الهه خداوردی - 810100132 فرشته باقری - 810100089 سیده زینب پیش‌بین - 810199597

تکلیف هشت

گام 1 و 2:

شناسه کامیت حاوی بوی بد	شرط مختصر بوی بد	شناسه کامیت باآرایی شده	توضیحات در صورت نیاز
7b3c49dfcfd2d6bde32bf2026b5bbcf2d40cb118	Long Method	d98a682099f42977e36b9d28cf46f55c03fc952e	newOrder کردن تابع extract و ایجاد دو تابع جدید برای بررسی enough position برای سفارش های فروش
d98a682099f42977e36b9d28cf46f55c03fc952e	Duplicate Code	eb3c72d80a9406f502625ad61028f5a2a9b89057	delegate constructor of Order class
eb3c72d80a9406f502625ad61028f5a2a9b89057	Long Method	0c3586fab598317a9ab4e2107a6f3b62a4586298	شکستن تابع newOrder و کردن تابع extract createOrderFromRequest که وظیفه ساخت سفارش بر اساس نوع آن را دارد
0c3586fab598317a9ab4e2107a6f3b62a4586298	Switch Statements	1cf4acf466462c82700fbe00740a32ebd4fdb95d	از آنجایی که سفارش های عادی و توقفی در صف های متفاوتی نگهداری می شوند، هنگام جست و جو با توجه به نوع سفارش در صف متفاوتی به دنبال آن می گردیم. در این قسمت هنگام حذف سفارش برای جست‌وجوی سفارش در صف مربوطه تابعی جدا تعریف کردیم.
1cf4acf466462c82700fbe00740a32ebd4fdb95d	Long Method	97f3d74842fa50de2fd543659a7fddd2fb94b0d5	نوشتن یک تابع جدید برای validation در هنگام حذف سفارش ها در security
9b40df2592239d72054a9f	Long Method	9b40df2592239d72054a	کردن تابع extract createTrade از match

	9fc4edf747d2ad59882b		c4edf747d2ad59882b
تابعی که در کامیت قبلی (سطر بالایی) ساخته بودیم را از کلاس تطابق حراج به کلاس انتقال می (matcher)پدر دهیم، چرا که تطابق پیوسته نیز از آن استفاده می کند.	67c5aa4d50bfabce653b8918379a27a7632df1ca	Feature Envy	f85b80d313a912a7e2494e97d71c20b4ce24764c
مراحل بروزرسانی (و در صورت نیاز حذف) سفارش خرید و فروشی که در یک ترید با هم جفت شده اند، به صورت یک تابع جدا نوشتیم.	7d6518edb4219c7a0f992424b7cd4eb3582e096e	Long Method / Switch Statements	9b40df2592239d72054a9fc4edf747d2ad59882b
در کلاسی که در کامیت قبلی تعریف کردیم، تمام ترید را دریافت می کردیم که نیازی نبود، پس در این کامیت آن را اصلاح کردیم و تنها کوانتیتی آن را به تابع می دهیم.	f85b80d313a912a7e2494e97d71c20b4ce24764c	Unnecessary Data Exposure	7d6518edb4219c7a0f992424b7cd4eb3582e096e
در حالت تطابق پیوسته، پس از انجام ترید نیاز است میزان کوانتیتی سفارش اصلاح شود. این مرحله با توجه به اینکه مقداری از کوانتیتی سفارش باقی مانده یا نه حالت بندی می شود. در این کامیت بررسی گفته شده را به صورت یک تابع شکل دادیم.	298c22e823db37b98b3f60f3b67483d4c429fded	Long Method	67c5aa4d50bfabce653b8918379a27a7632df1ca
کردن درهر دو حالت execute تطابق پیوسته و حراج استفاده میشد. این کد تکراری را از دو سمت حذف کردیم و در تطابق اصلی (که دو حالت حراج و پیوسته اکستند شده آن	3203e00b2c510ef41c5b38628244cb3b8b89abf6	Duplicated Code / Template method pattern	298c22e823db37b98b3f60f3b67483d4c429fded

