باسمه تعالی

فاز چهارم پروژه درس تحلیل و طراحی سیستم‌ها

استاد: فاطمه قاسمی اصفهانی

دستیار آموزشی: محمدسجاد نقی‌زاده

اعضای گروه:

علی قنبری ۸۱۰۱۹۹۴۷۳

بهراد علمی ۸۱۰۱۹۹۵۵۷

محمدجواد بشارتی ۸۱۰۱۹۹۳۸۶

System Operations (درخواست بسته درمانی):

سناریو اصلی:

۱) انتخاب بسته

۲) پر کردن پیشنیاز های بسته

۳) پاسخ کارشناس بیمارستان/کلینیک/دکتر به مدارک ارسالی

۴) تخصیص پشتیبان

سناریو فرعی:

۱) تعویض کارشناس سلامت سهل‌انگار

۲) درخواست پوشش بیماری جدید توسط سامانه

توضیح طراحی در عملیات سیستمی:

*تصمیمات اصلیِ طراحی در سه عملیاتِ سیستمیِ **انتخاب بسته**، **پر کردن پیشنیازهای بسته** و **تخصیص پشتیبان** گرفته شده است. در طراحی دیگر system operation ها از یک یا ترکیبی از این تصمیمات استفاده شده است.

انتخاب بسته:

ابتدا از selectPackageHandler به عنوان facade controller برای handle کردن انتخاب بسته در سیستم استفاده کردیم. این controller به صورت نزدیک با package container عملیات انجام می‌دهد پس یک وابستگی میان آن‌ها وجود دارد (چه بخواهیم چه نخواهیم)، از طرفی packageContainer با توجه به اصل container در GRASP با خود package در ارتباط است.

همچنین هر package برای دسته ای از بیماری/بیماری‌ها تعریف شده است، پس یک وابستگی میان package و بیماری (sickness) ای که برای آن تعریف شده است در ارتباط است.
در قسمت عمده‌ای از طراحیِ این system operation سعی شده است تا با استفاده از packageContainer تعداد وابستگی‌ها را کم کرده که منجر به low coupling خواهد شد.

پر کردن پیشنیازهای بسته:
مشابه قبل، prerequisiteHandler به عنوان facade برای این system operation انتخاب شده است. مسئله‌ای که در طراحی این قسمت و قسمت‌های دیگر استفاده شده، کلاس‌های form و formSender هستند. در توضیح در نظر گرفتن این دو کلاس میبایست این نکته را مطرح کرد که قسمتی از ایده‌ی ایجاد آن‌ها از الگوی Chain of Responsibility در object oriented design patterns الهام گرفته شده است. به این صورت که با بعضی درخواست‌ها به صورت form رفتار می‌شود و در نتیجه‌ی ایجاد form، برای process کردنشان از کلاس formSender کمک گرفته می‌شود تا نتایج ایجاد و ثبت آن form را اعمال کند.

تخصیص پشتیبان:
یک تصمیم حائذ اهمیت در قسمت گذاشتن تایمر برای کارشناس سلامت گرفته شده است. با توجه به اصل Information Expert، تنها سیستم می‌تواند به ساعت سیستم (خودش) دسترسی پیدا کند. که System را به عنوان بهترین کلاس پیاده کننده این متود، خاطرنشان می‌کند.
یک تصمیم مهم و کلیدی در پیاده‌سازیِ assignHealthExpertToPackage گرفته شده است. به این دلیل که قرار است دو کلاس کلیدی و نقش‌آفرین کل پروژه (یا حتی کل bussiness) به یکدیگر assign شده و با یکدیگر ارتباط برقرار کنند. به همین منظور، انتخاب یک کلاس مناسب برای پیاده‌سازی تابع نامبرده، یک چالش اساسی در این قسمت بود. در نهایت پس از مدتی به سه کاندید اصلی برای انتخابِ کلاسی که این تابع را پیاده کند رسیدیم:

- package/HealthExpert: همانطور که توضیح داده شد، این دو کلاس، با یکدیگر در ارتباط بسیار نزدیک با قرار داشتند("B closely uses A" صفحه ۲۹۲ کتاب رفرنس)، پس هر کدام از

آن‌ها یکی از کاندیدهای اصلی برای در برگرفتن کلاس دیگر و در نتیجه امکان پیاده‌سازی تابعِ نامبرده را خواهند داشت.

- System: یکی دیگر از کاندیدای اصلی، خودِ سیستم بود. با توجه به اصل‌های Container و Information Expert، کلاس System هر دو کلاس package و healthExpert را درون خود می‌گنجاند اطلاعات لازم در مورد آن‌ها را خواهد داشت. پس استفاده از آن منجر به low coupling شده و می‌تواند به طراحی کمک کند.
- SystemRelations: و اما بهترین کلاسی که به نظر می‌رسد می‌تواند این کار را انجام داده و همزمان منجر به low coupling برای کل پروژه و از طرف دیگر منجر به high cohesion برای هر یک از کلاس‌های System, package,healthExpert خواهد شد کلاسی مانند SystemRelations است. دلیل بوجود آمدن high cohesion با بوجود آوردن این کلاس این است که هر کدام از کلاس های نامبرده به اندازه کافی بزرگ هستند و به اندازه کافی مسئولیت دارند که بارور کردن آن‌ها با یک مسئولیت دیگر می‌تواند منجر به overburden کردنشان شود. به عنوان مثال کلاس package یا به عبارتی بسته درمانی، باید چیزهای زیادی را درون خودش دنبال کند مثل روند درمانی که در آن استفاده می‌شود و پیشروی پروسه بر اساسِ آن و همچنین افراد یا ذی‌نفعانی که به آن بسته ممکن است assign شده باشند/بشوند و پاسخ‌دهی مناسب به درخواستِ هر کدام از این ذی‌نفعان و مسائل دیگری که ممکن است نمود پیدا نکنند. پس یک کلاس اضافه می‌خواهیم قسمتی از بار این مسئولیت را برای هر یک از این کلاس‌ها به دوش بکشد. کلاسی که میخواهیم باید در عین حال به اندازه کافی بزرگ مانند سیستم باشد و از طرفی به اندازه کافی (و نه بیشتر) دارای اطلاعات لازم برای انجام و پیاده‌سازی این تابع باشد و کلاس SystemRelations هر دوی این نیازمندی‌ها را پشتیبانی می‌کند. نکته‌ای قابل توجه که میبایست برای این کلاس در نظر گرفت این است که در ادامه پروژه ممکن است با کنار هم گذاشتن تمام پروژه (نه لزوما در حدی که در صورت پروژه آورده شده است و با جوانب و جزئیاتی بیشتر) کلاس SystemRelations خود، دچارِ مسئولیت‌های زیادی شده و منجر به low cohesion شود. در این صورت می‌توان برای هر کدام از کلاس‌ها

یک کلاس اضافی External Relations در نظر گرفت (تا حدی مشابهِ الگوی Adapter) تا مسائل و ارتباطات خارجی هر کلاس را در صورت نیاز handle کند.

در طراحی کلی کلاس‌ها، کلاسی تحت عنوان inputEventHandler به عنوان interface یا لایه‌ای به منظور invoke کردن facade controller های مورد در نیاز در event های مختلف برای System در نظر گرفته شده است تا باری را از دوش System برداشته و منجر به cohesion بیشتر شود.