به نام خدا

پروژه 8 مهندسی نرم افزار

TinyME Refactoring

سید احمد رکنی حسینی 810100154 - نیما تاجیک 810100104 - امیرحسین راحتی 810100144

آدرس مخزن : https://github.com/nimaCod/SE01-A06-A07

	شرح مختصر بوی بد	شناسه کامیت حاوی بوی بد	شناسه کامیت بازآرایی‌شده	توضیحات (در صورت نیاز)
1	Long Class: Matcher is too long	ddf1d56eeb0a2e42d2f1f4b850d17d193bb6bf94	C5f74dc8427451d5beca71dd106c45d4a8ff661a,	
2	Long Method: auctionExecuteOrderAndCheck	ddf1d56eeb0a2e42d2f1f4b850d17d193bb6bf94		
3	Hide Delegate: Order Order.getSecurity() is just used for getting OrderBook of security, not more!	ddf1d56eeb0a2e42d2f1f4b850d17d193bb6bf94		We can use Hide Delegate refactoring method to make a method for Order which returns Orderbook directly
4	?: Matcher: makeTrade() It adds trades to a parametrized list of Trades instead of returning a list of trades	c321ee381dbb0dc52e90bf7876eecaf8816c1e2d		Because of calling rollback function in makeTrade and it’s dependency on trades it can’t be done
5	Insensice return value: Matcher: makeTrade() If makeTrade run successfully it returns null which makes no sense	c321ee381dbb0dc52e90bf7876eecaf8816c1e2d	5932fdedc5dddf75e9459e28c1910775dae36bb3	

	64b4f10d450923fe11267eb50100026d3bd314cd	c321ee381dbb0dc52e90bf7876eecaf8816c1e2d	Inappropriate Naming: Matcher: checkQuantityAfterMatch It doesn't do the checking, it's changing the quantities	6
		c321ee381dbb0dc52e90bf7876eecaf8816c1e2d	Multi-purpose method : Matcher: checkQuantityAfterMatch Alongside changing quantity it works on Orderbook and pushes some items to it	7
	7b16bdb6547dacc8906876b30f5ad2e664758375	ddf1d56eeb0a2e42d2f1f4b850d17d193bb6bf94	Matcher: CheckMinimumExecutionQuantity This function and its below codes can merged (Lines 48 - 53) inside a function	8
	c5f74dc8427451d5beca71dd106c45d4a8ff661a	ddf1d56eeb0a2e42d2f1f4b850d17d193bb6bf94	Long Method : Matcher:calculateSecurityAuctionPriceAndQuantity	9
	c5f74dc8427451d5beca71dd106c45d4a8ff661a		Inappropriate Placing: "calculateSecurityAuctionPriceAndQuantity", and "calculateAuctionTradedQuantity", are relative to Security but placed In the Matcher file	10
	2e0c7cea35fc389c5f49e316a0e662b824ee9116	3ce06cc4ca36c8c2b1e39b22de2c39eb4de03a1d	Incomplete function: Matcher: auctionExecuteOrderAndCheck	11
			Unused function: Matcher: applySellRemainderOnSecurity	12
			Duplicate code : Matcher: auctionMatch and match the same line of codes	13
			?: Matcher: execute (Line 96) Execute function that just calls auctionExecute?	14

	86ad21c7ae9316cda403952e014ed2f7907b493f	Fixed in 27	Large methods in security.java	15
Created a new class: EntityObj for pass needed entities to methods	d523b2e3a5b743a1478a80faef948ccfa90030c3	17e8ac39d83a95368d2c8d2e51a8807dbbec9af0	Broker and shareholder are passed to many methods together	16
~newOrder is long and if-else must change	34705030d236049de3fc63bca858ac05854f2abd	17e8ac39d83a95368d2c8d2e51a8807dbbec9af0	Duplicate codes in newOrder and updateOrder in security	17
	e5fce65d705d2abbe03b93d5c6b5f3a34a99bc51	17e8ac39d83a95368d2c8d2e51a8807dbbec9af0	large methods in order handler	18
	089bae18d612ab48fd05a1d56100ffac1765cedb	17e8ac39d83a95368d2c8d2e51a8807dbbec9af0	Chain of method call, when want to check match Result trades, is empty or not(line 63)	19
Primitive obsession	gave up	17e8ac39d83a95368d2c8d2e51a8807dbbec9af0	Can implement new types such as Price and quantity for types of variables such as Stop Limit Prices.	20
Single Responsibility Principle	dc23d94b5d2e16417a0098801e735cade70dd17d	17e8ac39d83a95368d2c8d2e51a8807dbbec9af0	The OrderHandler class has many responsibilities such as messaging , checking and handling orders.	21
Single Responsibility Principle	afc94d2350688306129dfdd4530243e2f19095f9	6afda7cab37c22f37faffe62f3440c047d03013f	We can move reqToOrderMap in orderHandler to a new class	22
	e5fce65d705d2abbe03b93d5c6b5f3a34a99bc51	17e8ac39d83a95368d2c8d2e51a8807dbbec9af0	Some method names can be changed such as : matchRequest	23
we can create a new class named Repository and move all of this repositories	d523b2e3a5b743a1478a80faef948ccfa90030c3	Available in unchanged source Code.	There are many uses of Repositories together in many functions .	24

