دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

تمرین درس الگوها در مهندسی نرم‌افزار

تمرین شماره ۲

نگارش
یحیی پورسلطانی

استاد درس
دکتر رامان رامسین

خرداد ۱۴۰۰

بسم الله الرحمن الرحيم

فهرست مطالب

فصل۱: طرح مسئله و زيرمسائل آن ۱

۱.۱ طرح مسئله ۱

۲.۱ زيرمسئله‌ها و الگوهاي مناسب براي هركدام از زيرمسائل ۱

۱.۲.۱ توزيع‌شدگي ۲

۲.۲.۱ ساختار سلسله‌مراتبي ۲

۳.۲.۱ تعامل پيچيده در بين اجزايي كه بهم ديد دارند ۲

۴.۲.۱ رفتار وابسته به حالت و تغيير روش تعامل، بر مبناي آن ۳

۵.۲.۱ كاهش وابستگي ۳

۳.۱ تشريح معماري پيشنهادي ۳

۱.۳.۱ ساختار زيرسيستم‌ها ۴

۲.۳.۱ سناريوها ۶

۳.۳.۱ نقدهايي بر معماري ۸

فصل۲: بررسي دو Idiom در جاوا ۱۰

۱.۲ Favor static member classes over nonstatic ۱۰

۱.۱.۲ مقصود ۱۰

۲.۱.۲ انگيزه ۱۰

۳.۱.۲ راهكار ۱۰

۲.۲ Limit source files to a single top-level class ۱۱

۱.۲.۲ مقصود ۱۱

۲.۲.۲ انگيزه ۱۱

۳.۲.۲ راهكار ۱۱

فهرست تصاویر

شکل ۱.۱: نمودار مولفه‌اي سيستم ۳
شکل ۲.۱: نمودار ساختار کلاسي..... ۵
شکل ۳.۱: سناريوي تغيير وضعيت کل زيرسيستم‌ها ۷
شکل ۴.۱: سناريوي تغيير استراتژي تعامل ۸
شکل ۱.۲: نمونه‌اي از عدم رعايت قاعده‌ي اول..... ۱۱
شکل ۲.۲: نمونه‌اي از عدم رعايت قاعده‌ي دوم..... ۱۲

فهرست جداول

فصل ۱

طرح مسئله و زیرمسائل آن

در این بخش، مسئله‌ي اصلي را مطرح کرده و زیرمسائل آن را تشریح مي‌کنیم و براي هر زیرمسئله، الگوهاي مناسب را معرفي مي‌کنیم.

۱.۱ طرح مسئله

یک سیستم توزیع شده، متشکل از سلسله مراتبی از زیرسیستم‌ها (مولفه‌ها) است، که در آن هر زیرسیستم سطح بالا، از تعدادی زیرسیستم سطح پایین‌تر تشکیل شده‌است. در پایین‌ترین سطح، کلاس‌ها قرار دارند. در همه‌ي سطوح، اجزایی به یک دیگر دید دارند و به صورتی پیچیده با هم در تعامل هستند. الگوریتم تعامل، به حالت اجزاي درگیر بستگی داشته و متغیر است. حالت هر زیرسیستم مرکب نیز به حالت اجزاي داخلی‌اش بستگی دارد. می‌خواهیم سیستم طوری طراحی شود که وابستگی بین اجزا کم باشد و الگوریتم‌های تعامل را بتوان به راحتی و در زمان اجرا تغییر داد.

۲.۱ زیرمسئله‌ها و الگوهاي مناسب براي هرکدام از زیرمسائل

در ادامه، مسئله‌ي طرح شده در بخش قبل را به زیرمسائلي تقسیم مي‌کنیم و براي هر زیرمسئله، یک یا چند الگو را پیشنهاد مي‌کنیم.

۱.۲.۱ توزيع‌شدگي

اولين چالشي که در اين سيستم وجود دارد، توزيع‌شدگي آن است. با توجه به اين‌که چندين زيرسيستم داريم که با يک ديگر در تعامل هستند، استفاده از الگوي معماري Broker مناسب‌تر به‌نظر مي‌رسد؛ چرا که در صورت مسئله مشاهده مي‌کنيم که برخي از زيرسيستم‌ها (بر اساس سلسله مراتب) به يک‌ديگر ديد دارند و حتي ممکن است با يک ديگر، متجانس نباشند. وجود اجزايي مثل Bridge و Broker با ايجاد سطوحي از غيرمستقيم‌سازي [1] مي‌توانند وابستگي شديد بين زيرسيستم‌ها را تا حد خيلي خوبي مديريت کنند. همچنين، با توجه به اين که رفتار سيستم بر مبناي رفتار زيرسيستم‌ها تعيين مي‌شود، استفاده از اين الگوي معماري و ترکيب آن با الگوهاي ديگر (در بخش‌هاي بعدي بحث مي‌شود) مي‌تواند به ما کمک کند تا براي هر زيرسيستم، هويتي مجزا قائل باشيم.

