تحلیل و طراحی سیستم ها

گزارش مربوط به تمرین کامپیوتری 7

اعضای گروه:

بردیا خلفی 810199414

آرش شاهین 810199442

محمد متاعی 810199493

گام 3.

دو موقعیت برای استفاده از design pattern ها داریم:

1. استفاده از Chain of Responsibility در validation: در ابتدا کد های مربوط به validation را به یک کلاس انتقال دادیم و آن را بر اساس بوی بد طولانی بودن متد و کد تکراری, refactor کردیم.
   و بعد برای هر validation خاص, یک کلاس ساخته و به این ترتیب با استفاده از این pattern, توانستیم کیفیت کد را بهتر کنیم.

2. استفاده از dependency inversion در matcher: در کلاس matcher و مخصوصا متد execute, با استفاده از تعریف یک interface به نام MatchingController, منطق ها و عملیات های مربوط به هر بخش ,credit ownership, MEQ, stopLimit را در کلاس فرزند مربوطه پیاده سازی کرده, و توانستیم وابستگی عملیات ها را از بین برده و وظایف را جدا کنیم.

گام 4.

شناسه کامیت حاوی بوی بد	شرح مختصر بوی بد	شناسه کامیت بازآرایی شده	توضیحات (در صورت نیاز)
e87192f90901421050cdc756d94cdc2d1492e789	طولانی بودن متد Update در Security	087393be36fa5a91f80d1e706c615b78fcf2ff9a	با وجود جابجایی بخش validation , باز هم متد طولانی بود
087393be36fa5a91f80d1e706c615b78fcf2ff9a	طولانی بودن و چند کار کردن متد handleEnterOrder در OrderHandler	c910ee79915387149c9e6bda3dad14bdc59d2c65	این متد وظایف زیادی از جمله publish کردن event های مختلف, validate کردن سفارش را انجام میداد که نیاز به refactor داشت
24d8187b567d132562ae8b306383081adc977557	طولانی بود متد matcher و داشتن کد تکراری	488089c7c8832fc7fe19dc22ed6e2b3cd9cbb937	به دلیل داشتن دو نوع matcher یکی continuous و auction کد تکراری داشتیم که با بردن متد های مشترک به abstract class این مشکل حل شد

بخش های مختلفی از کد به refactor های کوچک نیاز داشتند کامیت های مربوط به آنها, کلمه refactor دارد. در جدول بالا, بخش های مهم و با بیشترین بوی بد نوشته شده.