Задачи

?.

Известно, что объекты могут обладать сложной внутренней структурой, и обращение к полям вложенных объектов может быть довольно длинным, например, A.B.C.D. Реализовать обобщённый метод, принимающий в качестве обобщённого параметра некоторый класс T и возвращающий Func<T,U>, который для заданного строкой пути в классе (например, “B.C.D” или “B[1].C[2].C[42]”) создаёт делегат, получающий для объекта класса T значение, лежащее по заданному пути. При этом все промежуточные элементы класса – это:

А) Свойства

Б) Поля

В) Массивы

Г) Объекты обобщённого интерфейса ICollection<T>.

Показать тестами работоспособность решения.

Пул потоков

Реализовать объект ThreadPool, реализующий паттерн «пул потоков» с поддержкой continuation (наподобие https://docs.microsoft.com/en-us/dotnet/api/system.threading.threadpool?view=netframework-4.8 + https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskfactory?view=netframework-4.8).

Пул потоков:

- Число потоков задаётся константой в классе пула или параметром конструктора.
- У каждого потока есть два состояния: ожидание задачи, выполнение задачи.
- Задача — вычисление некоторого значения, описывается в виде Func<TResult> и инкапсулируется в объектах интерфейса IMyTask<TResult>.
- Добавление задачи осуществляется с помощью нестатического метода класса пула Enqueue(IMyTask<TResult> a).
- При добавлении задачи, если в пуле есть ожидающий поток, то он должен приступить к ее исполнению. Иначе задача будет ожидать исполнения, пока не освободится какой-нибудь поток.
- Класс должен быть унаследован от интерфейса IDisposable и корректно освобождать ресурсы при вызове метода Dispose().
- Метод Dispose должен завершить работу потоков. Завершение работы коллаборативное, с использованием CancellationToken — уже запущенные задачи не прерываются, но новые задачи не принимаются на исполнение потоками из пула.
- Возможны два варианта решения --- дать всем задачам, которые уже попали в очередь, досчитаться, либо выбросить исключение во все ожидающие завершения задачи потоки.

IMyTask:

- Свойство IsCompleted возвращает true, если задача выполнена.
- Свойство Result возвращает результат выполнения задачи.
- В случае, если соответствующая задаче функция завершилась с исключением, этот метод должен завершиться с исключением AggregateException, содержащим внутри себя исключение, вызвавшее проблему.

- Если результат еще не вычислен, метод ожидает его и возвращает полученное значение, блокируя вызвавший его поток.
- Метод ContinueWith — принимает объект типа Func<TResult, TNewResult>, который может быть применен к результату данной задачи X и возвращает новую задачу Y, принятую к исполнению.
- Новая задача будет исполнена не ранее, чем завершится исходная.
- В качестве аргумента объекту Func будет передан результат исходной задачи, и все Y должны исполняться на общих основаниях (т.е. должны разделяться между потоками пула).
- Метод ContinueWith может быть вызван несколько раз.
- Метод ContinueWith не должен блокировать работу потока, если результат задачи X ещё не вычислен.
- ContinueWith должен быть согласован с Shutdown – принятая как ContinueWith задача должна либо досчитаться, либо бросить исключение ожидающему её потоку.

Ограничения:

- В данной работе запрещено использование TPL, PLINQ и библиотечных классов Task и ThreadPool.
- Все интерфейсные методы должны быть потокобезопасны.
- Для каждого базового сценария использования должен быть написан несложный тест (добавление 1 задачи, добавление задач, количественно больших числа потоков, проверка работы ContinueWith для нескольких задач). Для всех тестов обязательна остановка пула потоков.
- Также должен быть написан тест, проверяющий, что в пуле действительно не менее n потоков.

API

Написать веб-сервис с использованием REST API для отображения текущей погоды в Санкт-Петербурге. Источниками данных являются открытые веб-сервисы, например:

1) https://www.tomorrow.io/

2) https://stormglass.io/

3) https://openweathermap.org/api

В сервисе необходимо показать данные не менее, чем от двух источников. Требуется отобразить текущую температуру (в градусах Цельсия и Фаренгейта), облачность, влажность, осадки, направление и скорость ветра. В случае, если соответствующих данных нет, отображать надпись «Данных нет». Предусмотреть корректную обработку ошибок в случае, если сервисы по какой-либо причине недоступны. Реализация должна следовать принципам ООП.

Общение с REST API происходит по протоколу HTTP (HTTPS допустим, если сможете раздобыть и прикрутить SSL-сертификат, но не является обязательным). Методы REST API позволяют получить список сервисов, с которым ваш интегрирован, и погоду от каждого из них или ото всех сразу. Данные пересылаются в формате JSON. Решение должно включать в себя автоматически сгенерированную документацию о вашем API в формате OpenAPI (например, ч помощью подключения Swagger).

Написать юнит-тесты, демонстрирующие работоспособность решения. Сборку решения оформить в виде Dockerfile или файла Docker compose. Docker-образ выдаёт API на 80 порту (8080 в случае HTTPS). Ключи сервисов прописывать или в отдельном env-файле, или в параметрах к докер-образу.