Day 06 Многопоточность

## Общие требования

* Убедитесь, что на вашем компьютере установлен [SDK для разработки на .NET 5](https://dotnet.microsoft.com/download) и вы используете именно его.
* Помните, ваш код будут читать! Обратите особое внимание на оформление вашего кода и именование переменных. Обязательно придерживайтесь общепринятых стандартов [C# Coding Conventions](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/inside-a-program/coding-conventions).
* Самостоятельно выберите удобную для себя IDE.
* Программа должна иметь возможность запуска через командную строку dotnet.
* В каждом из заданий указаны примеры ввода и вывода. Решение должно использовать их как верный формат.
* В начале каждого задания приведен список рекомендованных языковых конструкций.
* Если затрудняетесь в решении задачи, обратитесь с вопросами к другим участникам бассейна, интернету, Google, посмотрите на StackOverflow.
* С основными возможностями языка C# можно ознакомиться в [официальной спецификации](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/language-specification/introduction).
* Избегайте **хардкода** и “**магических чисел**”.
* Вы демонстрируете все решение, верный результат работы программы – лишь один из способов проверки ее корректной работы. Поэтому когда необходимо получить определенный вывод в результате работы ваших программ, запрещено показывать пред рассчитанный результат.
* Обращайте особое внимание на термины, выделенные **bold** шрифтом: их изучение пригодится вам как в выполнении текущего задания, так и в вашей дальнейшей карьере .NET разработчика.
* Have fun :)

## Требования к заданиям дня

* Все сборки должны быть в одном решении.
* Каждому из заданий должно соответствовать отдельное консольное приложение, созданное на основе стандартного шаблона .NET SDK.
* Используйте **var**.
* Название решения (и его отдельного каталога) - d{*xx*}, где *xx* - цифры текущего дня.
* Для форматирования выходных данных используйте **культуру** en-GB: N2 для вывода денежных сумм, d для дат.

## Что нового

* TimeSpan
* Parallel computing, multithreading
* Deadlocks, race conditions
* PLINQ

## Интро

Времена, люди и задачи меняются, но есть в мире вещи, которые остаются вечными. Любовь, теория относительности, сияние чистого разума. Очереди. Почему бы нам не вернуться к тому вопросу, который мы уже рассматривали на старте нашего курса обучения?

В течение Дня 01 мы реализовали предметную модель, основанную на работе магазина, его покупателей и касс. Давайте переиспользуем эту модель, но доведем эмуляцию очередей до конца.

Воспользуйтесь собственным результатом Дня 01 или скачайте исходный код [предметной модели](https://github.com/smarthead/dotnet_exercise06). Это будет основой нашего проекта, изучите ее и разберитесь, если еще не сталкивались с заданиями Дня 01.

А теперь добавим сложности и параллельной работы. Наша задача сегодня: создать полноценную симуляцию параллельной работы нескольких касс с очередями в магазине.

## Дополнительно

Многопоточное программирование в .NET - очень глубокая тема, полностью изучить которую за день просто невозможно. Сегодня вы получите лишь поверхностное представление о некоторых из возможностей.

Однако при желании вы можете самостоятельно познакомиться с этой областью знаний с помощью этой бесплатной онлайн книги: [Threading in C#](http://www.albahari.com/threading/).

# Exercise 00 Законы мира

“The story so far:

In the beginning the Universe was created.

This has made a lot of people very angry and been widely regarded as a bad move.”

― **Douglas Adams,** [**The Restaurant at the End of the Universe**](https://www.goodreads.com/work/quotes/1877624)

Кассы справляются с покупателями и их покупками не мгновенно. Добавьте в класс *CashRegister* два новый свойства: время, которое кассир тратит на каждый предмет в корзине, и время, которое кассир тратит на переключение между покупателями. Для простоты пусть это будет количество секунд. Решите сами, хотите ли вы воспользоваться типом **TimeSpan**.

Пусть эти параметры определяются глобально для магазина: реализуйте это в конструкторах классов *CashRegister* и *Store.* Доработайте приложение так, чтобы значения этих параметров (в секундах) брались при запуске из конфигурации приложения в файле **appsettings.json**.

Подразумевается, что все кассы магазина работают и пробивают товары [параллельно](https://docs.microsoft.com/en-us/dotnet/standard/threading/threads-and-threading), не так ли? То есть каждая - в своем потоке.

Давайте реализуем эмуляцию занятости одной кассы: добавьте в *CashRegister* метод *Process*, в рамках которого будет обрабатываться один покупатель. В методе [приостановите текущий поток обработки](https://docs.microsoft.com/en-us/dotnet/standard/threading/pausing-and-resuming-threads) на время, соответствующее времени обработки одного покупателя и всех его товаров. Не забудьте посчитать это время!

Добавьте в класс *CashRegister* свойство, которое позволит получить все время загрузки кассы. Сделайте так, чтобы это время изначально было нулевым, но увеличивалось каждый раз, когда обрабатывается новый пользователь - увеличивалось на то время, сколько в общей сложности заняла обработка.

Мы сделали эмуляцию нагрузки кассы. Попробуйте отладить ее работу и посмотрите, верно ли выполняются задержки.

# Exercise 01 Борьба за ресурсы

“Profitable bookstores sell books. Unprofitable book sellers store books.”