۲.۲.۱ ساختار سلسله‌مراتبي

همان‌طور که در صورت مسئله مشاهده مي‌کنيم، سيستم ما از سلسله مراتبي از زيرسيستم‌ها تشکيل شده‌است. اين بدان معنا خواهد بود که ارتباط بين زيرسيستم‌ها، منوط به جايگاه آن‌ها در سيستم اصلي است و بدين‌وسيله، مشخص مي‌شود که هر زيرسيستم، با کدامين زيرسيستم مرتبط مي‌شود؛ با توجه به وجود سلسله مراتب، به‌نظر مي‌رسد که استفاده از الگوي معماري نمايش ـ انتزاع ـ کنترل [2] يا PAC گزينه‌ي مناسبي باشد. اما با توجه به اين‌که در اين معماري، سخني از نمايش داده‌ها به ميان نيامده است مي‌توان از آن استفاده نکرد و با توجه به بخش بعدي، از الگوي Mediator به منظور برقراري ارتباط سلسله مراتبي استفاده نمود.

۳.۲.۱ تعامل پيچيده در بين اجزايي که بهم ديد دارند

همان‌طور که در صورت مسئله قيد شده‌است، در بين زيرسيستم‌هايي که به يک‌ديگر ديد دارند، الگوريتم تعاملي پيچيده‌اي وجود دارد و اين زيرسيستم‌ها، به نحوي پيچيده با يک‌ديگر در تعامل هستند؛ همچنين در بخش ديگري از مسئله قيد شده‌است که الگوريتم تعامل، بسته به شرايط مسئله مي‌تواند تغيير کند و اين، بدان معنا خواهد بود که بهتر است که الگوريتم تعامل، توزيع‌شده نباشد و تا حد امکان متمرکز باشد؛ بنابراين، بهتر است که اجزاي مرتبط بهم(زير سيستم‌هاي مرتبط با يک ديگر) را با واسطه‌ي يک عنصر کارچرخان به يک‌ديگر مرتبط کنيم و عنصر کارچرخان، ارتباط بين اين اجزا را باهم

[1] Indirection

[2] Presentation - Abstraction - Controll (PAC)

برقرار كند؛ براي اين منظور، الگوي Mediator را پيشنهاد مي‌كنيم. مي‌توانيم بخش Bridge هر گره‌ي پردازشي را به يک Mediator متصل كنيم تا ارتباط بين آن‌ها، از اين طريق برقرار شود.

۴.۲.۱ رفتار وابسته به حالت و تغيير روش تعامل، بر مبناي آن

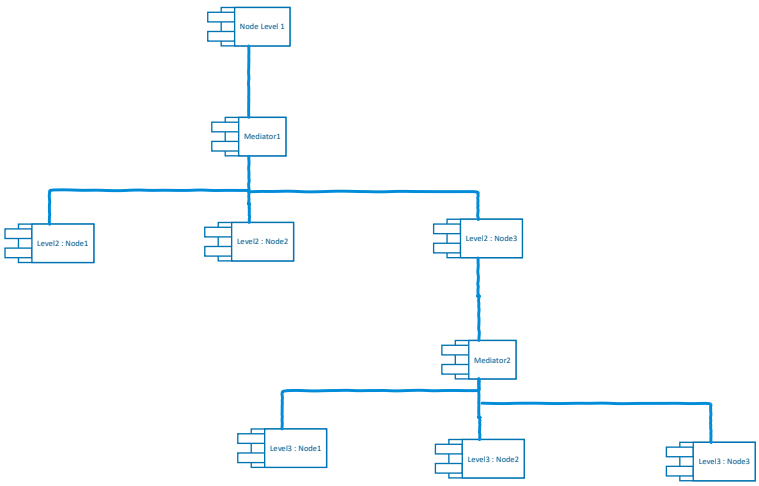
در بخش ديگري از صورت مسئله، مشاهده مي‌كنيم كه در خصوص نحوه‌ي تغيير الگوريتم تعامل صحبت شده‌است. مي‌توانيم از تلفيقي از الگوهاي State و Strategy استفاده كنيم. بدين صورت كه حالت هر جزء و هر زيرسيستم را در قالب يک State نگهداري كنيم و راهكارهاي تعامل را با استفاده از الگوي Strategy و در Mediator پياده‌سازي كنيم.

۵.۲.۱ كاهش وابستگي

براي اين منظور، مي‌توانيم از الگوي Proxy استفاده كنيم. پروكسي‌ها را مي‌توان براي جلوگيري از ارتباط مستقيم كلاس‌هاي كلاينت، سرور و كارچرخان با Broker به كار برد.

۳.۱ تشريح معماري پيشنهادي

در شكل ۱.۱ نمودار مولفه‌اي معماري پيشنهادي را مشاهده مي‌كنيم.



شكل ۱.۱: نمودار مولفه‌اي سيستم

همان‌طور که در شکل مشاهده مي‌کنید، هر کدام از زیرسیستم‌ها را یک مولفه در نظر گرفته‌ایم و ارتباط بین مولفه‌هاي هر سطح (و نیز، مولفه‌هایي که از سطوح بالاتر و یا پایین‌تر سلسله مراتب قصد ارتباط با دیگر زیرسیستم‌ها را دارند) از طریق یک مولفه‌ي کارچرخان صورت مي‌گیرد و بدین‌وسیله، مي‌توان ارتباطات پیچیده‌ي بین آن‌ها را به صورت پویا تغییر داد. بنابراین مي‌توان با اطمینان گفت که هر مولفه (زیر سیستم) با واسطه‌ي یک عنصر کارچرخان با زیرسیستم دیگر در ارتباط است.

۱.۳.۱ ساختار زیرسیستم‌ها

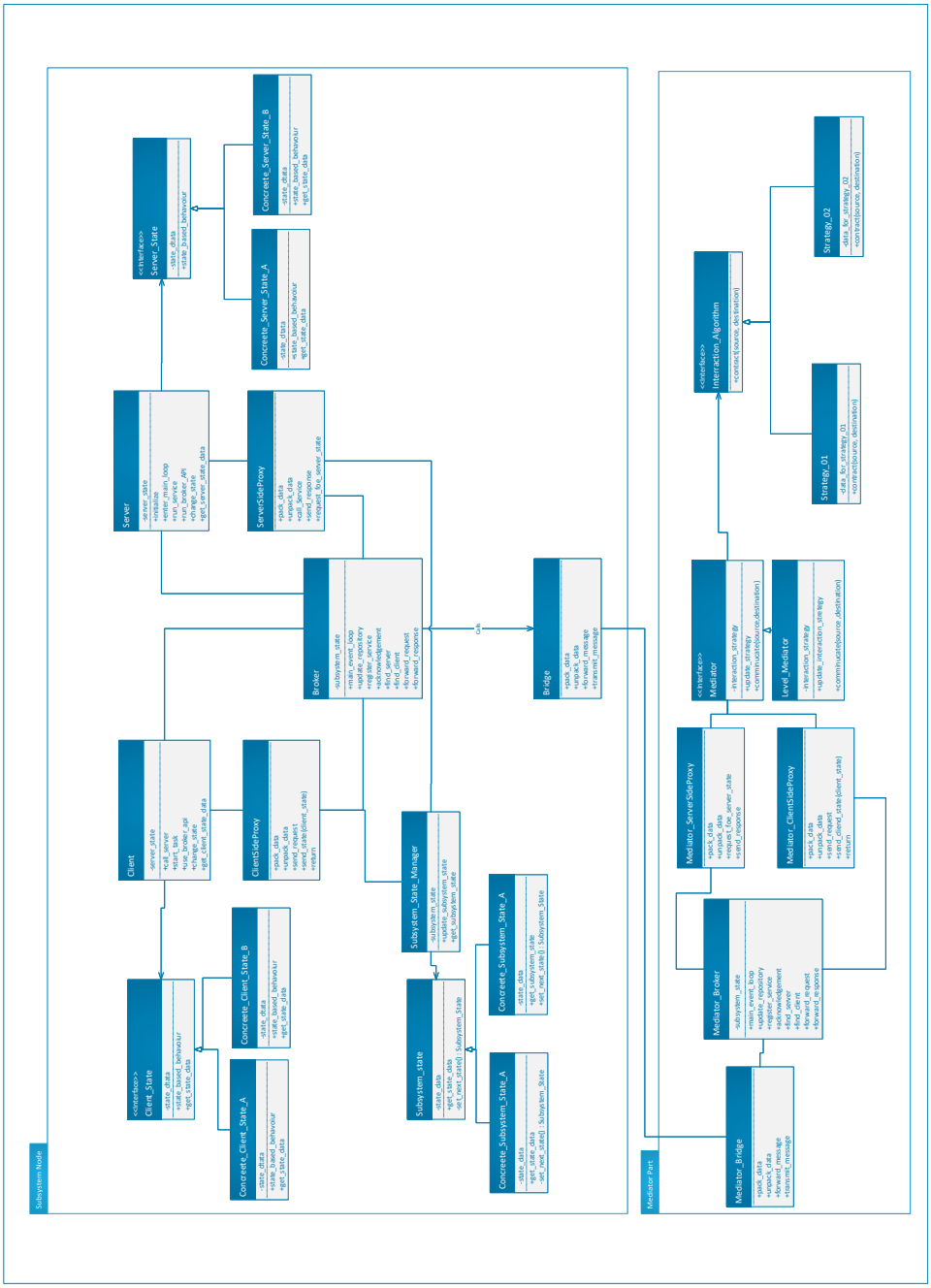
براي درک بهتر ساختار زیرسیستم‌ها، به نمودار ساختار کلاسي آن توجه کنید. همانطور که مشاهده مي‌کنیم، دو نوع زیر سیستم اساسي داریم :

