راهکارها برای کشف این نیازمندی‌ها می‌پردازیم.

قصه کاربر (user story)

خلاصه بگوییم! خودتان را به جای کاربر قرار دهید و خاطره استفاده از برنامه تان را مو به مو بنویسید. شاید خنده‌دار یا بیهوده به نظر برسد اما باور کنید دیدی که از این طریق نسبت به نرم افزارتان پیدا خواهید کرد در رفع بسیاری از مشکلات شما به کمکتان خواهد آمد.

ای بسا کاری که اول صعب گشت!

حالا به عنوان یک برنامه‌نویس خبره به سراغ برنامه بروید و

سعی کنید ابتدا داک پروژه را کامل مطالعه کنید و قسمت هایی که به نظرتان چالش‌برانگیز خواهد بود را مشخص کنید! توجه کنید که خوب است این کار را هر یک از اعضا انجام دهند. از مشخص کردن چالش ها نیز نهراسید، این چالش ها می‌توانند در نظر هر کسی از ساده‌ترین موارد تا پیچیده ترین قسمت‌های بازی متغیر باشند!

حالا وقت جلسه و هم‌فکری است، برای این فاز باید تقریبا هر روز باهم جلسه داشته باشید و برای کل پروژه از جمله چالش‌ها ایده‌پردازی کنید و با هم بحث کنید! (دقت کنیم که حجم زیادی از یادگیری ما به همین بحث‌ها وابسته است.)

مطمئن شوید که تمامی اعضای تیم در نهایت بعد از ایده دادن و انتخاب یک راه حل نهایی، دید و برداشت یکسانی از راه حل دارند و کل تیم می‌دانند که قرار است آن مشکل چطور حل شود.

کلاس بندی کنید

حالا شما برنامه‌ای که قرار هست بنویسید را یک بار از دید کاربر و یک بار از دید برنامه‌نویس مرور کرده‌اید. امیدواریم تا اینجای کار توانسته باشید نیازمندی‌ها و چالش‌های مسیر را درک و راه حل مناسبی برای آن‌ها پیدا کنید.(توجه کنید که با هر بار بررسی مراحل بالا، دید عمیق‌تر و شفاف‌تری از نتیجه نهایی پروژه کسب خواهید کرد.)

حال وقت آن رسیده که با ساده ترین روش، مسئله را روی کاغذ بیاورید و کلاس‌ها، متغیرها و توابعی که نیاز دارید را ایجاد کنید. در این مرحله، روی هر تابع، کلاس و متغیری که می‌نویسید بازنگری کنید چون ممکن است مواردی که برای مسئله‌تان مشخص کرده‌اید، برخی از چالش و نیازهای شما را پوشش ندهند. پس سعی کنید به صورت تیمی چندین مرتبه کلاس‌ها، روابط بین آن‌ها، پارامترها و توابعتان را مرور کنید تا علاوه بر رسیدن به یک دید جامع و مشترک در تیم، برای گام بعدی پروژه آماده شوید!

به وقت طراحی

در مرحله آخر و حتی در هنگام مرور کردن، هم برای داشتن نظم بیشتر و داشتن یک نقشه راه UML ای که بر روی کاغذ طراحی کردید را با استفاده از نرم افزارها و سایت هایی مثل:

- Software ideas modeler
- وب سایت draw.io
- وب سایت lucidchart
- Microsoft visio

و این قبیل موارد طراحی و پیاده سازی کنید.

خب! همه چیز حاضر است. پروژه شما تقریبا تمام شده و شما آن را یکبار با تمام جزئیات داخل ذهنتان ساخته‌اید. فقط مانده نوشتن کدها بر اساس این نقشه‌راهی که طراحی کرده‌اید.

امیدواریم این متن توانسته باشد اهمیت UML را به شما برساند و جای هیچ حسرتی برای UML بهتر در آخر پروژه را برایتان باقی نگذاشته باشد!

قواعد پیشرفته UML

سید پارسا نشایی

شما از طریق کلاس‌های حل تمرین و نیز انجام تمرین دوم درس برنامه‌سازی پیشرفته، با بسیاری از نکات مهم UML آشنا هستید، مانند:

- **نحوه‌ی جداسازی نام کلاس، خواص و متدها**
- **نحوه‌ی مشخص کردن سطح دسترسی و static بودن**

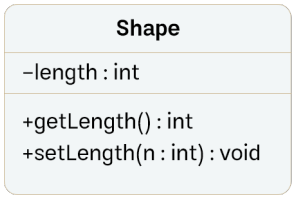
در طراحی یک پروژه، مشخص کردن «ارتباط» میان کلاس‌ها از اولویت فراوانی برخوردار است. در این مقاله، با نکاتی در این رابطه، آشنا می‌شوید.

قواعد نوشتن «نام کلاس»

نام کلاس باید به شکل PascalCase در بالا و وسط نوشته شود. اگر کلاس از نوع abstract است، اسم آن باید *italic* نوشته شود.

قواعد نوشتن خواص (property ها) و متدها

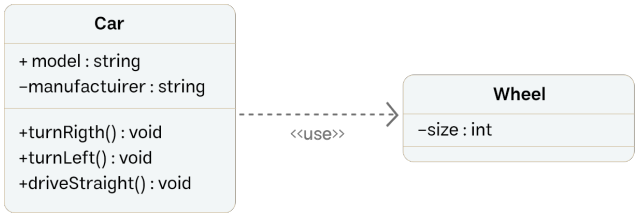
ابتدا باید سطح دسترسی خاصیت مشخص شود (+ برای public، - برای private، # برای protected و ~ برای package level) و سپس اسم داده نوشته شده و در نهایت پس از یک دو نقطه (:)، نوع آن مشخص می‌شود. اگر فیلد از نوع static است، زیر کل عبارت باید یک underline کشیده شود. قواعد نوشتن متدها نیز مشابه خواص است، با این تفاوت که پارامترهای ورودی نیز باید مشخص شوند. به مثال زیر توجه کنید:



روابط میان کلاس‌ها و نحوه‌ی مشخص کردن آن‌ها در UML

رابطه‌ی وابستگی

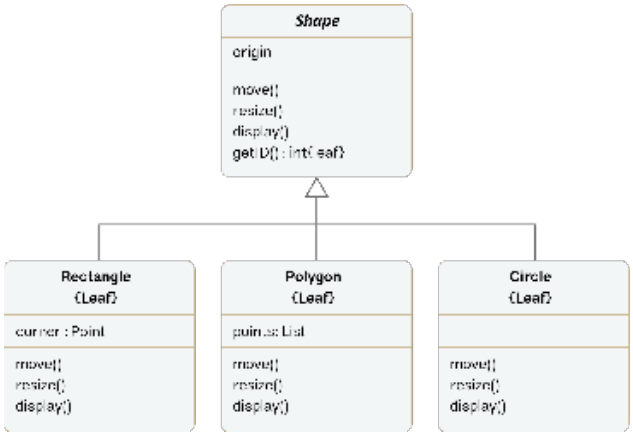
زمانی که دو کلاس از نظر مفهومی به هم وابسته‌اند و تغییر در خواص یک کلاس، ممکن است خواص کلاس دیگر را تغییر دهد و یا آن را دچار مشکل کند؛ به عنوان مثال، کلاس‌های Car و Wheel در شکل زیر:



نحوه‌ی نمایش این نوع از رابطه، یک فلش خط‌چین‌دار از کلاسی است که نیازمند کلاس دیگر است، به سمت کلاسی که نیازش را برطرف می‌کند. در زیر این خط چین، <<use>> نوشته می‌شود.

رابطه‌ی تعمیم و وراثت

هر جا که کلاسی از پدرش ارث می‌برد (یا به عبارت دیگر، extend می‌کند)، لازم است از این رابطه استفاده شود.



نحوه‌ی نمایش این رابطه، یک فلش است که انتهای آن، مثلثی توخالی قرار دارد. در مثال فوق، همه‌ی انواع Polygon، Circle یا Rectangle، نوعی Shape (که یک کلاس abstract است و در نتیجه italic نوشته شده) هستند و در نتیجه، رابطه‌ی میان آن‌ها، از نوع وراثت است. با وراثت در کلاس‌های درس، بیش‌تر آشنا خواهید شد.

توجه کنید که این رابطه تنها معادل extend کردن است و نه معادل implement کردن که در interface ها استفاده می‌شود؛ برای واسط‌ها، رابطه‌ای دیگر به زودی گفته خواهد شد.

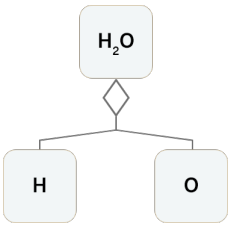
رابطه‌ی تجمعی (Aggregation)

بسیاری از روابط، رابطه‌ی میان «کل» و «اجزا»ی یک پدیده را نشان می‌دهند؛ مثلا رابطه‌ی «دانشگاه» و «دانش‌جو». دیدیم اگر این رابطه به حدی شدید باشد که تغییر در یکی، دگرگونی بسیار دیگری را نتیجه دهد (مثلا اگر بخواهیم از نوع دیگری از چرخ استفاده کنیم، طراحی ماشین دگرگون می‌شود)، از رابطه‌ی وابستگی استفاده می‌شود، اما اگر این رابطه به گونه‌ای باشد که کلاس‌ها کاملا به هم وابسته نباشند، از رابطه‌ی تجمعی بهره گرفته می‌شود.

نحوه‌ی نمایش این رابطه، یک خط با یک شکل همانند لوزی توخالی است.

رابطه‌ی ترکیب (Composition)

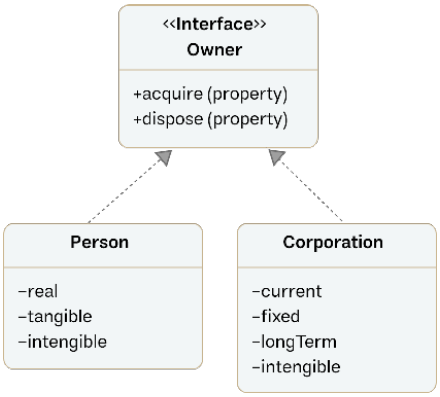
این رابطه، نسخه‌ی قوی‌تر رابطه‌ی تجمعی است، با این تفاوت که هیچ‌یک از دو کلاس دو طرف رابطه، بدون هم معنایی ندارند. در رابطه‌ی «وابستگی»، تنها پدر بدون فرزند بی‌معناست و فرزندان بدون پدر معنا دارند؛ اما در این رابطه (ترکیب)، فرزندان نیز بدون پدر معنایی ندارند.



در مثال فوق، اگر در نظر بگیریم که برنامه‌مان تنها با «مولکول»ها کار دارد و نه با اتم‌ها، آن‌گاه H و O بدون مولکولی که شامل آن‌هاست، معنایی ندارند و خود مولکول نیز بدون اجزایش، معنایی ندارد. این رابطه با یک لوزی توپر روی خط نشان داده می‌شود.

رابطه‌ی تفهیم یا تحقق (Realization)

از این رابطه، برای ارتباط دادن کلاس‌هایی به یک کلاس دیگر استفاده می‌شود که وظیفه‌ی تکمیل کردن آن کلاس دیگر را داشته باشند. برای ارتباط enum هایی که در یک کلاس استفاده می‌شوند به آن کلاس، از این رابطه بهره گرفته می‌شود. هم‌چنین، کلاس‌هایی که یک interface را پیاده می‌کنند نیز به کمک این رابطه به interface متصل می‌شوند. نحوه‌ی نمایش interface و enum در نمودار UML، مشابه کلاس است، با این تفاوت که عبارت «Interface» و یا «Enum»، همان‌طور که در شکل مشخص است، در بالای box نوشته می‌شود.



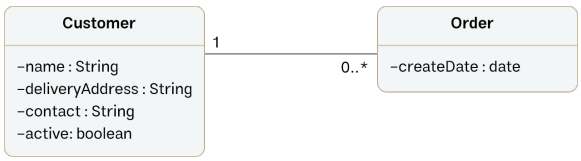
نحوه‌ی نمایش این رابطه، یک فلش خط‌چین‌دار با یک مثلث توپر است.

رابطه‌ی عادی (Normal)

اگر رابطه‌ی میان دو کلاس در هیچ‌یک از حالات فوق جای نمی‌گرفت، از رابطه‌ی نرمال استفاده می‌کنیم که شامل یک خط بدون هیچ نمادی است.

چندی (multiplicity) روابط

می‌توانید تعداد مواقعی که هر کلاس نیازمند دیگری است را با یک عدد روی خط نمایش دهید؛ مثلا هر customer می‌تواند تعداد زیادی order داشته باشد، اما هر order تنها به یک customer مرتبط است. در این حالت، در خط میان دو کلاس Customer و Order، دو عدد می‌نویسیم:



عدد ۱ روی خط یعنی هر سفارش فقط یک مشتری دارد، و عدد ۰..* یعنی هر مشتری، از صفر تا بی‌نهایت سفارش دارد؛ اگر مثلا ماکسیمم ۴ سفارش ممکن بود، می‌توانستیم از ۰..۴ استفاده کنیم.