― **Mokokoma Mokhonoana**

В нашей модели каждый покупатель берет товары с одного и того же склада - за его работу отвечает класс *Storage*. И если касс может быть много, то склад магазина все еще один. И это значит, что параллельно работающие кассы претендуют на ограниченное количество товаров на одном складе - создают таким образом конкуренцию.

При использовании нескольких потоков важно понимать, что они могут блокировать совместно используемые ресурсы. Более того, эта блокировка важна для поддержания данных **консистентными**. Для более глубокого понимания обратитесь к [лучшим практикам](https://docs.microsoft.com/en-us/dotnet/standard/threading/managed-threading-best-practices).

Добавим в класс *Storage* метод, позволяющий взять со склада необходимое число товаров. Воспользуйтесь средствами класса [Interlocked](https://docs.microsoft.com/en-us/dotnet/api/system.threading.interlocked?view=net-5.0), чтобы обеспечить **потокобезопасный механизм блокировки ресурсов**.

# Exercise 02 Параллели

“I am free of all prejudice. I hate everyone equally. ”

― **W.C. Fields**

Теперь когда мы реализовали работу каждой отдельной кассы и механизм “покупки” товаров со склада, давайте сделаем так, чтобы магазин запускал работу всех касс параллельно.

Добавьте в класс *Store* метод *OpenRegisters,* который для всех касс магазина будет [запускать отдельный поток](https://docs.microsoft.com/en-us/dotnet/standard/threading/creating-threads-and-passing-data-at-start-time): пока на складе есть товары, каждая касса должна последовательно обработать каждого покупателя в своей очереди. При этом “покупка” должна совершаться с уменьшением соответствующего количества товаров на складе.

Создайте магазин с несколькими кассами и вручную добавьте в их очереди несколько покупателей с товарами. Проверьте себя отладкой, все ли работает верно? Для информации выводите информацию о каждом покупателе на кассе в консоль.

# Exercise 03 На старт

“It ain't no fun if the homies can't have none. ”

― **Snoop Dogg**

Теперь обратимся к финальной части нашего уравнения: покупателям. Они тоже делают выбор кассы параллельно, давайте реализуем это. Пусть на старте работы магазина в нем уже будет какое-то начальное количество покупателей.

Предположим, что на старте у нас есть 20 покупателей. Создайте их и сделайте так, что перед запуском магазина каждый из них заполнит свою корзину и встанет в очередь на кассу, но сделайте это параллельно. Здесь вам помогут средства [PLINQ](https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/introduction-to-plinq), посмотрите в сторону метода **Parallel.ForEach**.

Мы обеспечили нашему магазину стартовых покупателей и их параллельный выбор очереди. Попробуйте отладить свой код и посмотрите, все ли выполняется верно.

# Exercise 04 Внимание, марш!

“Any fool can know. The point is to understand.”

― **Albert Einstein**

Кажется, все кусочки мозаики реализованы по отдельности, время сложить их воедино. Допишите код приложения так, чтобы сэмулировать следующую ситуацию.

Есть магазин, в котором одновременно работают 4 кассы и склад которого вмещает до 50 товаров. В кассы могут выстраиваться очереди покупателей. У каждого покупателя в корзине может быть от 1 до 7 товаров. Для того, чтобы провести один товар через кассу, требуется 500 мс. Для смены покупателя требуется 2 секунды. Пока магазин открыт, каждые 5 секунд появляется новый покупатель.

На старте работы касс в магазине находятся 10 покупателей. Вам необходимо смоделировать два режима работы магазина.

Покупатели всегда выбирают самую короткую очередь. Новые покупатели всегда выбирают очередь с наименьшим количеством покупателей, делая предположение о том, что такая очередь продвигается быстрее.

Покупатели выбирают очередь с наименьшим количеством товаров. Новые покупатели всегда выбирают очередь с наименьшим количеством товаров в корзинах, делая предположение о том, что очередь в которой у покупателей корзины заполнены меньше движется быстрее.

Для каждого покупателя, закончившего покупку, выведите в консоль: кассу, покупателя, количество товаров в корзине, количество покупателей в очереди за ним. После закрытия магазина выведите для каждой кассы: кассу, среднее время покупателя в очереди в этой кассе (для этого у вас есть общее время обработки всех и количество прошедших покупателей).

# Bonus A little bit random

“Ph'nglui mglw'nafh Cthulhu R'lyeh wgah'nagl fhtagn.

― **H.P. Lovecraft,** [**The Call of Cthulhu**](https://www.goodreads.com/work/quotes/25692046)

Привнесем в работу нашей эмуляции еще больше хаоса, сделав ее еще больше похожей на реальность.

Измените код приложения так, чтобы время обработки одного покупателя и одного товара на каждой кассе был не постоянной величиной, глобальной для всего магазина, а вычислялся как рандомное количество секунд от 1 до заданного максимума (использованные ранее и определяемые в appsettings.json параметры timePerItem и timePerCustomer). Для каждой кассы.

Для каждого покупателя, закончившего покупку, выведите в консоль: кассу, покупателя, количество товаров в корзине, количество покупателей в очереди за ним. После закрытия магазина выведите для каждой кассы: кассу, среднее время покупателя в очереди в этой кассе (для этого у вас есть общее время обработки всех и количество прошедших покупателей).