- زیر سیستم‌هاي کارچرخان
- زیرسیستم‌هاي اصلي

زیرسیستم‌هاي اصلي

این زیرسیستم‌ها، مشتمل هستند بر کلاس‌هایي که مي‌توانند کلاینت و یا سرور باشند. هر کدام از کلاس‌هاي کلاینت و یا سرور، کوچکترین (پایین‌ترین سطح) اجزاي سیستم هستند. هرکدام از این زیرسیستم‌ها، از الگوي Broker تبعیت مي‌کنند. هرکدام از کلاینت‌ها و سرورها، به واسطه‌ي یک پراکسي به این عناصر متصل هستند و این عناصر به برقراري ارتباط محلي بین این اجزا، کمک مي‌کند. همان‌طور که مي‌دانیم، وضعیت کلي زیرسیستم بر مبناي وضعیت هرکدام از این کلاس‌هاي کوچک تعیین مي‌شود؛ بنابراین لازم است که وضعیت هر کدام از این کلاس‌ها را در خودشان نگهداري کنیم. براي این منظور، از الگوي State استفاده مي‌کنیم؛ بنابراین، این الگو را هم در کلاس‌هاي کلاینت و هم در کلاس‌هاي سرور، به کار مي‌گیریم.

همان‌طور که در صورت مسئله دیدیم، وضعیت کلي هر زیرسیستم نیز براي تعیین روش تعامل بین زیرسیستم‌ها، موثر است؛ در نتیجه، لازم است که براي ذخیره‌ي حالت کلي سیستم نیز چاره‌اي اندیشیده شود. براي این منظور، کلاسي تحت عنوان Subsystem State Manager را در نظر گرفتیم. در این کلاس نیز از الگوي State استفاده مي‌کنیم و این‌بار، وضعیت کلي زیرسیستم را در آن نگهداري مي‌کنیم. این کلاس مي‌تواند به واسطه‌ي Serverside proxy و یا Clientside proxy با عنصر Broker مرتبط شود. این امر به دو دلیل صورت مي‌گیرد:



شکل ۲.۱: نمودار ساختار کلاسی

۱. گاهي اوقات لازم است که براي تعيين وضعيت کل زيرسيستم، وضعيت کلاينت‌ها را دريافت کنيم؛ بنابراين لازم است که به کلاينت درخواست بفرستيم تا وضعيت خود را در اختيار ما قرار دهد. بنابراين، از پروکسي سمت کلاينت استفاده مي‌کنيم.

۲. گاهي اوقات لازم است که براي تغيير رويکرد عناصر کارچرخان، به درخواست عنصر کارچرخان مبني بر ارسال وضعيت کل زيرسيستم پاسخ دهيم. در نتيجه در اين حالت، اين کلاس در نقش سرور عمل مي‌کند و وضعيت خود را به صورت يک پاسخ به عنصر کارچرخان مي‌فرستد.

زیرسیستم‌هاي کارچرخان

این زیرسیستم‌ها نیز در قالب الگوي Broker پیاده‌سازي شده‌اند. با این تفاوت که عنصر Mediator به صورت یک جزء یکتا در این زیرسیستم‌ها وجود دارند. دریافت وضعیت زیرسیستم‌هاي مرتبط و نیز، رد و بدل کردن پیام در بین آن‌ها، از طریق این عنصر صورت مي‌گیرد. همچنین الگوریتم تعامل نیز در این عنصر قرار داشته و در صورت بروز شرایط مختلف، تغییر مي‌کند. براي این منظور نیز، از الگوي Strategy استفاده مي‌کنیم. استراتژي تعامل، بر مبناي وضعیت کل زیرسیستم‌هاي مرتبط تعیین مي‌شود.

عنصر کارچرخان نیز همانند عنصر مدیریت کننده‌ي زیرسیستم‌ها در بخش قبل، مي‌تواند هم در نقش کلاینت و هم در نقش سرور ظاهر شود. بنابراین، به هر دو نوع پراکسي متصل است. این عنصر، موارد زیر را مدیریت مي‌کند:

۱. تعامل بین دو جزء در دو زیرسیستم مختلف باید مدیریت شود.

۲. وضعیت کلي زیرسیستم‌ها، باید از عنصر مدیریت‌کننده‌ي زیرسیستم درخواست شود.

۲.۳.۱ سناریوها

عمده‌ي سناریوهاي این سیستم، همانند سناریوهاي الگوي Broker در حالت معمولي خواهد بود. تنها تفاوت اصلي، در وجود عناصري است که وضعیت سیستم‌ها را بازنمایي مي‌کند. این سناریوهاي خاص را بیشتر توضیح مي‌دهیم.

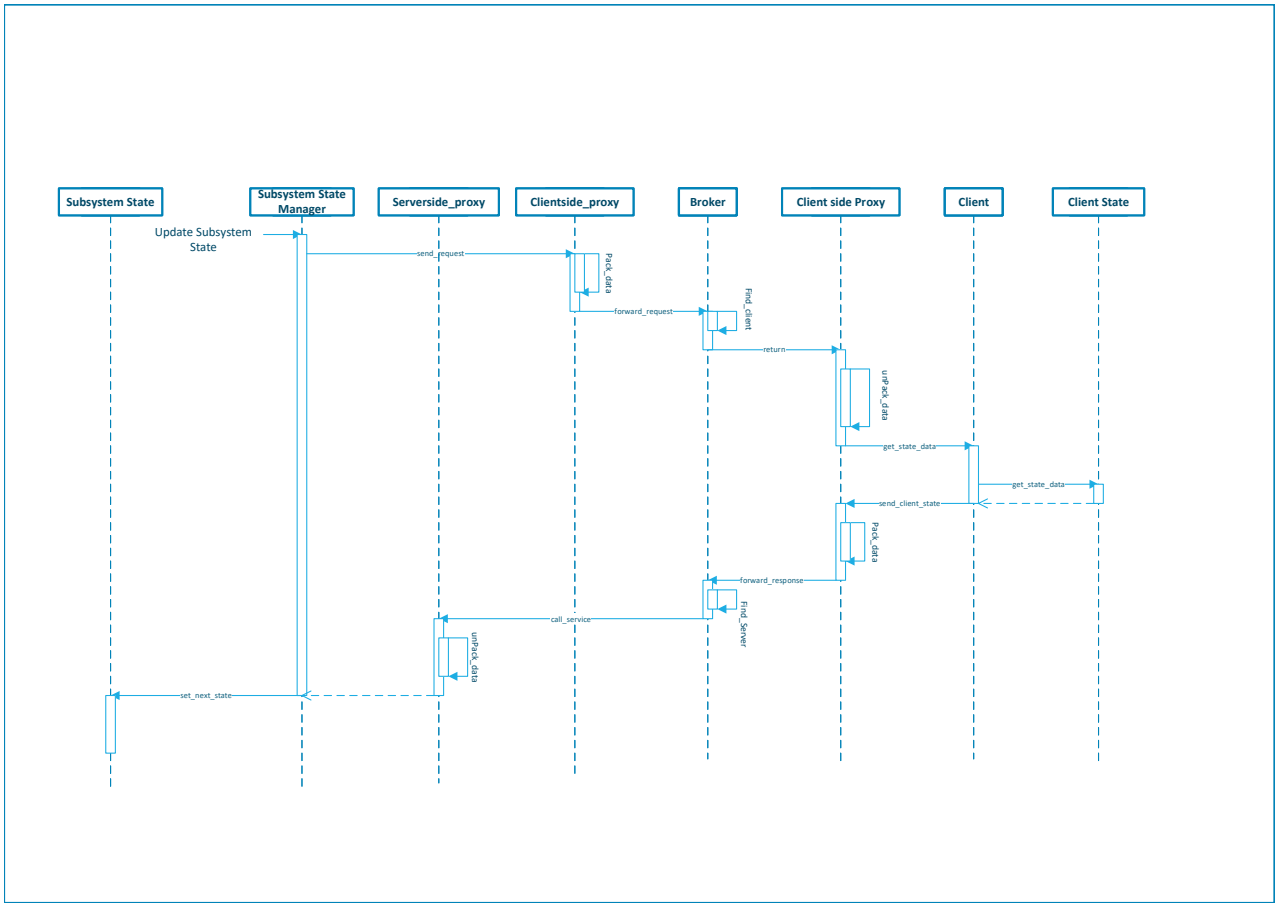
تعیین وضعیت کل زیر سیستم

وضعیت کلي هر زیرسیستم، در کلاس خاصي در زیرسیستم نگهداري مي‌شود که به آن اشاره کردیم. این سناریو در نمودار توالي شکل ۳.۱ مشخص شده‌است. این سناریو، بدین ترتیب است:

۱. رفتار بروز رساني وضعیت کلي زیرسیستم، فراخواني مي‌شود. بنابراین، لازم خواهد بود که وضعیت اجزاي زیر سیستم را بدست آوریم.

۲. با توجه به این‌که در نقش کلاینت هستیم، به پراکسي سمت کلاینت درخواست مي‌دهیم. این پراکسي، درخواست را بسته‌بندي مي‌کند و به بروکر تحویل مي‌دهد.

۳. بروکر، کلاینت مورد نظر را پیدا مي‌کند و بسته را به پراکسي آن مي‌دهد.



شکل ۳.۱: سناریوي تغییر وضعیت کل زیرسیستم‌ها

۴. با توجه به این که جزء هدف، یک کلاینت است، یک پراکسي سمت کلاینت این درخواست را تحویل مي‌گیرد.