هستند.) قرار دادیم			
برای استفاده ReqHandler تابع در برخی از متدها طراحی شده.	8fd09668a3268ab32b0a7be53f161f529c9f4180	Refused Bequest	a98cfb47456a301aecaa8c5cac172cbdb33d4e4e
هنگامی که یک درخواست بروزرسانی داریم، بررسی می شود که این درخواست صحت دارد یا خیر، و اگر اشتباهی در آن باشد، ارور متناسب ارسال می شود. قسمت بررسی درخواست از تابع بروزرسانی سفارش خارج شده و در یک تابع دیگر تعریف شده.	dd57a7a767e8d2fee156b0907b30bfb0dcdbcf4c	Long Method	8fd09668a3268ab32b0a7be53f161f529c9f4180
در دو قسمت کد، دو شرط چک کردن بسیار مشابه داریم که پیش از این یکی از آنها به صورت تابع شکل گرفته. تابع را مقداری تغییر می دهیم تا برای قسمت مشابه دیگر نیز کاربردی باشد. این قسمت چک کردن پوزیشن ها در آپدیت سفارش است.	d900daf38b7f66a3b447120546b04abe55b9f55f	Duplicated Code	dd57a7a767e8d2fee156b0907b30bfb0dcdbcf4c
برای تشخیص آنکه سفارش پس از بروزرسانی اولویت خود را از دست داده است یا نه، شرطی بسیار طولانی داریم که می تواند برای خواننده گیج کننده باشد. در نتیجه آن را به صورت یک تابع استخراج می کنیم.	751cb190ed41d39572ac2adaa49cdbe6e272a853	Long Method	d900daf38b7f66a3b447120546b04abe55b9f55f
تابع یافتن قیمت بازگشایی را به کلاس سفارش ها (orderBook) انتقال دادیم.	4f1f32e9627dc95223d57ee2c4567901ce8085dd	**Inappropriate intimacy** / Feature Envy	0bf8dff60c462b7c5b74232bd61cbadd13331f92
تابع یافتن قیمت بازگشایی بسیار طولانی شده و تمام	0446cc887cc8c7805307ca2dec8db57c9bebd648	Long Method	4f1f32e9627dc95223d57ee2c4567901ce8085dd

			عملیات ها و محاسبات در همین تابع انجام میشود که باعث ناخوانایی آن شده. در نتیجه قسمت هایی از این تابع به صورت توابعی دیگر استخراج شدند.
eb5195adca21209c8cd3404c12faef50bdc98b73	Data Clumps	7e751df9fd5e5977d06ba476646c1a8bfd20fb67	در این کامیت یکی از کارهایی که انجام شد انتقال فیلدهای مشترک درخواست ها به superclass بود.(توجه کنید که کامیت نهایی پس از تصحیح چند مورد جزئی پس از ایجاد سوپرکلاس اصلی قرار داده شده است و فاصله زیادی نسبت به کامیت حاوی بوی بد دارد.)
07993af8a8f1e84260772f201fdffec9310b0ff3	Refused Bequest / Duplicate code /(Visitor Pattern)	کامیت ابتدا: eb5195adca21209c8cd3404c12faef50bdc98b73 کامیت انتهایی: 2bd0732491e4ce694a3acb7f4190fba843b4e6ab	ها از requestHanlder برای استفاده visitor pattern کردیم.
2bd0732491e4ce694a3acb7f4190fba843b4e6ab	Long Method/ Large Class/ Single Responsibility	f0b2a842a32bca2a03d24cadf3a33eb800112ea3	validation کدهای مربوط به جدا hanlder های ریکوئست از شده و به یک کلاس دیگر منتقل شد (RequestControl)
f0b2a842a32bca2a03d24cadf3a33eb800112ea3	Duplicate Code	7c5beb94c50e3b470809c447fec5de27dd2513d5	کدهای تکراری مربوط به مچ شدن کامل سفارش در حالت حراج به صورت method جداگانه در نظر گرفته شد تا duplicate رفع شود.
0bcbdaab2a25f33d6a174bf623a85e1d01e0506d	Long Method	88a0c28d664d5ad13fff0d84fe9f1d65d560e861	کردن validate تابع مربوط به در EnterOrderReq

طولانی بود و orderHandler هایی method این را به جداگانه تبدیل کردیم که هر یک validate بخش مشخصی را میکنند.			
eventPublish های جدا کردن از قسمت EnterOrderReq کردن ریکوئست و ایجاد handle جداگانه method	0bcbdaab2a25f33d6a174bf623a85e1d01e0506d	Long Method	7b262dd23076e4aac6eed923ba92cbf3619bd444
جداگانه برای method ایجاد position مچ از پس ها. آپدیت updatePositionFromTrades	c5d5b8f49a88c78a20aab03c347cfb7de58efa02	Long Method / Single Responsibility	aad39ccdfda550f6336354bd683188e7c0d5d475
جدا برای method یک extract حالتی که سفارش کامل مچ شده از صف یک سفارش آیسبرگ است. handleIcebergOrder	ac9e4f4c24591c473329a6d28f915df10d246abc	Duplicate Code	f435aec1b6144eaefb953e6beefcfaa2590b16b4
کردن کد اضافه و refactor حذف کردن سفارش ها براساس نوعشان از صف های مربوط به هرکدام. برای سفارش های توقفی و سفارش های عادی.	28060913d5c545d0a767ce1ac5b31eef150dce8c	Duplicate Code	b0c85b86243e607d77ff0fd0dbe1e9bd58ebd481
ها و فیلدهایی که در import تست ها وجود داشت و استفاده نمیشد حذف شد.	659babc66442ecf7b819e1c0b3d170d0df2a1ea7	dead code	7e751df9fd5e5977d06ba476646c1a8bfd20fb67
حذف کامنت ها	a018821b539f747c7593caa97746d4e2e4d49733	comments	e149d07e81062bfd09b575f4b8e65caab53919c6
برای factory معرفی کلاس های ساخت انواع سفارش ها و حذف برای security این توابع از پایبندی به single responsibility principle.	136462f5c08c17f954e1bf3eadaa29e412a28d28	inappropriate intimacy/ coupling	a018821b539f747c7593caa97746d4e2e4d49733

			ایجاد سفارش های جدید از وظایف مربوط به سکیوریتی نمی باشد و بهتر است کلاس های برای آنها داشته centralized باشیم.

سایر bad smell ها:

برخی از bad smell ها که به دلیل ماهیت پروژه، نوع طراحی یا کمبود وقت برای تغییر تمام فایل ها و تست ها رفع نشده اند، آورده شده.

1. یک مشکل درباره کد هندل شدن eventPublish ها در هندلرهای درخواست‌ها بود. برای حل این مشکل میتوان یک کلاس EventPublisherService اضافه کرد و تمام قطعه کدهای مربوط به آن‌را به این کلاس منتقل کرد تا مشکل single responsibility و large class حل شود ولی با توجه به نحوه verify کردن در تست‌ها و بقیه ساختار با پیچیدگی‌هایی در ادامه این تغییر مواجه شدیم که از تغییر این مورد منصرف شدیم.
2. Primitive Obsession: به جای استفاده از تایپ primitive برای credit، می توان آن را یک تایپ مانند Money در نظر گرفت و عملیات های مرتبط با آن را در کلاس خود پیاده سازی کرد.
همچنین برای Id ها هم می توان همچین کاری کرد و عملیات هایی مانند validate کردن منفی نبودن آیدی یا... را در کلاس خودش انجام داد. این منطق برای سایر ویژگی های سفارش مانند quantity نیز صادق است و اگر فیلدی داریم که operation های مخصوص خود را دارد، بهتر است به جای primitive تایپ، یک کلاس جداگانه برای آن داشته باشیم و عملیات های مختص آن را در کلاس خود انجام دهیم.
3. Long Parameter List: در کلاس های مربوط به انواع order ها و همچنین توابع، constructor ها تعداد زیادی پارامتر را برای ثبت ویژگی های سفارش دریافت می کنند. از راه حل های رفع این bad smell ایجاد کلاس های جدید برای پارامتر هایی است که اکثرا با هم استفاده می شوند که همان ویژگی های اردر ها است. اگر کلاس جدید هم می زدیم دوباره کانستوراکتور آن تعدادی زیادی پارامتر می داشت. توابعی مانند createNewOrderRq و ... نیز همین مشکل را دارند.
یکی از ایده های دیگر، داشتن یک کلاس entity جداگانه برای price و quantity است چون این دو با هم در request ، trade و سفارش ها استفاده می شوند. به خاطر اینکه این دو در کنار هم معنی مستقلی ندارند (جزوی از ویژگی های سفارش یا ترید هستند) و همچنین کمبود زمان این تغییر انجام نشد.
4. Date clumps: مشکل ذکر شده در مورد 2، data clumps bad smell هم هست چون این ویژگی های سفارش اکثرا با هم دیگر به توابع پاس داده می شوند.

5. Large Class: به دلیل ماهیت tinyMe و تعدد عملیات ها و feature ها برخی از کلاس ها در پروژه بزرگ هستند. برخی از این کلاس ها به کلاس های کوچکتری تقسیم شدند و وظایف مرتبط به هر کدام جدا شد.
6. Large method: توابع طولانی تا حد ممکن به توابع کوچکتر شکسته شده اند ولی extract کردن بخش های کوچکتر از برخی توابع مانند find opening price به دلیل نحوه کارکرد و الگوریتمی بودنشان ممکن نیست.

گام 3:

1. در دو حالت auction و continuous ما می‌توانستیم از دو design pattern مربوط به strategy و template method استفاده کنیم. با توجه به اینکه تفاوت اصلی این دو matcher در هنگام match کردن بود ما از design pattern مربوط template method در method execute استفاده کردیم. منطق execute برای هر دو یکسان است سفارش ابتدا مچ می‌شود و سپس به صف اضافه می‌شود. تفاوت تنها در این است که در هنگام حراج ما سفارشی را مچ نمی‌کنیم.
و البته در نتیجه ما یک طراحی که شامل هر دو strategy pattern و template method pattern است داریم (توضیحات مربوط به strategy در مورد بعدی).

شناسه کامیت
3203e00b2c510ef41c5b38628244cb3b8b89abf6

2. در طراحی اولیه که برای هندل کردن auctionMatcher و continuousMatcher داشتیم از strategy pattern استفاده کردیم. همانطور که مشخص است همانند strategy دو کلاس برای هر حالت مستقل از هم وجود دارند که هر دو یک matcher هستند. سپس در هر دوی این کلاس method ها براساس شرایط و محدودیت‌ها overrride شده اند و ما با توجه به نیاز و شرایط کافی‌است matcher مورد نظر را استفاده کنیم (در methodهای ایجاد و آپدیت سفارش در کلاس security).
با توجه به اینکه پیدا کردن شناسه مربوط به پیاده‌سازی و اضافه کردن auctionMatcher به این صورت سخت بود شناسه کامیت مورد نظر برای این پیاده‌سازی به خصوص نیست ولی کامیتی است که حاوی نتیجه کد مورد نظر است.

شناسه کامیت
298c22e823db37b98b3f60f3b67483d4c429fded

3. با توجه به کلاس security که دو state برای matching دارد شاید به ذهن خطور کند که می‌توانیم از state pattern استفاده کنیم. ولی نمی‌توانیم این کار را کنیم؛ زیرا در کد ما اگر میخواستیم این کار را انجام دهیم می‌بایست در ابتدا سه کلاس جدید تعریف می‌شد : MatchingState, ContinuousMatchingState, AuctionMatchingState که دو کلاس دیگر از MatchingState ارث‌بری می‌کردند سپس برای اینکه هر کدام از کلاس‌ها بتوانند به درستی براساس pattern مربوطه عملیات match را انجام دهند بایستی از سرویس matcher های مربوطه استفاده می‌کردند. به نظر ما این کار در کد درست نبود زیرا ما entity هایی بوجود می‌آوردیم که نیازمند service های ما در خود بودند این درحالی بود که تنها کار این entity ها همین بود و تمام بار در خود های match انجام می‌شد و با طراحی فعلی مناسب بود.
4. برای هندل کردن request های مختلف در ابتدا تنها یک orderHanlder داشتیم و سپس ChangeMatchingStateHandler اضافه شد. برای خوانایی بهتر و طراحی بهتر هر دو از یک کلاس ارث بری می‌کردند زیرا در نهایت هر دو شامل بخش‌های یکسانی بودند و تفاوت اصلی در ریکوئست ها در نوع آن‌ها بود که باید متفاوت هندل می‌شدند. لذا از visitor pattern استفاده کردیم.
نتیجه آن نیز در نهایت (با refactor ها و ...) می‌شود کامیت زیر:

شناسه کامیت
2bd0732491e4ce694a3acb7f4190fba843b4e6ab

5. همچنین در قسمت‌های مربوط به test برنامه می‌توانستیم از کلاس Director برای builder ها استفاده کنیم. اما به دلیل اینکه کنترل builderها با توجه به تعدد تست‌ها به این صورت آسان‌تر و قابل فهم‌تر بود این کار را انجام ندادیم.

6. برای ایجاد سفارش‌ها با انواع مختلف برای درخواست سفارش جدید، logic مربوط به ساخت سفارش‌هارا جدا کرده و یک کلاس factory مربوط به ایجاد سفارش‌ها اضافه کردیم(Factory Method Pattern). با در نظر گرفتن dependency injection سه کلاس فکتوری مربوط به سفارش‌های مختلف ایجاد شد.

شناسه کامیت
136462f5c08c17f954e1bf3eadaa29e412a28d28

7. همچنین در مورد قبلی از آنجایی که ما فقط به یک instance از فکتوری‌های مختلف نیاز داریم از Singleton Pattern استفاده کردیم تا فقط یک instance از فکتوری‌ها داشته باشیم که در کامیت زیر(همان کامیت مورد قبل) قابل مشاهده است.

الهه خداوردی - 810100132 فرشته باقری - 810100089 سیده زینب پیش‌بین - 810199597

شناسه کامیت
136462f5c08c17f954e1bf3eadaa29e412a28d28