**بسمه تعالی**

**Stock Market Engine:**

دو صف خرید و فروش جهت انجام معاله سهام یک نماد وجود دارد. به عنوان مثال در نظر بگیرید یک سفارش خرید ثبت شده است و یک سفارش فروش در صف وجود دارد در صورتی که قیمت سفارش فروش مساوی یا کوچکتر از سفارش خرید باشد معامله با قیمت سفارش فروش انجام خواهد شد. مثال دیگر, در صورتی که یک سفارش خرید در صف وجود داشته باشد و سفارش فروش جدید وارد شود در صورتی که قیمت سفارش خرید مساوی یا بیشتر از قیمت فروش باشد معامله صورت میگیرد.

**Priority Queue:**

در سی شارپ نسخه 10 صف با الویت معرفی شد که الگوریتم مرتب سازی آن براساس درخت باینری می باشد. بدین صورت که صف براساس اولویت که در زمان ایجاد تعیین شده است ایجاد میگردد و دارای دو دستور کلیدی برای ورود به صف یا خروج از صف می باشد.

به عنوان مثال برای حل مشکل ایجاد صف خرید یا فروش با اولویت قیمت و ای دی میتوان از priority queue استفاده کرد.

و کد زیر نمونه کد برای ایجاد یک صف با الویت می باشد.

var patients = new List<(Patient, int)>()

{

(new("Sarah", 23), 4),

(new("Joe", 50), 2),

(new("Elizabeth", 60), 1),

(new("Natalie", 16), 5),

(new("Angie", 25), 3)

};

var hospitalQueue = new PriorityQueue<Patient, int>(patients);

hospitalQueue.Enqueue(new Patient("Roy", 23), 5);

var highestPriorityPatient = hospitalQueue.Dequeue();

نکته: در سی شارپ صف الویت دار به صورت min heap مرتب می شود و برای مرتب سازی به صورت max heap باید از اینترفیس Icomperar استفاده کرد.

Min heap: درخت براساس کم ارزش ترین اولویت مرتب می شود (نزولی).

نمونه مثال برای مرتب سازی max heap

public class ModifiedOrderPriorityMax : IComparer<Order>

{

public int Compare(Order? x, Order? y)

{

if (x.Price == y.Price)

{

return 0;

}

else if (x.Price < y.Price)

{

return 1;

}

else

{

return -1;

}

}

}

نکته: برای خواندن اولین مقدار صف بدون حذف کردن آن آیتم در متد Peek() می توان استفاده نمود.

برای درک صحیح تر می توانید به سایت زیر مراجعه نمایید.

<https://code-maze.com/csharp-priority-queue/>

**دیزاین پترن State**

یکی از زیرشاخه های الگوهای طراحی، Behavioral Design Patterns است. به شی این امکان می دهد که در صورت تغییر وضعیت, رفتار خود را تغییر دهد. پس در صورتی که بخواهیم رفتار کلاس را وقتی که مثلا مقدار یک فیلد عوض شد, تغییر بدهیم از این دیزاین پترن می توان استفاده کرد. به عنوان مثال وضعیت یک نماد در نظر بگیرید که حالت های باز, بسته, پیش گشایش دارد و در صورت باز بودن نماد می توان معامله انجام داد و در صورت بسته بودن هیچ عملیاتی نباید انجام شود و همچنین در صورت پیش گشایش فقط می توان سفارش را ثبت کرد.

**Blocking queue**

یک کالکش thread safe هست که پترن producer/consumer را پیاده سازی میکند. بدین معنی که producer دیتا را ایجاد میکند و consumer دیتا را فراخوانی میکند این صف مشکل هم زمانی را حل میکند.

به عنوان مثال مساله صف فروش و خرید در نظر بگیرید چند سفارش خرید یا فروش هم زمان بخواهد بدون مشکل ثبت شوند میتوانیم از blocking queue استفاده کنیم.

Thread safeچیست؟ چند thread همزمان میتوانند یک قطعه کد اجرا کنند بدون اینکه عملکرد برنامه مختل شود.

