به نام خدا

دانشگاه صنعتی شریف
درس طرّاحی شی‌گرای سیستم‌ها

تمرین شماره‌ی ۱

امیرمحمّد فخیمی
99170531

بهار ۱۴۰۳

۱)

الف:

- Assosiation: یک رابطه‌ی مانا (persistant) بین کلاس‌ها است که یعنی کلاس‌ها در طول زمان به همدیگر دید دارند و ارتباط (دید) آن‌ها با یکدیگر در طول زمان حذف نمی‌شود. حال این رابطه یا یک طرفه است، یا دو طرفه و یا reflexive. در دو طرفه هر دو طرف به همدیگر دید دارند. در یک طرفه سمت بدون فلش به سمت فلش‌دار دید دارد. در reflexive دید یک کلاس نسبت به خودش نشان داده شده است. به روی این رابطه چندی کلاس‌ها در ارتباط با یکدیگر و همچنین role name نمایش داده می‌شوند.
- Aggregation: یک رابطه‌ی کل به جزء را نشان می‌دهد که سمت لوزی‌دار کلاس کل و سمت دیگر کلاس جزء است. دید نیز معمولاً از کل به جزء است. اگر یک رابطه‌ی association یکی از حالات زیر را داشته باشد، رابطه‌ی aggregation خواهد بود:
- Assembly: یک کلاس از تعدادی از instanceهای کلاس دیگر تشکیل شده باشد مانند رابطه‌ی ماشین و قطعاتش.
- Containment: یک کلاس حاوی instanceهای کلاس دیگر باشد مانند رابطه‌ی یک container و محتویاتش.
- Membership: یک کلاس عضوی از یک کلاس دیگر باشد مانند رابطه‌ی یک گروه و عضوهایش.
- Inheritance (generalization/specialization): با این ارتباط می‌توانیم gen/spec را نشان دهیم. توجه کنید که هر رابطه‌ی is a یک gen/spec و در gen/spec نیز توارث نهفته است ولی هر توارثی gen/spec نیست چون ما می‌توانیم در کد توارث را به هر نوعی تعریف کنیم و لزوماً آن‌ها gen/spec نیستند. وقتی refuse request داشته باشیم، این مورد پدید می‌آید. اگر ما از کلاس‌های بچّه، پدر را بسازیم generalization داشته‌ایم و اگر از پدر، بچّه‌ها را استخراج کنیم، specialization داشته‌ایم.

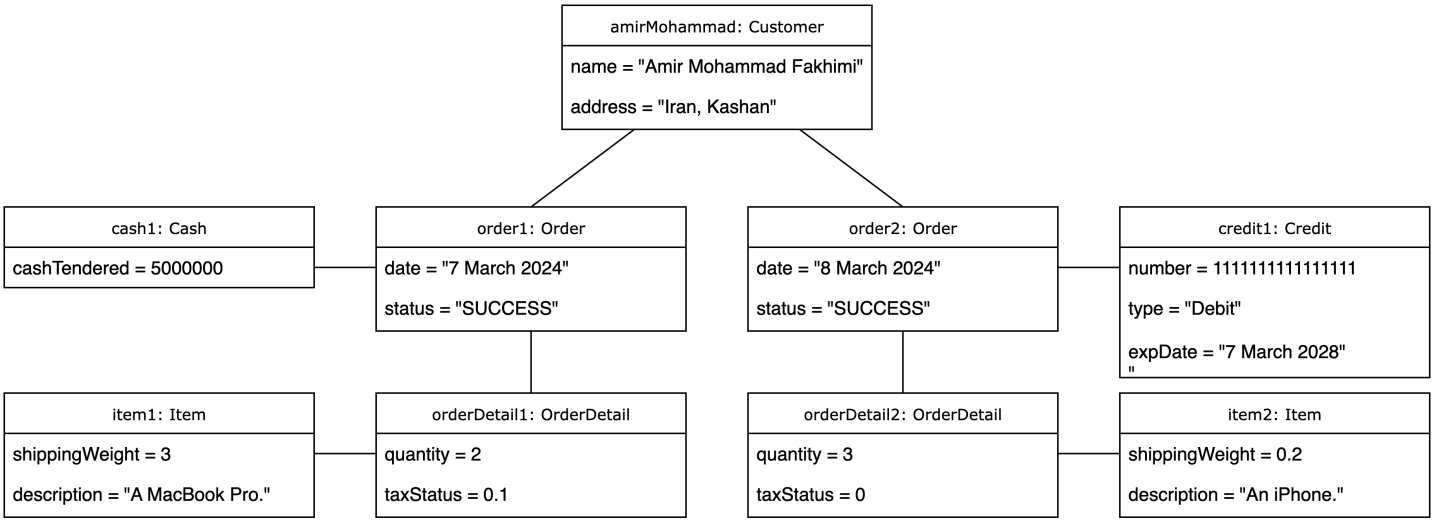
ارتباط‌های زیر در نمودار گفته شده وجود ندارند:

- Reflexive Association
- Composition
- Realization/Implementation
- Dependency

منبع ۱، منبع ۲

ب: در نمودار زیر یک Customer وجود دارد که دو سفارش دارد. هر دو سفارش در یک قسط پرداخت شده‌اند. یکی از سفارش‌ها با استفاده از Cash پرداخت شده و دیگری با استفاده از Credit

پرداخت شده. هر order یک OrderDetail و هر OrderDetail باید یک Item داشته باشد که در نمودار زیر آورده شده‌اند:

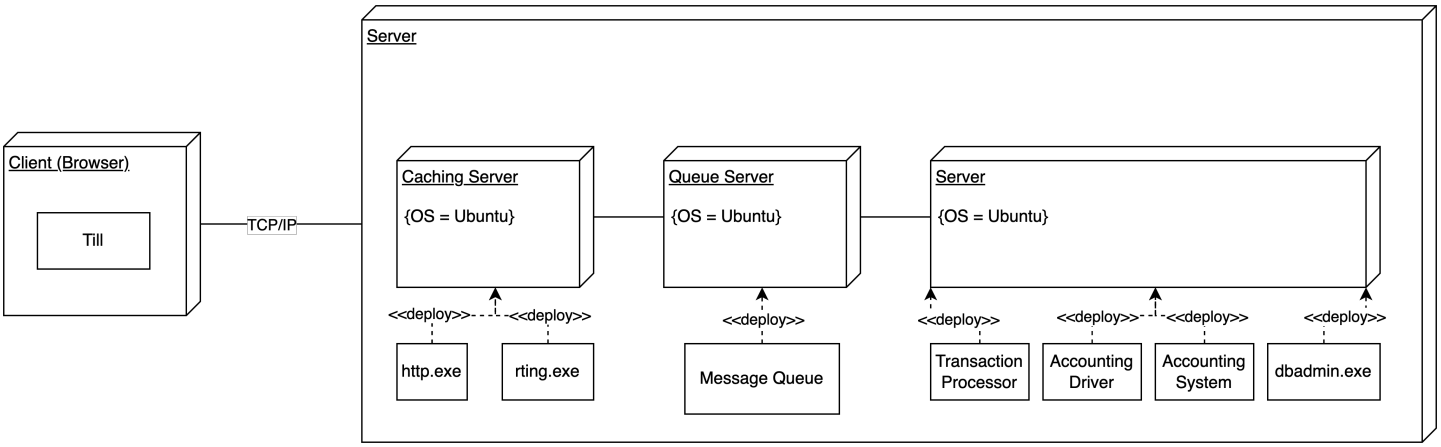


۲)

الف:

- هدف از wrap کردن، آن است که آن بخش نیز خودش به عنوان یک component در نظر گرفته شود. در نمودار داده شده خود Sales Server یک component است که دارای APIهای خاصّ خود است و می‌تواند با یک component دیگر که همان APIها را سرویس دهد، جابه‌جا شود. این کار باعث می‌شود تا ما به تدریج componentهای درشت‌دانه‌تر داشته باشیم و بتوانیم در سطوح متفاوت، پیچیدگی‌های پیاده‌سازی پروژه را بررسی کنیم. همچنین در سطوح متفاوت از abstraction، می‌توانیم تحلیل‌های متفاوت بدون نیاز به در نظر گرفتن پیچیدگی‌های اضافه، ارائه دهیم.
- این component یک سرویس به نام sales message به بیرون از خود ارائه می‌دهد که بقیه‌ی componentها یا اجزای پروژه می‌توانند از آن استفاده کنند. همچنین این component از سرویسی که Accounting System ارائه می‌دهد نیز استفاده می‌کند.
- به صورت ریزدانه‌تر، این component دارای دو component دیگر به نام‌های Transaction Processor و Accounting Driver است. سرویس sales message توسّط Transaction Processor ارائه می‌شود و سرویسی که توسّط Accounting System ارائه می‌شود، توسّط Accounting Driver استفاده می‌شود. در نهایت Accounting Driver نیز یک سرویس به Transaction Processor ارائه می‌کند.

ب: در نمودار زیر یک نمونه از مدل کردن سیستم client/server آورده شده است (توجّه کنید که مستطیل‌های بی‌علامت که در مکعب مستطیل‌ها هستند، هر کدام یک artifact هستند. همچنین این نمودار با کمک از منبع داده شده رسم شده است.):



منبع

۳)

ابتدا تمام نمودارهای گفته شده را از هر دو جهت توضیح می‌دهیم و در نهایت مقایسه‌های خواسته شده را انجام می‌دهیم.

Use-Case Diagram:

این نمودار ارتباطات بین اجزای بیرونی سیستم (بازیگران) و سیستم را نشان می‌دهد. اجزای اصلی نمودار عبارتند از:

- **Actors:** یک کاربر یا سیستم خارجی که با سیستم ما در تعامل است. این موارد در نمودار به صورت آدمک نشان داده می‌شوند. دو نوع primary و secondary داریم که از مورد اوّل برای trigger کردن use-caseها و از مورد دوم برای قرارگیری در جریان use-caseها استفاده می‌شود.
- **Use-cases:** وظیفه‌ی خاصّی است که سیستم ما انجام می‌دهد. این موارد در نمودار به صورت بیضی نشان داده می‌شوند. برای هر use-case در کنار نمودار، جدول‌های استانداری می‌آیند که موارد متفاوتی از use-caseها را از جمله preconditionهای آن، primary & secondary actors، event flows و ... را نشان می‌دهند.
- Relationships: تعاملات در نمودار را نشان می‌دهند که انواع زیر را دارد:
- Association: نشان‌دهنده‌ی یک ارتباط بین actor و use-case است.

- Include: از این مورد برای اشتراک‌گذاری یک use-case استفاده می‌شود. اگر چند use-case یک بخش مشترک داشتند، می‌توانیم آن را جدا کنیم و از رابطه‌ی include استفاده کنیم. توجّه کنید که از این رابطه نباید برای شکستن یک use-case بدون وجود داشتن هیچ دلیل اشتراک‌گذاری‌ای استفاده کرد. در صورت استفاده‌ی غلط به اصطلاح fan out و در صورت استفاده‌ی درست fan in داریم.
- Extend: از این رابطه برای نشان دادن یک رفتار خاص از یک use-case استفاده می‌کنیم.
- **Subject:** به هر زیرسیستم از سیستم اصلی که که در حال کشیدن use-case برای آن هستیم، subject گفته می‌شود. در گذشته به دلیل کوچک بودن سیستم و نبودن نیاز به شکاندن سیستم به زیرسیستم‌ها، به آن system boundary گفته می‌شده.

هدف از این نمودار نشان دادن رفتار سیستم از منظر بازیگران بیرونی آن است. با استفاده از این نمودار می‌توانیم سرویس‌های سیستم را ببینیم، متوجّه شویم کدام بازیگران به چه سرویس‌هایی نیاز دارند و در نهایت متوجّه شویم که آیا تمام نیازمندی‌ها به درستی پیاده‌سازی شده‌اند یا خیر. همچنین نمایی سطح بالا از عملکرد سیستم و سرویس‌هایی که ارائه می‌دهد نشان می‌دهد. توجّه کنید که این نمودار در اصل functional است چون توالی انجام use-caseها مشخّص نیست.

Interaction Overview Diagram:

این نمودار ارتباطات بین اجزای سیستم را در شرایط متفاوت نشان می‌دهد. اجزای اصلی نمودار عبارتند از:

- **نقطه‌ی شروع:** یک یا چند نقطه در نمودار که نشان‌دهنده‌ی محلّ شروع نمودار است.
- **نقطه‌ی پایان:** یک یا چند نقطه در نمودار که نشان‌دهنده‌ی محلّ پایان نمودار است.
- **Nodes:** گره‌های موجود در نمودار که هر یک می‌توانند یک iteration diagram دیگر باشند. البتّه timing diagram به این منظور زیاد استفاده نمی‌شود ولی ما می‌توانیم حتّی interaction overview diagramهای nested داشته باشیم. البتّه می‌توانیم هر node را باز نکنیم و به جای آن reference به جایی که نمودار آن موجود است، بدهیم.
- **Others:** نقاطی هم در نمودار وجود دارند که در آن‌ها یا نمودار شکسته و حالت‌بندی می‌شود یا شاخه‌های متفاوت نمودار به یکدیگر می‌رسند.

هدف از این نمودار نشان دادن ترتیب مجاز اجرای use-caseها است؛ برای مثال login قبل از پرداخت صورت حساب باید انجام شود. در اصل این نمودار برای نشان دادن flow of control استفاده می‌شود. این نمودار برای مدل‌سازی فرایندهای پیچیده یا جریان‌های کاری‌ای که decision pointهای متعدّدی دارد، مفید است. همچنین از این نمودار می‌توان برای نشان دادن رفتار پویای سیستم استفاده کرد و متوجّه شد که سیستم به هر ورودی، به طور پویا چه خروجی‌ای می‌دهد.

Timing Diagram:

این نمودار حضور هر object را در هر state در هر زمان نشان می‌دهد. این نمودار دو نوع دارد و هر کدام هم حالت expand و هم حالت compact دارند. به طور کلّی در حالت compact پرت فضا کم‌تر است. اجزای اصلی نمودار عبارتند از:

- **Object:** شی‌های موجود در سیستم. پیام‌های ردّ و بدل شده بین objectها در نمودار قابل مشاهده است.
- **State:** هر object یک سری state یا حالت دارد.
- **Event:** یک ورودی به سیستم است که ممکن است منجر به یک یا چند state change در بعضی از objectها بشود.
- **Timing constraint:** یک محدودیت زمانی در سیستم است که در نمودار نشان می‌دهیم. برای مثال می‌توانیم در این نمودار نشان دهیم که از زمانی که درخواست login می‌آید تا زمانی که کاربر login می‌شود نباید بیش‌تر از یک ثانیه شود.

هدف اصلی از این نمودار، نشان دادن محدودیت‌های زمانی‌مان است. با استفاده از این نمودار می‌توانیم محدودیت‌های زمانی، objectها، stateهای آن‌ها را در هر لحظه از سیستم داشته باشیم و objectها را در زمان track کنیم. همچنین در این نمودار توالی انجام کارها نیز مشخّص است.

Sequence Diagram:

در این نمودار توالی انجام کارها نشان داده می‌شود. اجزای اصلی نمودار عبارتند از:

- **Object:** شی‌های موجود در سیستم.
- **Life Line:** خطّی است که نشان‌دهنده‌ی خطّ زندگی هر object است. برای هر object یکی از این خطوط وجود دارد.
- **Activation bar:** نشان می‌دهد که یک object در کدام مراحل فعّال بوده است. توجّه کنید که ممکن است یک object چند بار همزمان فعّال باشد که باعث می‌شود چند activation bar به روی life line آن object قرار داشته باشد.
- **Deletion:** زمان پایان فعّالیت یک object است که با علامت ضربدر نشان داده می‌شود.
- **Message:** پیامی است که بین objectها مبادله می‌شود.
- **Object creation:** اگر فلشی در نمودار به جای آنکه به life line ختم شود، به خود object ختم شود، آنگاه این مورد نشان دهنده‌ی یک initialization از object است.
- **Iteration:** یک حلقه را نشان می‌دهد که تعداد تکرار ان را در نمودار می‌توانیم مشخّص کنیم. همچنین امکان تعریف متغیّر نیز در نمودار وجود دارد.
- **Condition:** نشان‌دهنده‌ی یک شرط در نمودار است. برای مثال یک شرط برای درست کردن object.
- **Note:** هر گونه متن برای اضافه کردن به نمودار.

هدف اصلی از این نمودار، نشان دادن توالی انجام کارها و پیام‌ها است. به نوعی می‌توانیم چگونگی تحقّق هر use-case را در این نمودار ببینیم.

مقایسه‌ها

الف: Interaction Overview Diagram

- در ساختار تا حدّ خوبی متفاوت هستند. تنها نکته آن است که در use-case diagram، موارد use-case در بیضی‌هایی آورده شده‌اند و هر interaction overview diagram برای یک use-case کشیده شده است.
- هدف‌های این دو نمودار نیز همان‌طور که توضیح داده شده، با یکدیگر متفاوت هستند و اشتراک خاصّی ندارند.

ب: Timing Diagram

- در ساختار با توجّه به توضیحات داده شده متفاوت هستند. درشت‌دانه‌ترین عنصر timing diagramها object هستند در صورتی که ریزدانه‌ترین عنصر در use-case diagramها use-case هستند. به طور کلّی اشتراک ساختاری خاصّی وجود ندارد.
- هدف‌های این دو نمودار نیز همان‌طور که توضیح داده شده، با یکدیگر متفاوت هستند و اشتراک خاصّی ندارند.

ج: Sequence Diagram

- در ساختار با توجّه به توضیحات داده شده متفاوت هستند. درشت‌دانه‌ترین عنصر sequence diagramها object هستند در صورتی که ریزدانه‌ترین عنصر در use-case diagramها use-case هستند. به طور کلّی اشتراک ساختاری خاصّی وجود ندارد.
- هدف‌های این دو نمودار نیز همان‌طور که توضیح داده شده، با یکدیگر متفاوت هستند و اشتراک خاصّی ندارند.

توجّه کنید که تمام نمودارهای گفته شده برای مقایسه، interaction diagram هستند که اشتراکاتی در اهداف و نحوه‌ی نشان دادن آن‌ها وجود دارد. تقریباً اشتراک خاصّی بین use-case diagram و نمودارهای گفته شده وجود ندارد چون هر کدام برای یک هدف خاص تنظیم و تهیه شده‌اند ولی در کنار یکدیگر می‌توانند مکمّل‌های عالی برای توصیف و مدل‌سازی یک سیستم باشند.