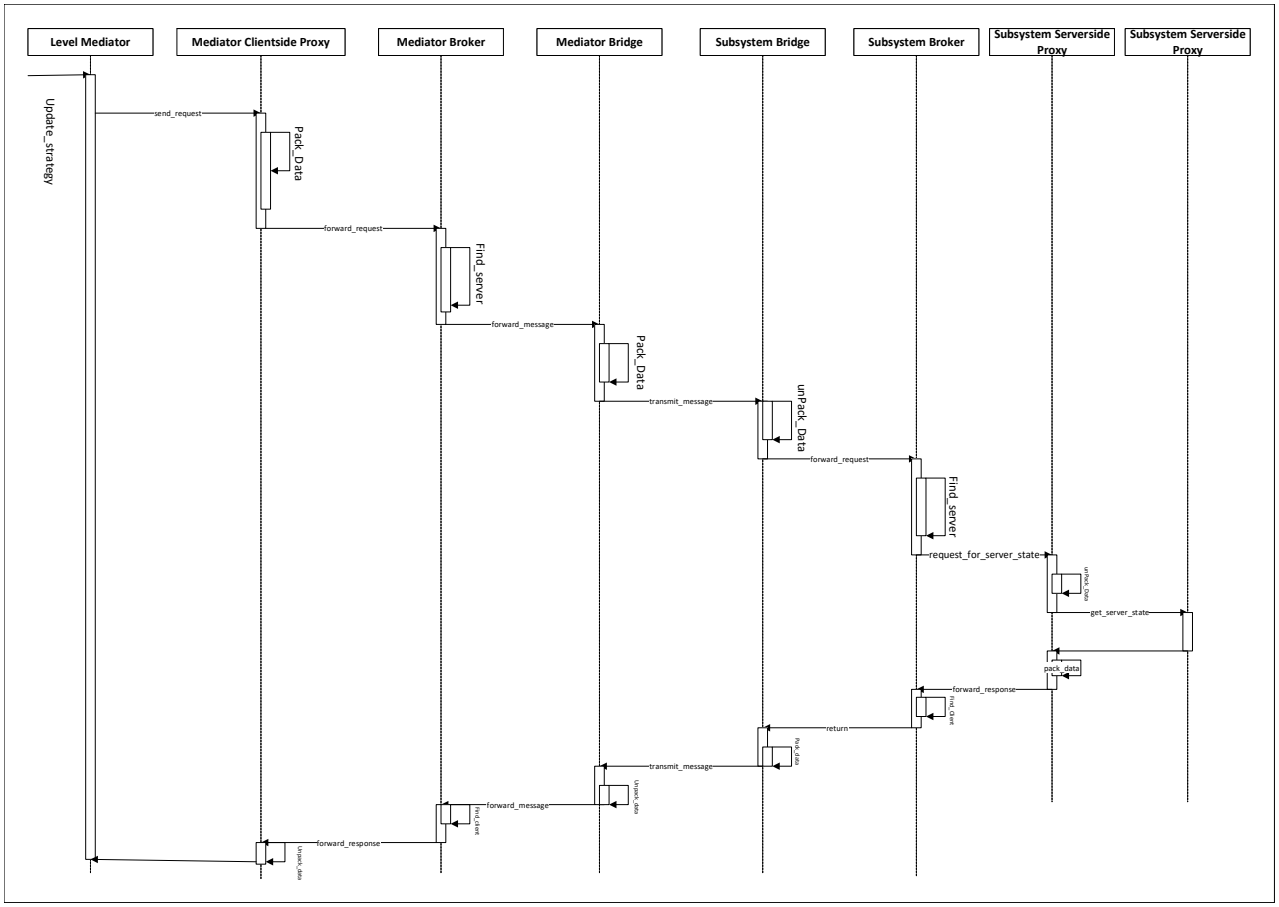
۵. کلاینت، وضعیت خود را در اختیار پراکسي قرار مي‌دهد و ارسال صورت مي‌گیرد.

۶. در نهایت، با توجه به این که کلاینت، پاسخ‌دهي ندارد (چون کلاینت صرفا درخواست مي‌فرستد و نه پاسخ) براي آن‌که احساس کند که به یک سرور درحال ارسال درخواست است، پراکسي سمت سرور و متصل به کلاس مدیریت‌کننده‌ي وضعیت کل زیر سیستم، آن را دریافت کرده و تحویل این کلاس مي‌دهد.

تعیین الگوریتم تعامل

همان‌طور که نشان دادیم، الگوریتم تعامل باید بر مبناي وضعیت تمام زیرسیستم‌ها تغییر کند. براي این منظور، لازم است که عنصر کارچرخان، وضعیت زیر سیستم را از عنصر مربوط به مدیریت وضعیت کلي زیرسیستم، دریافت کند. در چنین موردي، در نقش یک کلاینت به عنصر مدیریت‌کننده‌ي زیرسیستم

مورد نظر درخواست مي‌دهد و وضعيت آن را دريافت كرده و سپس بر مبناي آن وضعيت، استراتژي تعامل را تعيين كرده و يا تغيير مي‌دهد. اين سناريو در شكل ۴.۱ نشان داده شده است.



شكل ۴.۱: سناريوي تغيير استراتژي تعامل

۳.۳.۱ نقدهايي بر معماري

اين معماري، مزايا و معايبي را دارد كه در ادامه، به آن‌ها اشاره مي‌كنيم:

۱. در اين الگو، به علت وجود عنصر كارچرخان، وابستگي بين زيرسيستم‌ها بسيار كاهش يافته‌است و اين امر، كاپلينگ را بسيار كاهش مي‌دهد.

۲. با توجه به استفاده از الگوهاي State و Strategy مي‌توان تغييرات مورد نياز را در زمان اجرا انجام داد.

۳. يكي از مشكلاتي كه اين معماري دارد، وجود يک نقطه‌ي اتكا در عناصر كارچرخان است. در صورتي كه هر كدام از عناصر كارچرخان از كار بيفتند، تمام سلسله مراتب زير آن‌ها نيز از

کار مي‌افتد و ارتباط بين آن‌ها، قطع مي‌شود. بنابراين، تحمل خطا و خرابي در اين معماري کم است.

۴. به علت وجود پراکسي‌هاي متعدد در سطح سيستم، تبادل پيام و نيز تعداد اشياء بسيار زياد مي‌شود و اين امر، هم کارايي سيستم و هم حافظه را با مخاطره و محدوديت مواجه مي‌سازد.

۵. با توجه به وجود عنصر کارچرخان، احتمال به وجود آمدن کلاس‌هاي همه کاره و غير منسجم [۳] وجود دارد.

[3] non cohesive

فصل ۲

بررسي دو Idiom در جاوا

۱.۲ Favor static member classes over nonstatic

۱.۱.۲ مقصود

از اين راهكار، به منظور تعريف عناصري از جنس كلاس، در بدنه‌ي كلاس ديگر استفاده مي‌شود و هدف آن، محدود كردن دسترسي كلاس داخلي به اجزاي كلاس خارجي است.

۲.۱.۲ انگيزه

گاهي اوقات، لازم مي‌شود كه در بدنه يک كلاس بزرگ، كلاس‌هايي ديگر را به منظور مدل‌سازي اجزاي دروني آن، طراحي كرده و جاي دهيم. بنابراين، از كلاس‌هاي تودرتو استفاده مي‌كنيم. در بسياري از مواقع، تمايل نداريم كه كلاس داخلي، به محتواي كلاس خارجي اش دسترسي مستقيم داشته باشد. يكي از مثال‌هايي كه در جاوا مي‌توان زد، كلاس داخلي Entry در بدنه‌ي كلاسي مثل كلاس HashMap است.

۳.۱.۲ راهكار

تعريف كلاس داخلي به صورت استاتيک، باعث مي‌شود تا اجزاي كلاس داخلي، ارجاع پنهان به عناصر كلاس خارجي نداشته باشند. نمونه‌اي از عدم رعايت اين قاعده را در شكل ۱.۲ مي‌بينيم.

براي حل اين مشكل، كافيست از عبارت static قبل از تعريف كلاس استفاده كنيم.

```
// Typical use of a nonstatic member class
public class MySet<E> extends AbstractSet<E> {
    ... // Bulk of the class omitted

    @Override public Iterator<E> iterator() {
        return new MyIterator();
    }

    private class MyIterator implements Iterator<E> {
        ...
    }
}
```

شکل ۲.۱: نمونه‌اي از عدم رعايت قاعده‌ي اول

۲.۲ Limit source files to a single top-level class

۱.۲.۲ مقصود

اين راهکار به منظور جلوگيري از وجود چند تعريف براي يک کلاس تعريف شده‌است.

۲.۲.۲ انگيزه

در برنامه‌نويسي شي‌گرا، لازم است که هر کلاس، يک مفهوم مشخص و واحدي را داشته‌باشد.

۳.۲.۲ راهکار

راهکار ساده است! حتما بايد کلاس‌هاي متفاوت را در فايل‌هايي جداگانه تعريف کنيد و از تعريف چند کلاس، در يک فايل و يا در يک کلاس، خودداري کنيد. نمونه‌اي از عدم رعايت اين راهکار در شکل ۲.۲ آمده‌است.

براي جلوگيري از چنين مشکلاتي، حتما اين کلاس‌ها را در فايل‌هايي جداگانه تعريف کنيد.

```
// Static member classes instead of multiple top-level classes
public class Test {
    public static void main(String[] args) {
        System.out.println(Utensil.NAME + Dessert.NAME);
    }

    private static class Utensil {
        static final String NAME = "pan";
    }

    private static class Dessert {
        static final String NAME = "cake";
    }
}
```

شکل ۲.۲: نمونه‌اي از عدم رعايت قاعده‌ي دوم

کتاب‌نامه

[1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc., USA.

[2] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 1996. Pattern-Oriented Software Architecture - Volume 1: A System of Patterns. Wiley Publishing.

[3] Ramsin, R., 2021. Lecture4 in Patterns in Software Engineering course : GoF Design Patterns – Behavioral.

[4] Ramsin, R., 2021. Lecture5 in Patterns in Software Engineering course : GoF Design Patterns – Behavioral.

[5] Ramsin, R., 2021. Lecture 7 in Patterns in Software Engineering course : GoV Patterns – Architectural.

[6] Joshua Bloch. 2008. Effective java™, second edition (Second. ed.). Prentice Hall Press, USA.