**نحوه ایجاد blocking queue**

**BlockingCollection<int> bCollection = new BlockingCollection<int>();**

**افزودن به صف**

bCollection.Add(1);

**حدف از صف**

bCollection.Take();

**تعریف صف با ظرفیت**

**BlockingCollection<int> bCollection = new BlockingCollection<int>();**

نکته: در صورتی که بخواهیم بیشتر از ظرفیت تعریف شده, آیتم به صف اضافه کنیم میتوانیم از مند tryadd() استفاده کنیم. بدین صورت که آیتم جدید تا زمانی که صف پر باشد نمی تواند وارد شود **بنابراین می** توانیم یک زمانی برای ورود آیتم جدید در صف در نظر بگیریم تا یک آیتمی که در صف وجود دارد حذف شود و آیتم جدید وارد صف شود.

نکته: در صورتی صف خالی باشد و بخواهیم منتظر بمانیم تا آیتم جدید اضافه شود میتوان از trytaske() استفاده کرد و همجنین میتوان برای آن زمان معین در نظر گرفت.

نکته: در thread producer زمانی که تمامی آیتم ها اضافه کردیم و آیتمی قرار نبود اضافه شود باید متد CompleteAdding صدا زد شود که در thread consumer بدانیم صف خالی شده است و زمانی که متد CompleteAdding صدا بزنیم مقدار IsCompleted مساوی با true میشود.

**نمونه کد برای ایجاد و خواندن از صف**

static void Main(string[] args)

{

BlockingCollection<int> bCollection = new BlockingCollection<int>(boundedCapacity: 10);

Task producerThread = Task.Factory.StartNew(() =>

{

for (int i = 0; i < 10; ++i)

{

Thread.Sleep(TimeSpan.FromSeconds(1));

bCollection.Add(i);

}

bCollection.CompleteAdding();

});

foreach (int item in bCollection.GetConsumingEnumerable())

{

Console.WriteLine(item);

}

//Output:

// 0

// 1

// 2

// 3

// 4

// 5

// 6

// 7

// 8

// 9

}

**برای درک صحیح تر میتوانید به سایت زیر مراجعه نمایید.**

<https://dotnetpattern.com/csharp-blockingcollection>

**دیزاین پترن command**

[دیزاین پترن](https://holosen.net/what-is-design-pattern/) Command یکی از زیرشاخه های الگوهای طراحی از نوع Behavioral است. الگوی طراحی فرمان یک درخواست را به عنوان یک شیء در بر می گیرد و بدین ترتیب به شما امکان می دهد با درخواست های مختلف ، درخواست صف یا ورود به سیستم ، پارامترها را تنظیم کرده و عملیات غیرقابل انجام را پشتیبانی کنید. بدین صورت که یک درخواست داخل یک ابجکت محصور میشود که اسم آن کامند است و ابجکت فرخوان کننده دنبال ابجکتی میگردد که میتواند این دستور اجرا کند. سپس دستور جهت اجرا به ابجکتی که می تواند آن دستور را اجرا کند ارسال میشود. به طور کلی, یک سری دستور داریم و میخواهیم آنها اجرا شوند.

مثال: در سفارش گذاری می توان از این دیزاین پترن استفاده کرد بدین شکل که در زمان سفارش گذاری میتوانیم سفارش خود را ویرایش کنیم حذف کنیم یا ثبت کنیم این کارا میتوان با ساختن یک کلاس پدر و چند کلاس فرزند انجام داد اما کد دارای خوانایی مناسب نیست به همین دلیل میتوان یک کلاس به عنوان کامند در نظر گرفت که کار آن فقط اجرای دستور است و با توجه به اجکتی که ارسال میشود دستور را اجرا می کند.

**ConfigureAwait**

در برنامه نویس async یک متد به نام Task.ConfigureAwait() وجود دارد که می تواند true یا false باشد. در صورت true بودن, بدین معنی است که ادامه تسک توسط همان thread که تسک ایجاد کرده است انجام میشود که اینکار هزینه بر است و در صورت false بودن بدین معنی است که ادامه تسک توسط thread های دیگر که در thread pool هستند اجرا شود که اینکار مقرون به صرفه تر است. پس در زمانی که نیاز نیست ادامه اجرای تسک توسط همان thread ایجاد کننده ادامه پیدا کند بهتر است مقدار false باشد. به عنوان مثال در زمانی که برنامه شما به یک UI متصل است و نتیجه باید به UI باز گردد و مقداری در UIآپدیت شود نیاز است مقدار true باشد چون که در UI فقط یک Thread وجود دارد اما در زمانی که شما در حال نوشتن یک Library هستید و نیاز به آپدیت UI نیست باید مقدار ConfigureAwait =false باشد.