to this class				
	97c949291d2159163560ec72ead27fdece93f926	Available in unchanged source Code.	updateOrderIfPossibleThenMatch must do replace temp with query for loses priority	25
	86ad21c7ae9316cda403952e014ed2f7907b493f	c5f74dc8427451d5beca71dd106c45d4a8ff661a	Duplicate code in getAllUniquePrices and calculateAuctionTradedQuantity	26
			needToupdateAuctionPriceAndQuantity does not have a good name	27
Group members did not allowed this change 🙂			In OrderHandler class, matcher is always passed to security methods. we can move matcher to security	28
	9302310c4504e18585b4d3f4bec304c8748bcb91	Available in unchanged source Code.	Class TradeEvent needs many arguments . we can pass trade object and get fields from it	29
	2a2ba1e11f996c7e0b285982e4c74e8a4ccc0b61	17e8ac39d83a95368d2c8d2e51a8807dbbec9af0	methodCall chain while try to get awoke stopLimit list	30
	6cf38e285b0cbc891cb9a81048ec47f2991c8fbb	17e8ac39d83a95368d2c8d2e51a8807dbbec9af0	Some method names has typing error in security (`enqueOrderIfNotExcuted`)	31

تغییرات طراحی

1. مکانیسم MatchResult

در واقع در این مکانیسم اتفاقی که تاکنون افتاده است این است که **matcher** پس از انجام **Trade** ها یک **value** به عنوان این مقدار بر می گرداند. اما کاربرد آن چه بوده ؟

```
public void handleEnterOrder(EnterOrderRq enterOrderRq) {
    reqToOrderMap.put(enterOrderRq.getOrderId() , enterOrderRq.getRequestId());
    try {
        validateEnterOrderRq(enterOrderRq);
        Security security = securityRepository.findSecurityByIsin(enterOrderRq.getSecurityIsin());
        Broker broker = brokerRepository.findBrokerById(enterOrderRq.getBrokerId());
        Shareholder shareholder = shareholderRepository.findShareholderById(enterOrderRq.getShareholderId()
        MatchResult matchResult;

        matchResult = matchRequest(enterOrderRq, security, broker, shareholder);
        if(matchResult.outcome() == MatchingOutcome.INACTIVE_STOP_LIMIT_ORDER){
            eventPublisher.publish(new OrderAcceptedEvent(enterOrderRq.getRequestId(), enterOrderRq.getOrde
            return;
        }
        checkMatchResult(matchResult,enterOrderRq,isStopLimit:false);
        activateStopLimitOrders(security, matchResult);

    } catch (InvalidRequestException ex) {
        eventPublisher.publish(new OrderRejectedEvent(enterOrderRq.getRequestId(), enterOrderRq.getOrderId(
    }
}
```

همانطور که مشخص است خروجی آن در یک متغیر ریخته شده که بر اساس نتایج آن در تابع CheckMatchResult خروجی ها چاپ شود.

اما خیلی از این موارد ارورهای ما هستند و ما داریم از سه لایه این return Value را propagate می کنیم تا به اینجا برسانیم و آن ها را نشان دهیم.

راه حل: از مکانیسم مربوط به ارور هندلینگ یعنی throw استفاده کنیم تا مشخص باشد که ارور در حال ارسال شدن است . از طرف دیگر propagate کردن آن لازم نیست از طریق return Value انجام شود و خود مستقیم انجام می گیرد.

برای رفع این مشکل ، یک کلاس جدید تحت عنوان MatchingException اضافه می کنیم که وظیفه ایجاد exception ها و ایجاد ارور ها را دارد .

به این دلیل که لاگ های خروجی Matching Result برای ولیدیتی تست ها به کار می روند (هم در تست های ما و هم در تست های تی ای های محترم) از اضافه کردن این تغییر به کل پروژه منصرف شدیم و صرفا در یک branch دیگر میتوان به آن دسترسی داشت :

https://github.com/nimaCod/SE01-A06-A07/tree/RemoveMatchResult

2. مکانیزم validate کردن عملیات ها

چندین بخش مختلف در پروژه وجود دارد که حالت های مختلف ورود اردر جدید و ترید ها و ... را بررسی میکند و این کار در لایه های مختلف انجام میشود. میتوان این عملیات ها را در قالب یک یا چند کلاس validator که نقش یک جعبه ابزار (نه یک entity) برای تایید کردن حالت های مختلف عملیات ها را دارد انجام داد.

برای این کار ، چندین کلاس جدید در پکیج validator تعریف کردیم که عملیات های checking و validation در این بخش انجام میشود . شناسه کامیت هایی که در آن کلاس های validator اضافه شد:

332aa6d6af7264cf6b067f507a82d282f0d676f2	1
089bae18d612ab48fd05a1d56100ffac1765cedb	2
a5958a500417684fcb8e141d8efd99794d1ee79a	3
dc23d94b5d2e16417a0098801e735cade70dd17d	4
5747f8e10750873805fff75609f76a78e559cf52	5
ce697a329b3776fc08a769619c0b9de57ea118f0	6

3.مکانیزم مدیریت credit ها

به دلیل checking های فراوان برای بررسی موجودی و دارایی طرفین معامله ، قطعه کد ها و توابع زیادی برای چک کردن شرایط مختلف آن در کلاس های matcher و Security و OrderHandler وجود دارد که میتوان آنها را به کلاس جدیدی منتقل کرد که وظیفه مدیریت credit ها و position ها را دارد.