**Interlock**

در برنامه نویسی async چند thread می توانند به صورت موازی یک مقدار را بخوانند و آن را تغییر دهند و این باعث میشود که دو thread خروجی یکسان داشته باشند و باعث بروز مشکل در برنامه شود. به عنوان مثال در برنامه میخواهیم هر بار یک عدد به یک متغیر اضافه شود, در صورتی که مقدار اولیه صفر باشد و دو thread به صورت موازی مقدار را بخوانند و یک عدد به آن اضافه نمایند, دو thread عدد یک را بدست می آورند در صورتی که thread دوم باید عدد دو را بدست بیاورد, برای رفع این مشکل می توان از interlock استفاده نمود که در زمان خواندن مقدار متغیر آن مقدار lock میشود و اجازه نمی دهد دو thread همزمان بتوانند مقدار را بخوانند. همچنین برای رفع این مشکل میتوان از lock استفاده نمود اما دارای performance مطلوب نمی باشد در نتیجه بهتر است از interlock برای افزایش یا کاهش مقدار یک متغیر استفاده نمود.

# **FluentAssertions**:

یک مجموعه اکستنشن متد در دات نت است که به شما این قابلیت را می دهد که به طور طبیعی بتوانید انتظارات خود را از تست بیان کنید. در نتیجه از این ابزار برای نوشتن انتظارات خودتان از تست است و این ابزار در ناگت قابل دانلود می باشد. به عنوان مثال صف خرید و فروش را در نظر بگیرید میخواهید یک تست بنویسید که در یک سفارش فروش با قیمت 80 در صف وجود دارد و یک سفارش خرید با قیمت 80 ثبت میشود در این حالت انتظار دارید یک معامله با قیمت 80 انجام شود و در لیست معاملات ثبت شود شما میتوانید توسط این ابزار یک ابجکت از انتظار خود ایجاد نمایید و آن را با لیست معاملات مقایسه نمایید.

نمونه کد تست:

public async Task ProcessOrderAsync\_Should\_One\_Trade\_Commit\_With\_Amount10\_And\_Price100()

{

//Arrange

sut.PreOpen();

sut.Open();

var sellOrderId = await sut.ProcessOrderAsync(100, 10, Side.Sell);

//Act

var buyOrderId = await sut.ProcessOrderAsync(100, 10, Side.Buy);

//Assert

sut.Trade.

Where(o => o.Id == 1)

.FirstOrDefault()

.Should()

.BeEquivalentTo(

new

{

Id = 1,

BuyOrderId = buyOrderId

,

SellOrderId = sellOrderId,

Amount = 10,

Price = 100

});

}

برای درک صحیح تر می توانید به سایت زیر مراجعه نمایید.

<https://fluentassertions.com/introduction>

# Lock:

در برنامه نویسی multithreading هندل کردن چند thread به طور همزمان برای برنامه نویسان دارای اهمیت بالایی می باشد. Lock کردن یکی از روش های مدیریت multithreading می باشد. این دستور به شما اطمینان می دهد که در یک لحظه فقط یک thread می تواند یک قطعه کد را اجرا کند. بدین معنا است که این دستور اجازه نمیدهد زمانی که یک thread در حال اجرا یک کد است thread دیگر به طور هم زمان آن کد را اجرا کند و thread دیگر باید صبر کند تا thread قبلی کار خود را به اتمام برساند. به عنوان مثال هسته معاملات در نظر بگیرید که دو thread به طور همزمان میخواند سفارشات ثبت شده در دیتابیس بخوانند که در این صورت دو thread به دیتا یکسان دسترسی دارند و آخرین دیتا آپدیت شده را ندارند در نتیجه میتوانیم از دستور lock استفاده نماییم.

نمونه کد:

static bool isInitialized;

static object initLock = new object();

if (!isInitialized)

{

lock (initLock)

{

// init code here

isInitialized = true;

}

}

برای درک صحیح تر می توانید به سایت زیر مراجعه نمایید.

<https://www.c-sharpcorner.com/UploadFile/1d42da/thread-locking-in-C-Sharp/>

# covariant and contravariant

اصول طراحی متد ها: