

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

по лабораторной работе №1

по дисциплине «Технологии программирования»

Автор: Евтюхов Дмитрий

Факультет: ФИТиП

Группа: М32011

Преподаватель: Ивницкий Алексей



УНИВЕРСИТЕТ ИТМО

Санкт-Петербург 2021

Условие лабораторной работы

lab-1. Hello world

1. Изучить механизм интеропа между языками, попробовать у себя вызывать C/C++ (Не C++/CLI) код (суммы чисел достаточно) из Java и C#. В отчёте описать логику работы, сложности и ограничения этих механизмов.
2. Написать немного кода на Scala и F# с использованием уникальных возможностей языка - Pipe operator, Discriminated Union, Computation expressions и т.д. . Вызвать написанный код из обычных соответствующих ООП языков (Java и C#) и посмотреть во что превращается написанный ранее код после декомпиляции в них.
3. Написать алгоритм обхода графа (DFS и BFS) на языке Java, собрать в пакет и опубликовать (хоть в Maven, хоть в Gradle, не имеет значения). Использовать в другом проекте на Java/Scala этот пакет. Повторить это с C#/F#. В отчёте написать про алгоритм работы пакетных менеджеров, особенности их работы в C# и Java мирах.
4. Изучить инструменты для оценки производительности в C# и Java. Написать несколько алгоритмов сортировок (и взять стандартную) и запустить бенчмарки (в бенчмарках помимо времени выполнения проверить аллокации памяти). В отчёт написать про инструменты для бенчмаркинга, их особенности, анализ результатов проверок.
5. Используя инструменты dotTrace, dotMemory, всё-что-угодно-хоть-windbg, проанализировать работу написанного кода для бекапов. Необходимо написать сценарий, когда в цикле будет выполняться много запусков, будут создаваться и удаляться точки. Проверить два сценария: с реальной работой с файловой системой и без неё. В отчёте необходимо проанализировать полученные результаты, сделать вывод о написанном коде. Опционально: предложить варианты по модернизации или написать альтернативную имплементацию.

Задание 1

Что такое интероп?

Interop — это способ «общения» между двумя разными языками или платформами.

Без него мы не смогли бы легко зависеть от существующего кода, созданного другими платформами и языками, и нам пришлось бы перестраивать его самостоятельно.

Interop позволяет пользователям .NET взаимодействовать с кодом, отличным от .NET.

Без взаимодействия пользователи не смогут использовать библиотеки, отличные от .NET, в своих приложениях .NET.

Немного информации из лекции:

1) .NET поддерживает мультиязычность благодаря CLR (Common Language Runtime).

CLR (Common language runtime) — общезыковая исполняющая среда. Она обеспечивает интеграцию языков и позволяет объектам благодаря стандартному набору типов и метаданным), созданным на одном языке, быть «равноправными гражданами» кода, написанного на другом.

Другими словами CLR этот тот самый механизм, который позволяет программе выполняться в нужном нам порядке, вызывая функции, управляя данными.

2) CLR генерирует IL-код, который не привязан к ЯП и впоследствии может быть преобразован в машинный код.

IL (Intermediate Language) — код на специальном языке, напоминающем ассемблер, но написанном для .NET. В него преобразуется код из других языков верхнего уровня (с#, VisualBasic). Пропадает зависимость от выбранного языка. Все преобразуется в IL.

3) В рамках одного солюшена можно использовать разные языки, вызывать их друг из друга (с некоторыми ограничениями).

Ограничения:

1) Важно понимать, что механизм интеропа не типобезопасный (type safe).

Соответственно, есть вероятность возникновения ошибок, связанных с типами данных. (Может отсутствовать прямое преобразование между типами).

2) Также, стоит отметить, что организация механизма интеропа выливается в дополнительные строки кода, потому что при добавлении новой платформы будет необходимо прописать еще одно взаимодействие (т е interop =) /

Случай интеропа между C++ и C#.

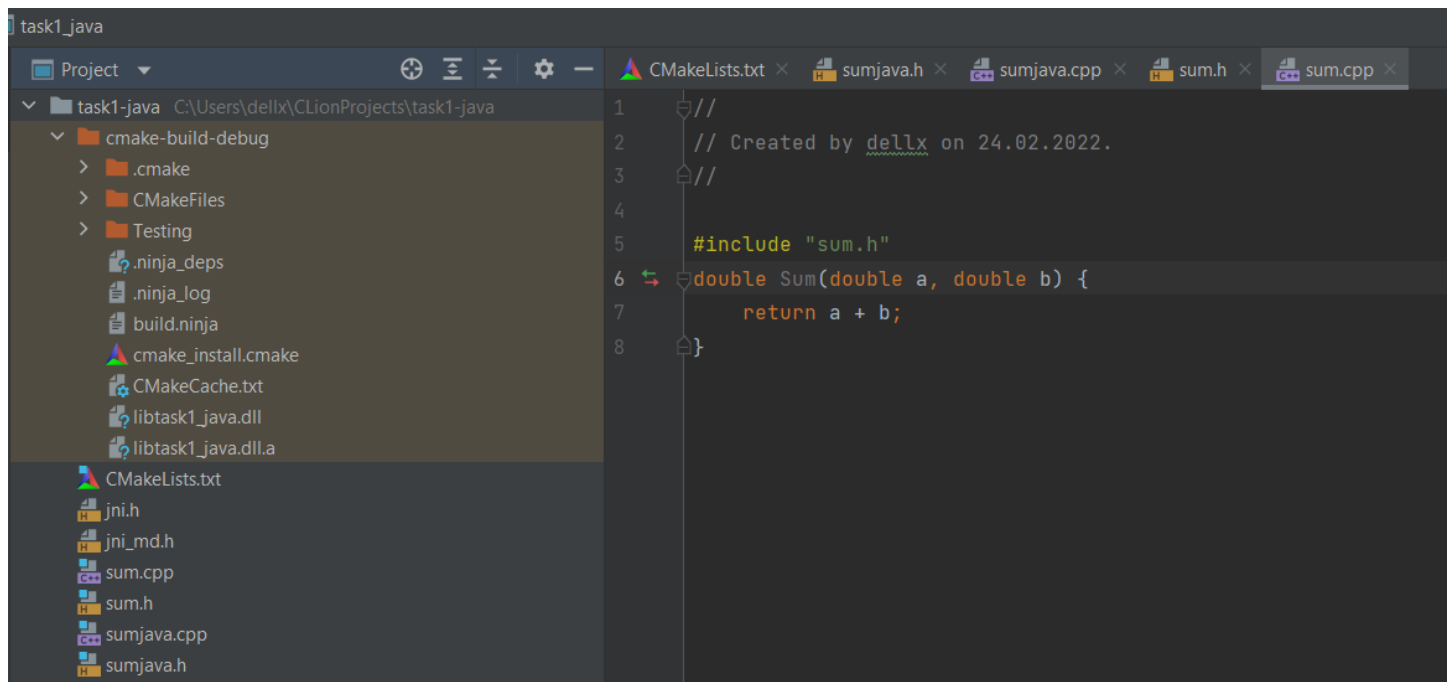
Хотим использовать код на C++ в C#.

Ход событий:

Создаем .cpp, который содержит весь необходимый код.

Компилируя, получаем IL код (.dll)

Далее подключаем dll к проекту на C#. (DLL — Dynamic Link Library — динамическая подключаемая библиотека в операционной системе (ОС) Windows.)



```
1 //  
2 // Created by dellx on 24.02.2022.  
3 //  
4  
5 #include "sum.h"  
6 double Sum(double a, double b) {  
7     return a + b;  
8 }
```

Это код предназначен для запуска на C#.

Создаем header.(он ниже)

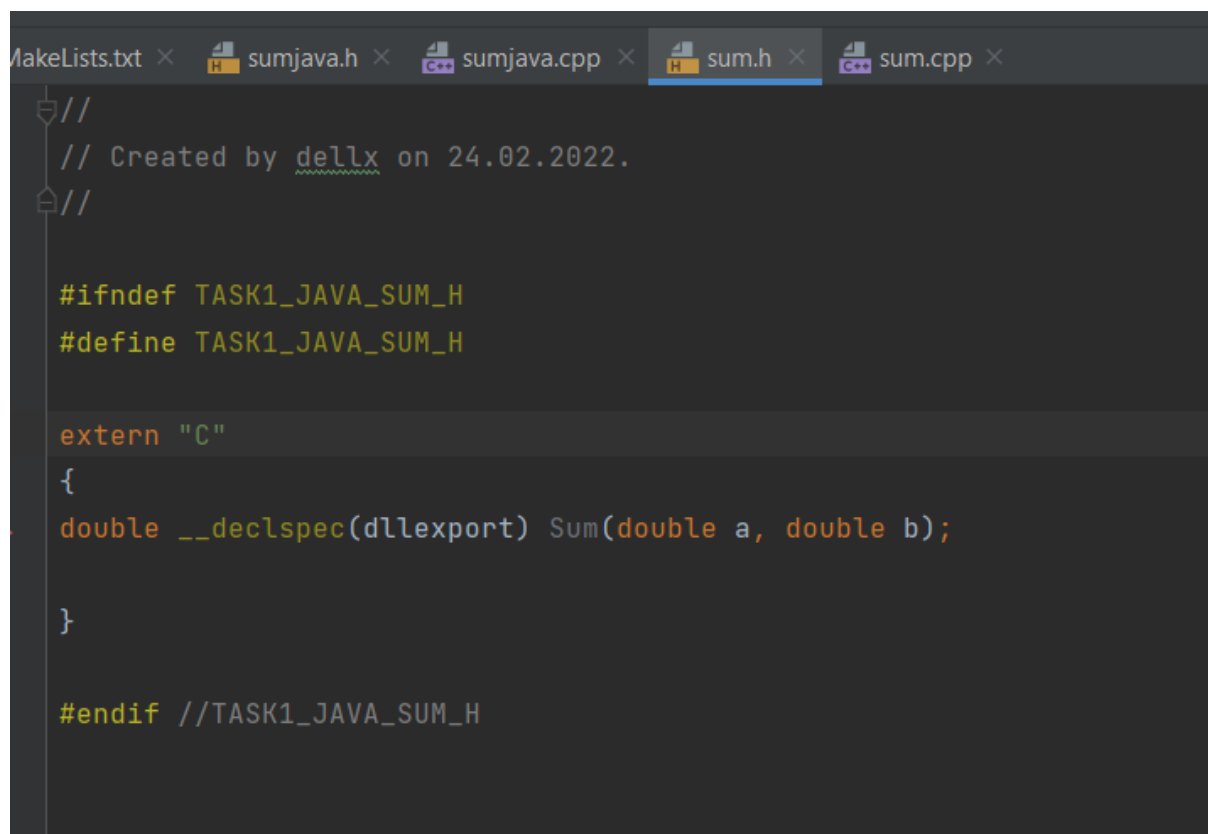
Ключевое слово extern может быть применено к глобальной переменной, функции или объявлению шаблона. Указывает, что символ имеет внешнюю связь.

extern "C" указывает, что функция определена в другом месте и использует соглашение о вызовах языка C.

Соглашение о вызове (англ. calling convention) — описание технических особенностей вызова подпрограмм, определяющее:

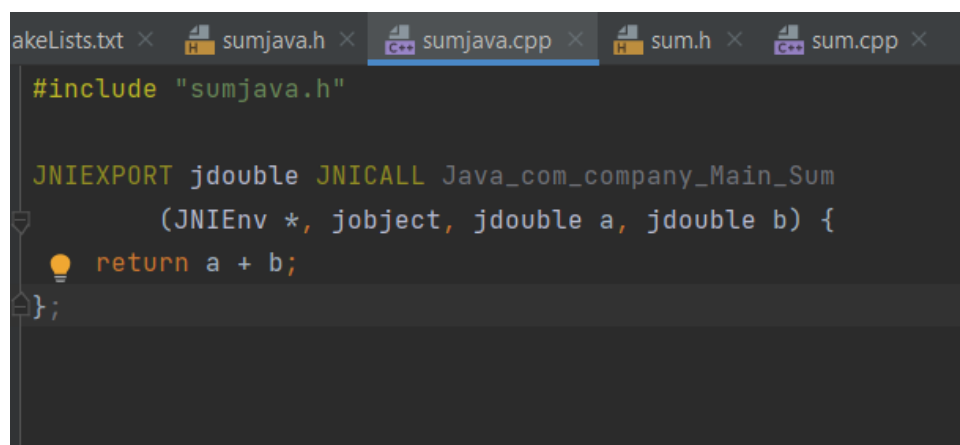
- 1) способы передачи параметров подпрограммам;
- 2) способы вызова (передачи управления) подпрограмм;
- 3) способы передачи результатов вычислений, выполненных подпрограммами, в точку вызова;
- 4) способы возврата (передачи управления) из подпрограмм в точку вызова.

`__declspec (dllexport)` используется для функций и данных, ЭКСПОТИРУЕМЫХ библиотекой динамической компоновки (DLL), и может использоваться вместе с файлом DEF или вместо него. Файлы определения модуля (DEF) предоставляют компоновщику сведения о экспорте, атрибутах и другие сведения о программе, которую необходимо связать. DEF-файл наиболее удобен при создании библиотеки DLL.



```
//  
// Created by dellx on 24.02.2022.  
//  
  
#ifndef TASK1_JAVA_SUM_H  
#define TASK1_JAVA_SUM_H  
  
extern "C"  
{  
    double __declspec(dllexport) Sum(double a, double b);  
}  
  
#endif //TASK1_JAVA_SUM_H
```

Снизу приведена реализация `sumjava.cpp`

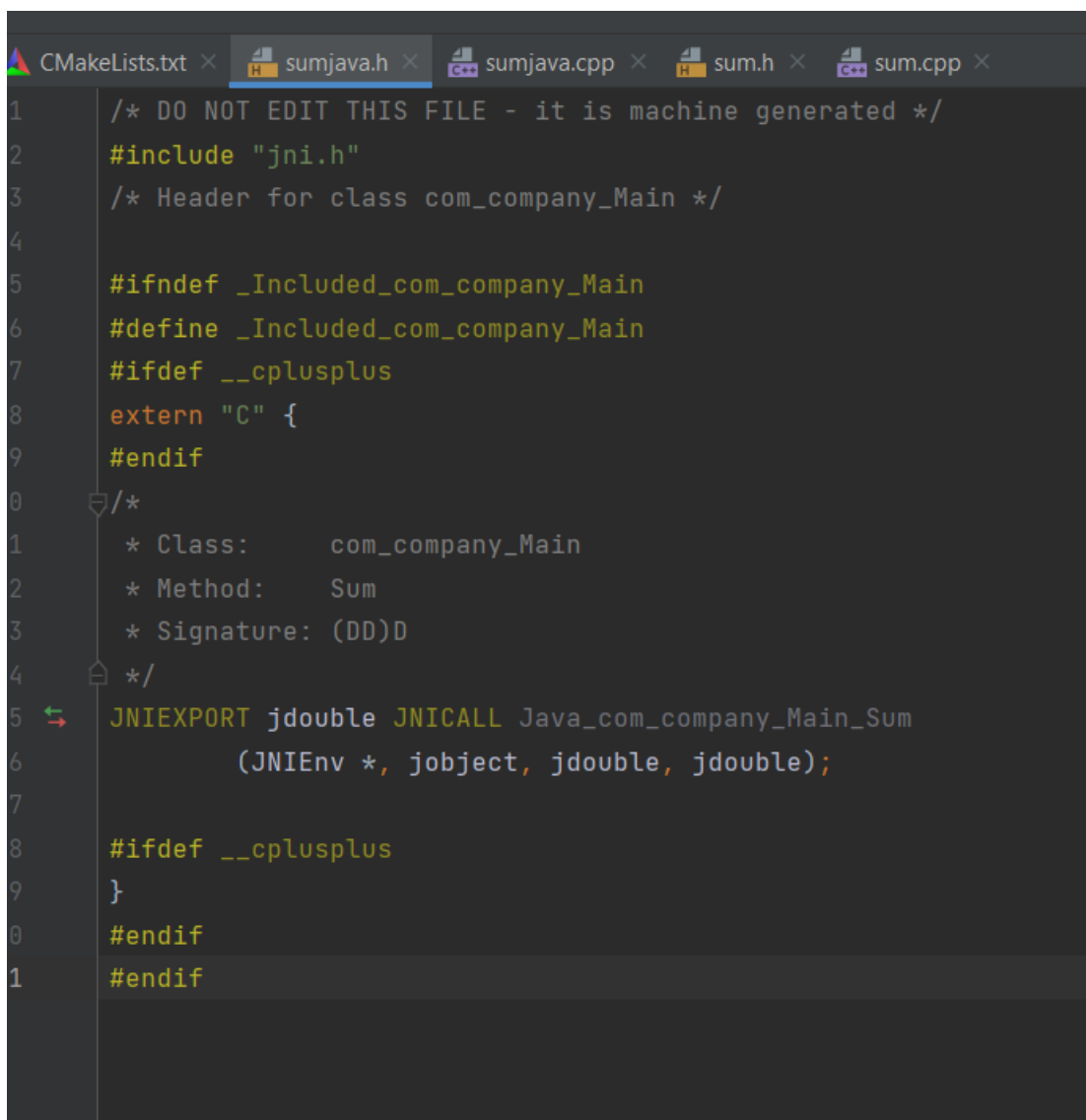


```
#include "sumjava.h"  
  
JNIEXPORT jdouble JNICALL Java_com_company_Main_Sum  
    (JNIEnv *, jobject, jdouble a, jdouble b) {  
    return a + b;  
};
```

`JNIEXPORT` - помечает функцию в общей библиотеке как экспортируемую, чтобы JNI смог ее найти (будет лежать в таблице функций).

`JNICALL` — в сочетании с `JNIEXPORT` обеспечивает доступность методов для среды JNI.

Java Native Interface (JNI) — стандартный механизм для запуска кода под управлением виртуальной машины Java (JVM), который написан других языках и скомпонован в виде динамических библиотек.



```
1  /* DO NOT EDIT THIS FILE - it is machine generated */
2  #include "jni.h"
3  /* Header for class com_company_Main */
4
5  #ifndef _Included_com_company_Main
6  #define _Included_com_company_Main
7  #ifdef __cplusplus
8  extern "C" {
9  #endif
10 /*
11  * Class:      com_company_Main
12  * Method:     Sum
13  * Signature:  (DD)D
14  */
15 JNIEXPORT jdouble JNICALL Java_com_company_Main_Sum
16     (JNIEnv *, jobject, jdouble, jdouble);
17
18 #ifdef __cplusplus
19 }
20 #endif
21 #endif
```

Формирование имени для метода, который будет использоваться в Джаве:

Java - ключевое слово

Далее название класса, в котором метод должен будет использоваться

Далее само название метода

Вызов кода на c# и java:

```
using System;
using System.Runtime.InteropServices;
namespace Task1
{
    public class Exercise
    {
        [DllImport(@"C:\Users\delix\CLionProjects\task1-java\cmake-
build-debug\libtask1_java.dll", CallingConvention =
CallingConvention.Cdecl)]
        public static extern double Sum(double a, double b);
        public double Summary(double a, double b)
        {
            var x = Sum(a, b);
            return x;
        }
    }
}
```

DllImport - получает всю необходимую информацию для нахождения и последующего использования библиотеки.

Про Джаву:

Ключевое слово native говорит о том, что перед нами что-то нечто абстрактного метода. Любой таковой метод, должен быть реализован в shared библиотеке.

```

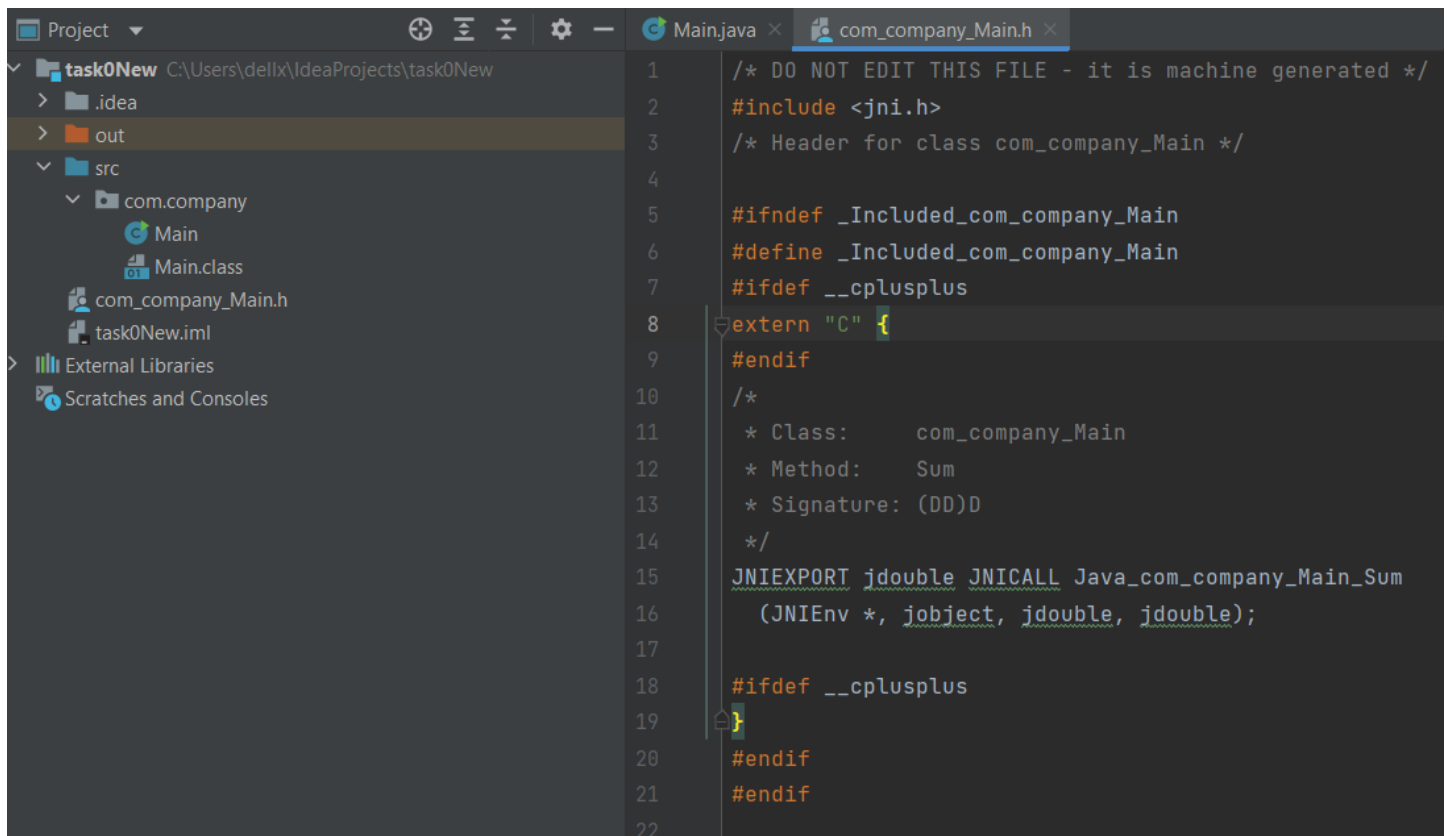
package com.company;

public class Main {
    static {
        System.load("C:\\Users\\dellx\\CLionProjects\\task1-
java\\cmake-build-debug\\libtask1_java.dll");
    }

    public static void main(String[] args) {
        System.out.println(new Main().Sum(2,2));
    }

    private native double Sum(double a, double b);
}

```



Задание 2

Начнем с F# и C#.

F# — мультипарадигмальный язык программирования из семейства языков .NET, поддерживающий функциональное программирование в дополнение к императивному (процедурному) и объектно-ориентированному программированию.

Исходный код на F#:

Я осознаю, что код может быть плохо читаемым, поэтому

<https://pastebin.com/s00WMCX2>

```
module MyProgram
open System
open System.IO
```

//Это пример применения одной из возможностей F# - pipe operator

```
let (|>) x f = f x

let func x = x * x + 2 * x + 4 - 11 * 11 - x * x * x

let toString (x : int) = x.ToString() + "DimaEvtyukhov"

let reverseString (x : string) = new String(Array.rev
(x.ToCharArray())) + "M32011"

let result1 (x : int) = x |> func |> toString |> reverseString
```

//Это пример Discriminated Union

```
type GeometricFigure =
    | Rectangle of width : float * length : float
    | Circle of radius : float
    | Triangle of length : float
    | Ellipse of focus : float

let rectangle = Rectangle(length = 100., width = 100.)
let circle = Circle (10.)
let triangle = Triangle(11.)
let ellipse = Ellipse(228.)

let getFigureParameter figure =
    match figure with
```

```

| Rectangle(length = len) -> len |> printfn "%A"
| Circle(radius = r) -> r |> printfn "%A"
| Triangle(length = len) -> len |> printfn "%A"
| Ellipse(focus = f) -> f |> printfn "%A"
let result2 = rectangle |> getFigureParameter

```

//Computation expressions

```

type Logger() =
    let log p = printfn "logged %A" p
    member this.Bind(x, f) =
        log x
        f x
    member this.Return(x) = x
let logger = new Logger()
let loggedWork =
    logger
    {
        let! x = 1
        let! y = 5
        let! z = x + y
        return z
    }

```

Теперь декомпилируем код на F# в код на C#:

<https://pastebin.com/dAWssRQk>

(Так как код очень объемный, больше 1000 строк кода, решил полный код выложить на pastebin, а в отчете привести лишь какие-то интересные части)

Код начинается с создания абстрактного класса геометрической фигуры, дальнейшего объявления классов-наследников: Rectangle, Circle, Triangle, Ellipse.

Причем, каждый получает свой уникальный Teg (от 0 до 3 соответственно).

```

public abstract class GeometricFigure : IEquatable<GeometricFigure>, IStructuralEquatable, IComparable<GeometricFigure>, IComparable,
IStructuralComparable
{
    public static class Tags
    {
        public const int Rectangle = 0;

        public const int Circle = 1;

        public const int Triangle = 2;

        public const int Ellipse = 3;
    }
}

```

Далее идет объявление классов-наследников (например, круга)

```

public class Circle : GeometricFigure
{
    [DebuggerBrowsable(DebuggerBrowsableState.Never)]
    [CompilerGenerated]
    [DebuggerNonUserCode]
    internal readonly double _radius;

    [CompilationMapping(SourceConstructFlags.Field, 1, 0)]
    [CompilerGenerated]
    [DebuggerNonUserCode]
    public double radius
    {
        [CompilerGenerated]
        [DebuggerNonUserCode]
        get
        {
            return _radius;
        }
    }

    [CompilerGenerated]
    [DebuggerNonUserCode]
}

```

```

        internal Circle(double _radius)

            : base(1)

        {

            this._radius = _radius;

        }

    }

```

Далее идет геттер тега фигуры, а также реализация bool методов is(любое название фигуры)

```

public int Tag
{

    [CompilerGenerated]

    [DebuggerNonUserCode]

    get

    {

        return _tag;

    }

}

[CompilerGenerated]

[DebuggerNonUserCode]

[DebuggerBrowsable(DebuggerBrowsableState.Never)]

public bool IsRectangle
{

    [CompilerGenerated]

    [DebuggerNonUserCode]

    get

    {

        return Tag == 0;

    }

}

```

Далее идет перегрузка методов Equals и GetHashCode, а также, что самое главное - реализация CompareTo и непосредственно самого Discriminated Union

[CompilerGenerated]

```
public sealed override int CompareTo(object obj, IComparer comp)
{
    GeometricFigure geometricFigure = (GeometricFigure)obj;

    if (this != null)
    {
        if ((GeometricFigure)obj != null)
        {
            int tag = _tag;

            int tag2 = geometricFigure._tag;

            if (tag == tag2)
            {
                return CompareTo$cont@11-1(comp, this, geometricFigure, null);
            }

            return tag - tag2;
        }

        return 1;
    }

    if ((GeometricFigure)obj != null)
    {
        return -1;
    }

    return 0;
}
```

Создание новых фигур

```
[CompilationMapping(SourceConstructFlags.UnionCase, 0)]  
    public static GeometricFigure NewRectangle(double _width, double _length)  
  
    {  
  
        return new Rectangle(_width, _length);  
  
    }
```

```
[CompilationMapping(SourceConstructFlags.UnionCase, 1)]  
  
public static GeometricFigure NewCircle(double _radius)  
  
{  
  
    return new Circle(_radius);  
  
}
```

```
[CompilationMapping(SourceConstructFlags.UnionCase, 2)]  
  
public static GeometricFigure NewTriangle(double _length)  
  
{  
  
    return new Triangle(_length);  
  
}
```

```
[CompilationMapping(SourceConstructFlags.UnionCase, 3)]  
  
public static GeometricFigure NewEllipse(double _focus)  
  
{  
  
    return new Ellipse(_focus);  
  
}
```

//Основная часть

```
public static void getFigureParameter(GeometricFigure figure)  
  
{  
  
    switch (figure.Tag)  
  
    {  
  
        default:
```

```

{

    GeometricFigure.Rectangle rectangle = (GeometricFigure.Rectangle)figure;

    double focus = rectangle._length;

    PrintfFormat<FSharpFunc<double, Unit>, TextWriter, Unit, Unit> format = new
PrintfFormat<FSharpFunc<double, Unit>, TextWriter, Unit, Unit, double>("%A");

    PrintfModule.PrintFormatLineToTextWriter(Console.Out, format).Invoke(focus);

    break;

}

case 1:

{

    GeometricFigure.Circle circle = (GeometricFigure.Circle)figure;

    double focus = circle._radius;

    PrintfFormat<FSharpFunc<double, Unit>, TextWriter, Unit, Unit> format = new
PrintfFormat<FSharpFunc<double, Unit>, TextWriter, Unit, Unit, double>("%A");

    PrintfModule.PrintFormatLineToTextWriter(Console.Out, format).Invoke(focus);

    break;

}

case 2:

{

    GeometricFigure.Triangle triangle = (GeometricFigure.Triangle)figure;

    double focus = triangle._length;

    PrintfFormat<FSharpFunc<double, Unit>, TextWriter, Unit, Unit> format = new
PrintfFormat<FSharpFunc<double, Unit>, TextWriter, Unit, Unit, double>("%A");

    PrintfModule.PrintFormatLineToTextWriter(Console.Out, format).Invoke(focus);

    break;

}

case 3:

{

    GeometricFigure.Ellipse ellipse = (GeometricFigure.Ellipse)figure;

    double focus = ellipse._focus;

```

```

        PrintfFormat<FSharpFunc<double, Unit>, TextWriter, Unit, Unit> format = new
PrintfFormat<FSharpFunc<double, Unit>, TextWriter, Unit, Unit, double>("%A");

        PrintfModule.PrintFormatLineToTextWriter(Console.Out, format).Invoke(focus);

        break;
    }

}

}

```

Теперь про остальную часть кода:

[CompilerGenerated]

```

internal static int f@36-1(int x, int y)
{
    int num = x + y;

    logger.log(num);

    return num;
}

```

[CompilationMapping(SourceConstructFlags.Value)]

```

public static Logger logger
{
    get
    {
        return $MyProgram.logger@41;
    }
}

```



```
[CompilationMapping(SourceConstructFlags.Value)]
```

```
public static int loggedWork
```

```
{
```

```
    get
```

```
    {
```

```
        return $MyProgram.loggedWork@42;
```

```
    }
```

```
}
```

```
[SpecialName]
```

```
[CompilationArgumentCounts(new int[] { 1, 1 })]
```

```
public static b op_PipeRight<a, b>(a x, FSharpFunc<a, b> f)
```

```
{
```

```
    return f.Invoke(x);
```

```
}
```

Теперь про Scala и Java

Исходный код на Scala

//Первая особенность - именованные аргументы

```
object Task2S {
```

```
    def printName(first: String, last: String): Unit = {
```

```
        println(first + " " + last)
```

```
    }
```

```
    printName("John", "Smith") // Prints "John Smith"
```

```
    printName(first = "John", last = "Smith") // Prints "John
```

```
Smith"
```

```
    printName(last = "Smith", first = "John") // Prints "John  
Smith"
```

//вторая особенность - трейты

```
trait Iterator[A] {  
    def hasNext: Boolean  
    def next(): A  
}
```

```
class IntIterator(to: Int) extends Iterator[Int] {  
    private var current = 0  
    override def hasNext: Boolean = current < to  
    override def next(): Int = {  
        if (hasNext) {  
            val t = current  
            current += 1  
            t  
        } else 0  
    }  
}
```

```
def printIterator(): Unit = {  
    val iterator = new IntIterator(10)  
    println(iterator.next())  
    println(iterator.next())  
    println(iterator.next())  
    println(iterator.next())  
}
```

```
}
```

<https://pastebin.com/G53ywkyr>

Полный код декомпилированной программы на Scala в Java

Трейты (Traits) используются, чтобы обмениваться между классами информацией о структуре и полях. Они похожи на интерфейсы из Java 8. Классы и объекты могут расширять трейты, но трейты не могут быть созданы и поэтому не имеют параметров.

Это функция с именованными аргументами

```
public void printName(final String first, final String last) {  
    .MODULE$.println((new StringBuilder(1)).append(first).append(" ").append(last).toString());  
}
```

Сам трейт

```
public static class IntIterator implements Task2S.Iterator {  
    private final int to;  
    private int current;  
  
    private int current() {  
        return this.current;  
    }  
  
    private void current_$eq(final int x$1) {  
        this.current = x$1;  
    }  
  
    public boolean hasNext() {  
        return this.current() < this.to;  
    }  
  
    public int next() {  
        int var10000;  
        if (this.hasNext()) {  
            int t = this.current();
```

```

        this.current_$eq(this.current() + 1);

        var10000 = t;

    } else {

        var10000 = 0;

    }

    return var10000;

}

// $FF: synthetic method
// $FF: bridge method

public Object next() {

    return BoxesRunTime.boxToInteger(this.next());

}

public IntIterator(final int to) {

    this.to = to;

    this.current = 0;

}

}

```

Небольшой вывод: функциональные языки отлично подходят для математиков, потому что работа с функциями очень удобна, также очень существенно сокращают размер кода.

Задание 3

C#+F#

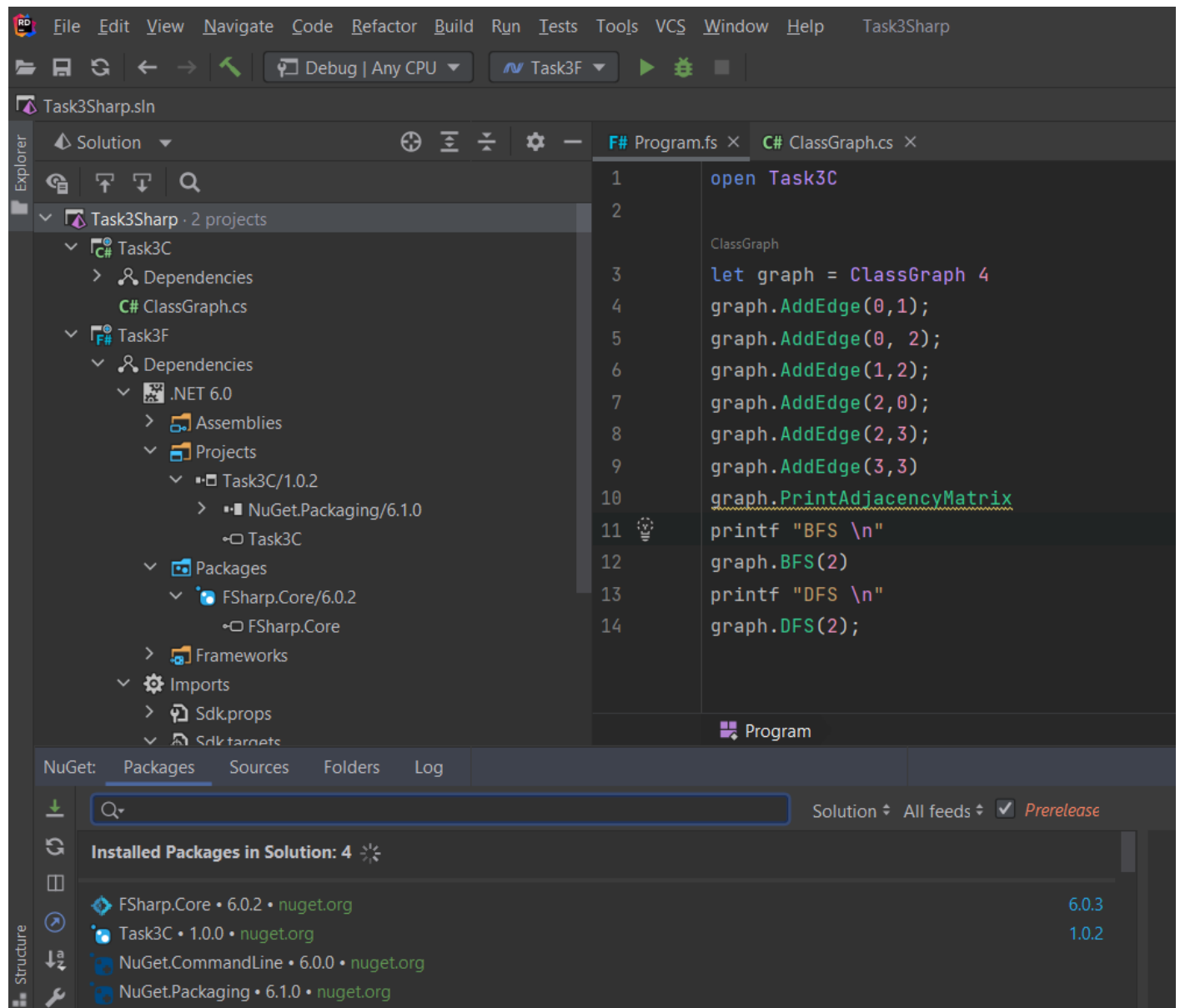
Ссылка на мой nuget пакет

Скачивайте =)

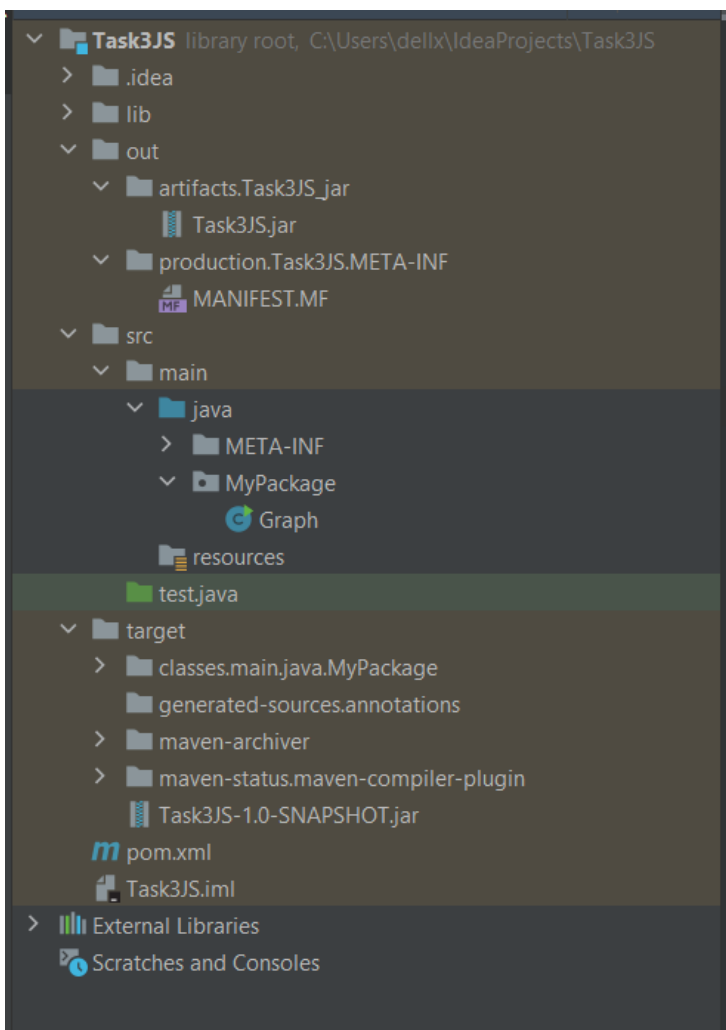
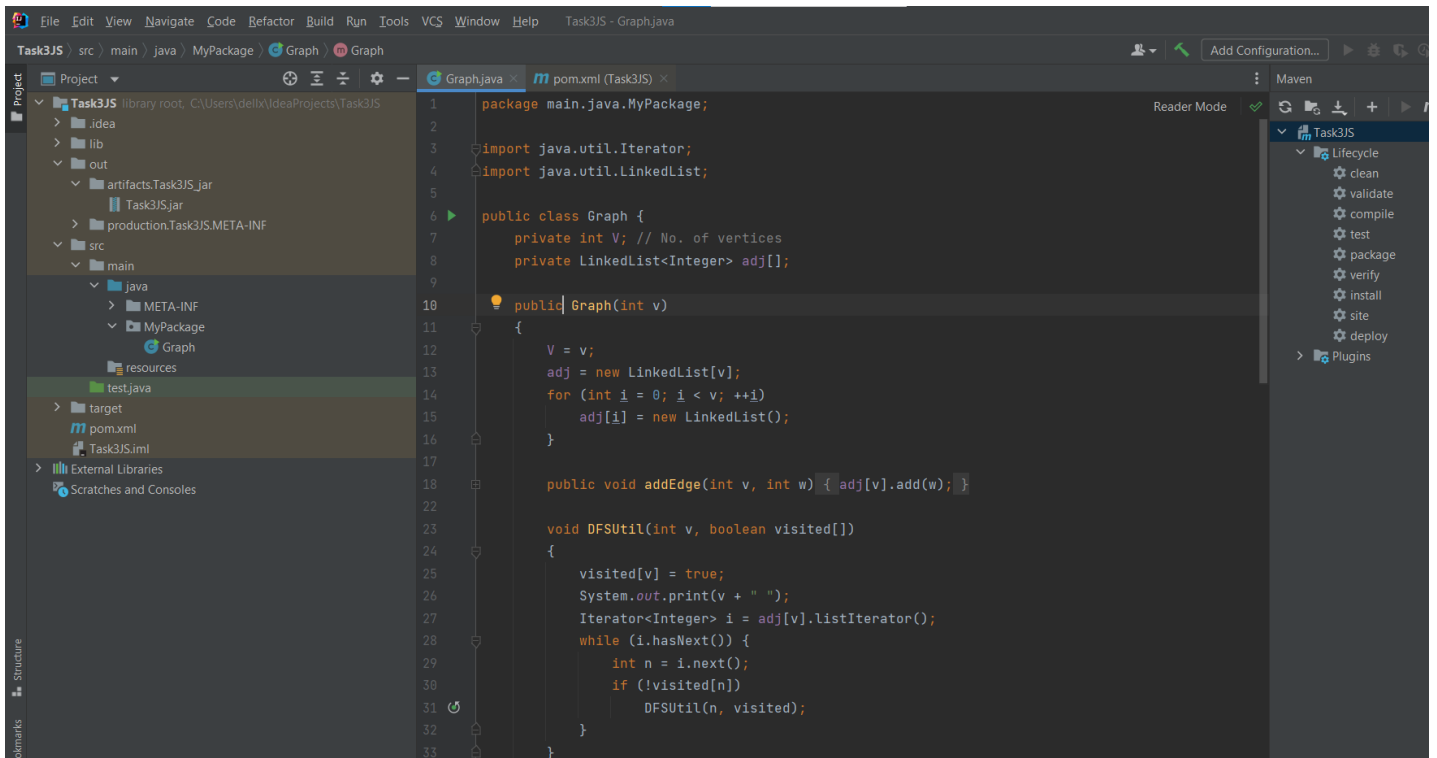
<https://www.nuget.org/packages/Task3C/>

Использован в F# проекте

Подключен через nuget и импортирован в проект

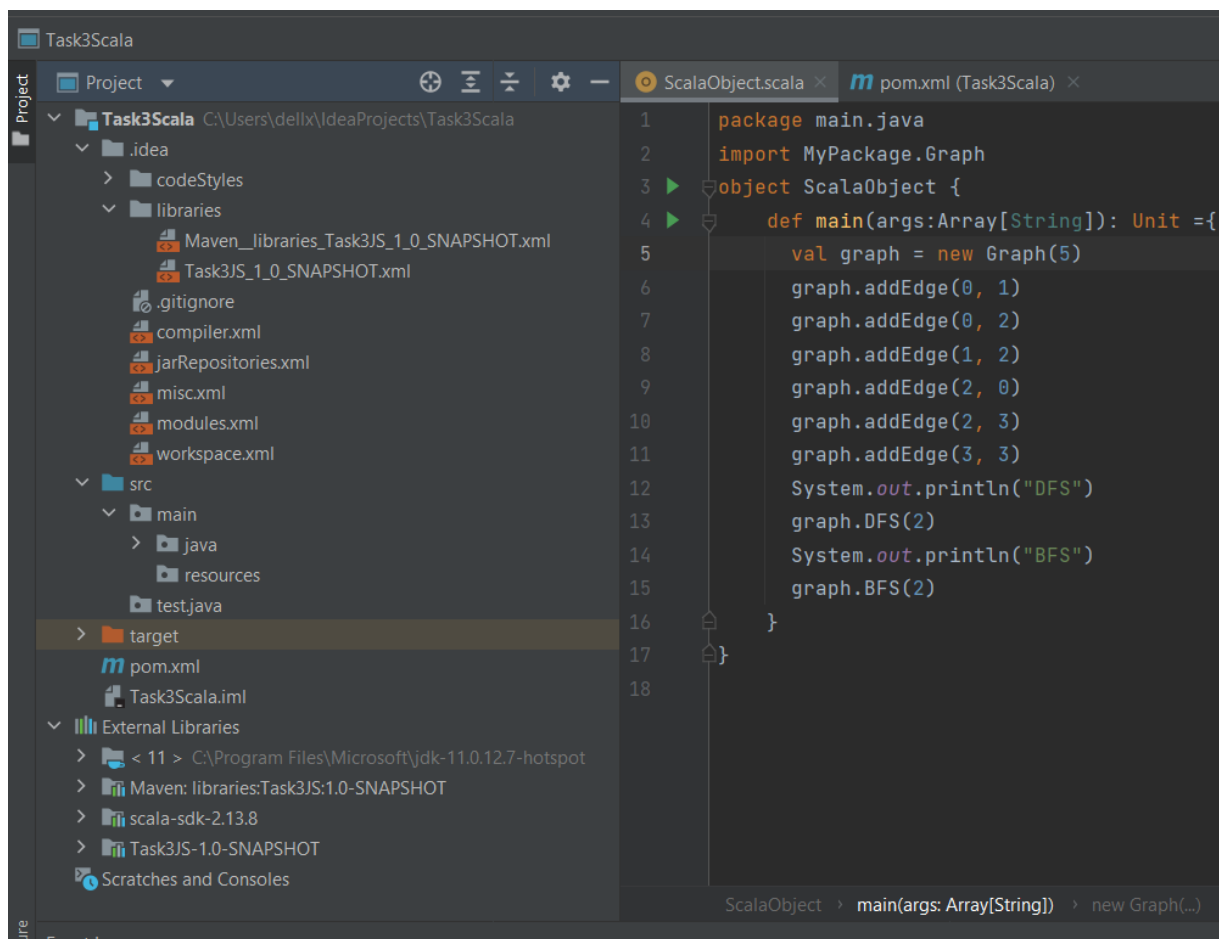


JAVA+SCALA



ТТут с пакетом было не все так просто, поэтому сначала я создал обычный пакет Task3JS.jar, а потом с боями создал Task3JS-1.0-SNAPSHOT.jar

Который в дальнейшем был подключен к данному проекту на Скале.



В основе принципов работы пакетных менеджеров лежит идея формирования некоего промежуточного файла между, который будет содержать в себе всю необходимую для дальнейшего использования информацию о содержимом пакета (Этот может быть как .nupkg, так и .jar, зависит лишь от того, с чем мы имеем дело - C# или Java.

JAR-файл — это Java-архив (Java ARchive). Это простой архивный файл, сжатый (иногда с нулевой компрессией) по алгоритму zip.

Он был создан для удобства распространения программ, написанных на Java. Так как обычная программа содержит сотни, тысячи, а иногда и миллионы файлов. Файл может содержать:

- 1) файл манифеста META-INF/MANIFEST.MF
- 2) java-файлы (исходный код)
- 3) class-файлы
- 4) файлы, необходимые для работы программы: картинки, файлы с настройками и

Манифест - это текстовый файл формата ключ: значение; он содержит описание jar-файла. В нем могут быть следующие ключи:

- 1) Manifest-Version - версия манифеста
- 2) Main-Class - имя главного класса (должен содержать метод main), такой jar-файл можно запустить как обычный исполняемый файл
- 3) Class-Path - позволяет указать CLASSPATH, который необходим для полноценной работы программы

NUPKG - это архивный файл пакета кода, используемый менеджером пакетов NuGet. Он позволяет разработчикам обмениваться многократно используемым кодом.

Структура файла:

Сам файл представляет собой архив ZIP, который содержит скомпилированный код, другие файлы, связанные с этим кодом, и описательный манифест, включающий такую информацию, как номер версии пакета.

Задание 4

Бенчмарк — это измерение или набор измерений, относящихся к выполнению некоторого кода. Контрольные показатели позволяют сравнивать относительную производительность кода по мере того, как вы начинаете прилагать усилия для повышения производительности.

Инструменты бенчмаркинга

- 1) библиотека BenchmarkDotNet
- 2) JMH

BenchmarkDotNet - это легкая, мощная библиотека .NET с открытым исходным кодом, которая может преобразовывать ваши методы в тесты производительности, отслеживать эти методы, а затем предоставлять понимание собранных данных о производительности.

Чтобы запустить BenchmarkDotNet в приложении .NET Framework или .NET Core, необходимо:

- 1) Добавить необходимый пакет NuGet
- 2) Добавить атрибуты Benchmark в методы
- 3) Создать экземпляр BenchmarkRunner
- 4) Запустить приложение в режиме Release

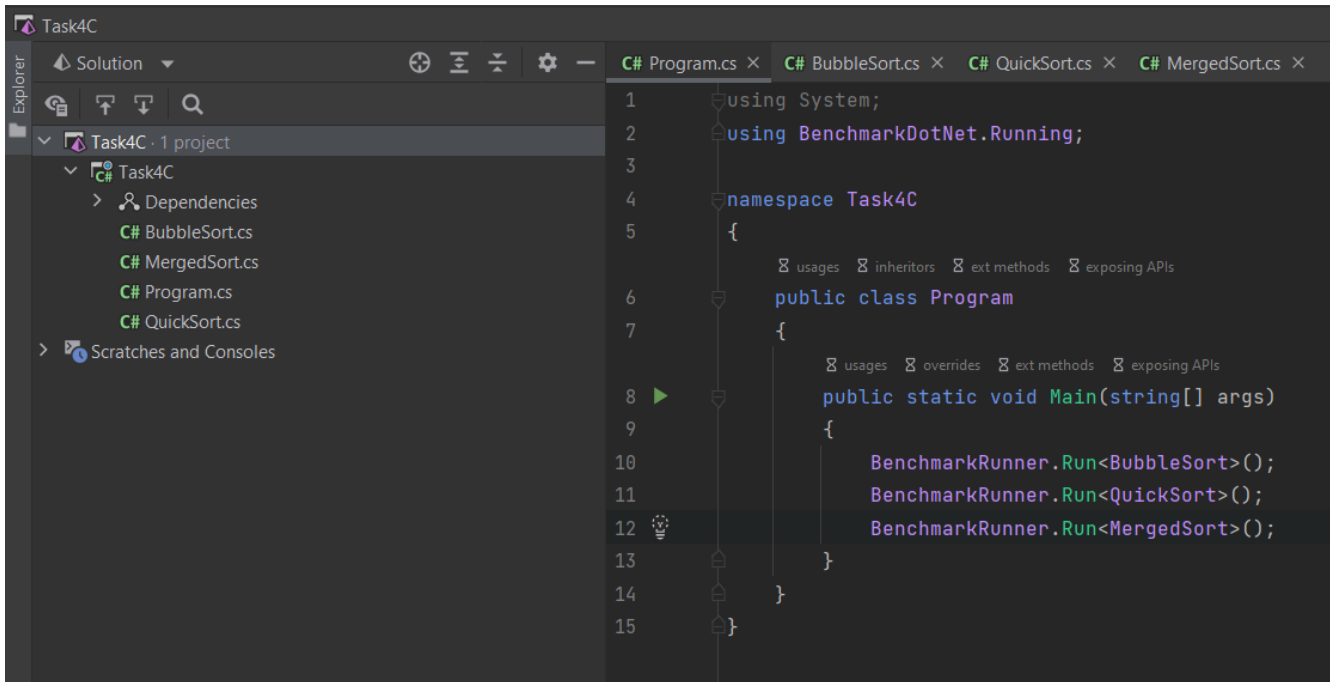
Конфигурирование бенчмарков осуществляется с помощью атрибута Config.

Возможности: настройки окружения\платформы, количество запусков, настройки вывода, логгеров, анализаторы...

Самый простой вариант настройки: вешаем атрибут Config на класс, содержащий Benchmark-методы, и в конструкторе передаем строку с настройками.

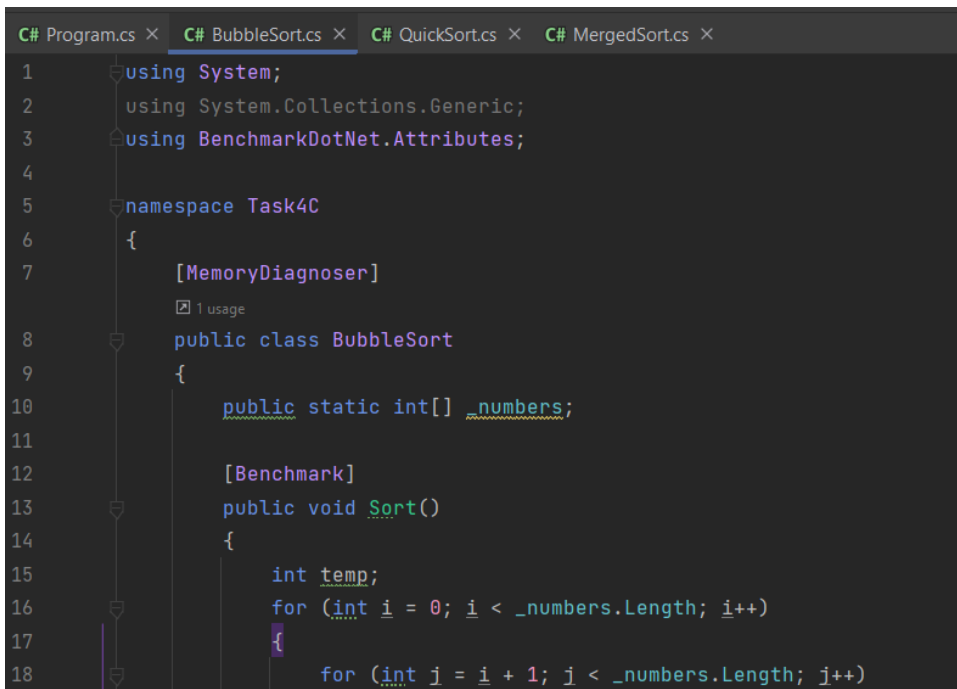
Некоторые фичи

- 1) Параметризованные тесты (можно использовать атрибут Params.)
- 2) Относительное время запуска (Предположим, мы желаем узнать не только абсолютные времена тестовых методов, но и относительные. Для этого выберем метод, время которого считаем "нормой", и изменяем его Benchmark атрибут, установив BaseLine = true.)



The screenshot shows the Visual Studio IDE with the Task4C project. The Explorer pane on the left shows the project structure: Task4C (1 project) containing Dependencies, C# BubbleSort.cs, C# MergedSort.cs, C# Program.cs, and C# QuickSort.cs. The Solution Explorer shows the same structure. The main editor displays the Program.cs file with the following code:

```
1  using System;
2  using BenchmarkDotNet.Running;
3
4  namespace Task4C
5  {
6      public class Program
7      {
8          public static void Main(string[] args)
9          {
10             BenchmarkRunner.Run<BubbleSort>();
11             BenchmarkRunner.Run<QuickSort>();
12             BenchmarkRunner.Run<MergedSort>();
13         }
14     }
15 }
```



The screenshot shows the Visual Studio IDE with the Task4C project. The main editor displays the BubbleSort.cs file with the following code:

```
1  using System;
2  using System.Collections.Generic;
3  using BenchmarkDotNet.Attributes;
4
5  namespace Task4C
6  {
7      [MemoryDiagnoser]
8      public class BubbleSort
9      {
10         public static int[] _numbers;
11
12         [Benchmark]
13         public void Sort()
14         {
15             int temp;
16             for (int i = 0; i < _numbers.Length; i++)
17             {
18                 for (int j = i + 1; j < _numbers.Length; j++)
```

```

using System;
using System.Collections.Generic;
using BenchmarkDotNet.Attributes;

namespace Task4C
{
    [MemoryDiagnoser]
    public class QuickSort
    {
        private int[] _numbers;

        int Partition(int[] array, int start, int end)
        {
            int marker = start;
            for (int i = start; i <= end; i++)
            {
                if (array[i] <= array[end])
                {
                    (array[marker], array[i]) = (array[i], array[marker]);
                    marker++;
                }
            }
            return marker - 1;
        }
    }
}

```

```

private int[] MergeSort(int[] array, int lowIndex, int highIndex)
{
    if (lowIndex < highIndex)
    {
        var middleIndex = (lowIndex + highIndex) / 2;
        MergeSort(array, lowIndex, middleIndex);
        MergeSort(array, middleIndex + 1, highIndex);
        Merge(array, lowIndex, middleIndex, highIndex);
    }

    return array;
}

[Benchmark]
public void MergeSort()
{
    MergeSort(_numbers, 0, _numbers.Length - 1);
}

```

<https://pastebin.com/SeHQe8fL>

<https://pastebin.com/Zioswtv2>

<https://pastebin.com/w1qF3fxX>

Сверху все сортировки с навешенными атрибутами

Результаты:

BubbleSort

Method	Mean	Error	StdDev	Allocated
Sort	7.518 ms	0.4533 ms	1.337 ms	384 B

QuikSort

Method	Mean	Error	StdDev	Allocated
ArraySort	162.8 us	6.05 us	17.46 us	384 B

MergeSort

Method	Mean	Error	StdDev	Allocated
MergeSort	237.0 us	13.15 us	38.76 us	134 KB

Среднее - Погрешность - Среднеквадратическое отклонение - память

Тогда

BubbleSort – 7518 us, 384B

QuikSort – 162.8 us, 384B

MergeSort – 237 us, 137216B

Вывод: quiksort быстрее и тратит меньше памяти, что в целом предсказуемо, исходя из знаний, полученных на предмете “Алгоритмы и структуры данных”

2)JMH

Java Microbenchmark Harness — набор библиотек для тестирования производительности небольших функций (то есть тех, где пауза GC увеличивает время работы в разы).

Перед запуском теста JMH перекомпилирует код, так как:

- 1) Для уменьшения погрешности вычисления времени работы функции необходимо запустить её N раз, подсчитать общее время работы, а потом поделить его на N.
- 2) Для этого требуется обернуть запуск в виде цикла и вызова необходимого метода. Однако в этом случае на время работы функции повлияет сам цикл, а также сам вызов замеряемой функции. А потому вместо цикла будет вставлен непосредственно код вызова функции, без reflection или генерации методов в runtime.

JMH использует fork java процесса. (системный вызов)

В случае Windows это сделать так просто нельзя, а потом новый процесс просто запускается с тем же classpath. И весь список jar файлов передается через командную строку, размер которой ограничен. В итоге, если GRADLE_USER_HOME (папка, внутри которой лежит в том числе кеш gradle) находится в глубине файловой структуры, список jar файлов для fork становится настолько большим, что Windows отказывается запускать процесс с таким громадным число аргументов командной строки. Следовательно, если JMH отказывается делать fork — просто переместите кеши Gradle в папку с коротким именем, т.е. запишите в environment variable GRADLE_USER_HOME что-то вроде c:\gradle (ТО ЧТО Я СНАЧАЛА НЕ СДЕЛАЛ И ЕМУ БЫЛО ПЛОХО)

Иногда предыдущий процесс JMH делает lock на файле (возможно, это делает byte code rewrite). В итоге, повторная компиляция может не работать, так как файл с нашим benchmark открыт кем-то на запись.

Все сортировки:

<https://pastebin.com/QzG7euPE>

<https://pastebin.com/makHQ2vV>

<https://pastebin.com/iGZS1qER>

```
package MyPackage;

import org.openjdk.jmh.annotations.*;
import org.openjdk.jmh.runner.Runner;
import org.openjdk.jmh.runner.RunnerException;
import org.openjdk.jmh.runner.options.Options;
import org.openjdk.jmh.runner.options.OptionsBuilder;

import java.util.concurrent.TimeUnit;
@State(Scope.Benchmark)
public class Main {

    @Benchmark
    @BenchmarkMode(Mode.SampleTime)
    @Fork(warmups = 0, value = 1)
    @Measurement(iterations = 10)
    @OutputTimeUnit(TimeUnit.MICROSECONDS)
    public void MergeSort(){
        var ms = new MergeSort();
        ms.ArraySort();
    }
}
```

```
@Benchmark
@BenchmarkMode(Mode.SampleTime)
@Fork(warmups = 0, value = 1)
@Measurement(iterations = 10)
@OutputTimeUnit(TimeUnit.MICROSECONDS)
public void QuickSort(){
    var qs = new QuickSort();
    qs.ArraySort();
}
```

```

@Benchmark
@BenchmarkMode(Mode.SampleTime)
@Fork(warmups = 0, value = 1)
@Measurement(iterations = 10)
@OutputTimeUnit(TimeUnit.MICROSECONDS)
public void BubbleSort(){
    var bs = new BubbleSort();
    bs.ArraySort();
}

```

```

public static void main(String[] args) throws RunnerException {
    Options opt = new OptionsBuilder()
        .include(Main.class.getSimpleName())
        .forks(1)
        .build();
    new Runner(opt).run();
}

```

MyPackage.Main.BubbleSort sample 324937 307,673 ± 0,370 us/op

MyPackage.Main.MergeSort sample 1373962 36,586 ± 0,074 us/op

MyPackage.Main.QuickSort sample 2249088 11,201 ± 0,029 us/op

Тут также получился ожидаемый результат: Quick самый эффективный, а пузырька не было шансов...

Задание 5

Пришлось немного доработать Бекапы: сделать еще одну реализацию IRepository и еще одну реализацию IAlgorithm, чтобы все-таки оторвать их от файловой системы окончательно.

MockRepo.cs

```
public class MockRepo : IRepository
{
    [1 usage]
    public MockRepo()
    {
        RestorePoints = new List<RestorePoint>();
    }

    [0+5 usages]
    public string Path { get; set; }

    [2+17 usages]
    public List<RestorePoint> RestorePoints { get; }

    [1+1 usages]
    public int NumberOfRestorePoints { get; set; }

    [0+1 usages]
    public void SavePoint(RestorePoint restorePoint)
    {
        RestorePoints.Add(restorePoint);
        NumberOfRestorePoints++;
    }
}
```

Сохранение точек восстановления происходит только в List<RestorePoint>

TestSingleStorageAlgo.cs

```
using System.Collections.Generic;
using System.IO;

namespace Backups
{
    [1 usage]
    public class TestSingleStorageAlgo : IAlgorithm
    {
        [0+1 usages]
        public RestorePoint MakePoint(List<JobObject> listJobObjects)
        {
            RestorePoint a = new RestorePoint( name: "a");
            foreach (var job in listJobObjects)
            {
                a.AddFile(new FileInfo(job.Path));
            }

            return a;
        }
    }
}
```

После того, как мы наконец-то отвязали бекапы от файловой системы, запускаем два раза создание 500 точек и запускаем dotTrace и dotMemory.

<https://pastebin.com/tGqQj2uN>

Тут полный код Main().

```
BackupService a = new BackupService( algorithm: new TestSingleStorageAlgo(), repository: new MockRepo());
JobObject x = new JobObject( path: "dima");
a.AddJobObject(x);
for (int i = 0; i < 500; i++)
{
    a.MakePoint();
}
```

```

Directory.CreateDirectory(

Directory.GetParent(Environment.CurrentDirectory)?.Parent?.Parent?.Parent?.FullName +
    "/Backups/WorkFiles/");
FileStream fileStream1 = File.Create(

Directory.GetParent(Environment.CurrentDirectory)?.Parent?.Parent?.Parent?.FullName +
    "/Backups/WorkFiles/1.txt");

fileStream1.Close();

Directory.CreateDirectory(

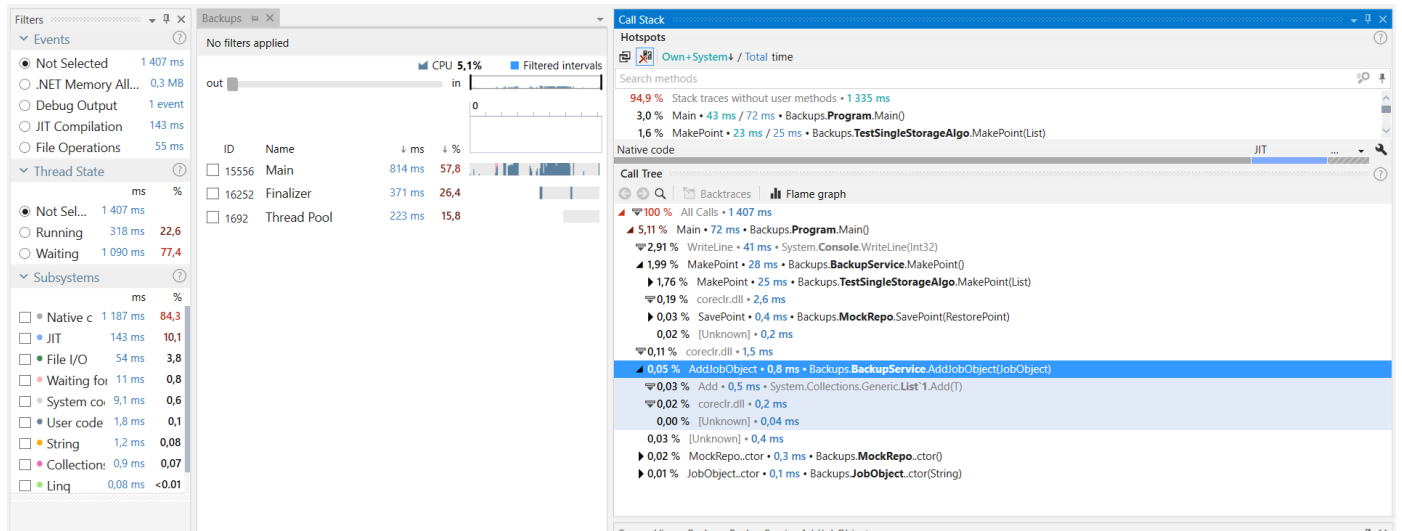
Directory.GetParent(Environment.CurrentDirectory)?.Parent?.Parent?.Parent?.FullName +
    "/Backups/BackupWorkFiles/");
BackupService a = new BackupService(new SplitStoragesAlgo(), new
Repository(Directory.GetParent(Environment.CurrentDirectory)?.Parent?.Parent?.Parent?.
FullName + "/Backups/BackupWorkFiles/"));
JobObject x = new
JobObject(Directory.GetParent(Environment.CurrentDirectory)?.Parent?.Parent?.Parent?.F
ullName + "/Backups/WorkFiles/1.txt");
a.AddJobObject(x);
for (int i = 0; i < 500; i++)
{
    a.MakePoint();
}

Console.WriteLine(a.Repository.NumberOfRestorePoints);

```

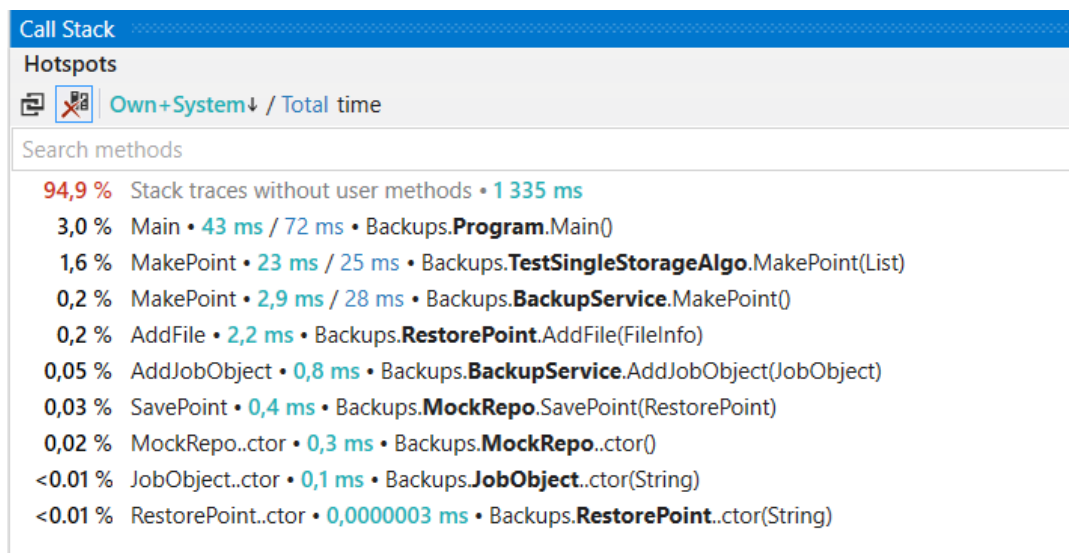
Без файловой системы

Сначала запустили Timeline



Так как сверху скорее всего ничего не видно, снизу все прикрепляю

Стек вызова

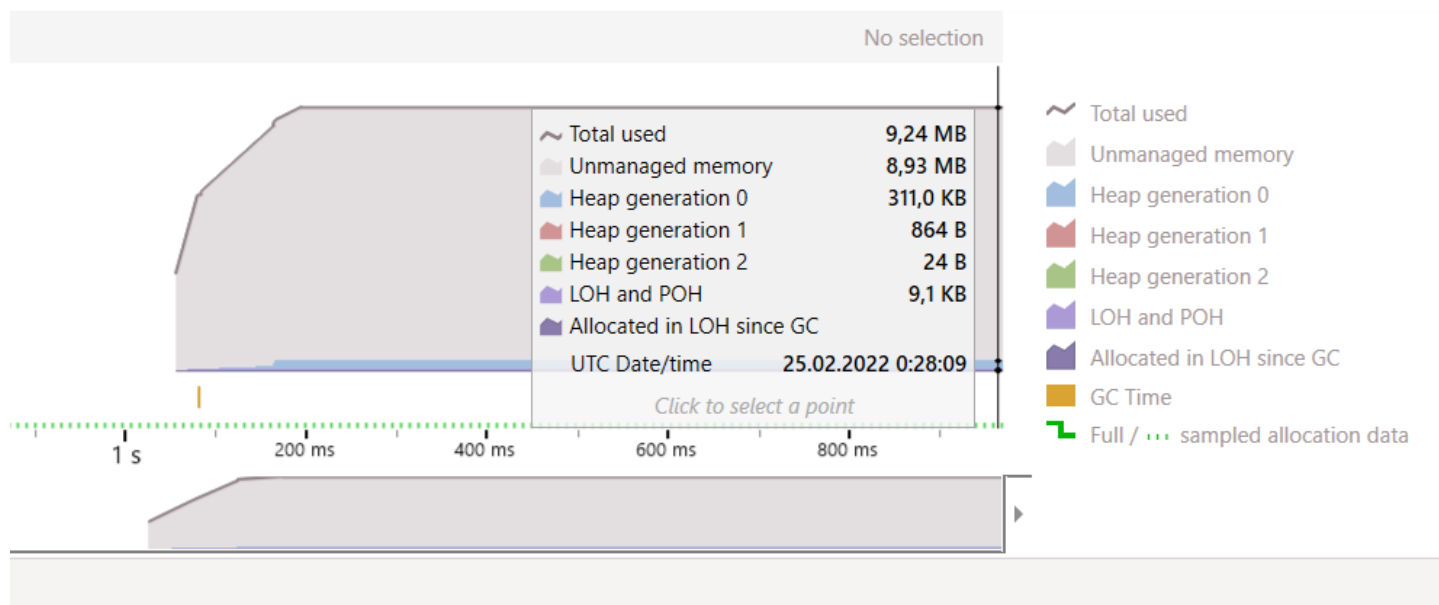


Теперь запускаем Трассировку

Subsystems		
50% System code		
System	System.Diagnostics.Tracing	
Subsystem	Payload	
▸ System code	49,53%	46 ms
▸ String	17,44%	16 ms
▸ Collections	12,93%	12 ms
▸ File I/O	11,14%	10 ms
▾ User code	8,69%	8,0 ms
Backups.TestSingleStorageAlgo.MakePoint(List)	2,79%	2,6 ms
Backups.RestorePoint.AddFile(FileInfo)	2,24%	2,1 ms
Backups.BackupService.MakePoint	2,08%	1,9 ms
Backups.Program.Main	0,98%	0,9 ms
Backups.MockRepo.SavePoint(RestorePoint)	0,20%	0,2 ms
Backups.BackupService.AddJobObject(JobObject)	0,13%	0,1 ms
Backups.MockRepo..ctor	0,08%	0,07 ms
Backups.RestorePoint..ctor(String)	0,07%	0,06 ms
Backups.JobObject..ctor(String)	0,05%	0,04 ms
Backups.RestorePoint+<>c__DisplayClass5_0..ctor	0,02%	0,02 ms
▸ Linq	0,27%	0,3 ms

И получаем процентное соотношение времени выполнения конкретного метода к общему времени выполнения.

Теперь запускаем dotMemory



Сборщик мусора в .NET является generational, т.е. управляемая куча (соответственно и объекты) делится на поколения. Все объекты делятся по жизненному циклу на несколько поколений.

1) Generation 0.

Жизненный цикл объектов этого поколения самый короткий. Обычно к Gen0 относятся временные переменные, созданные в теле методов.

2) Generation 1.

Жизненный цикл объектов этого поколения также короткий. К нему относятся объекты с промежуточным временем жизни – объекты, переходящие из Gen0 в Gen2.

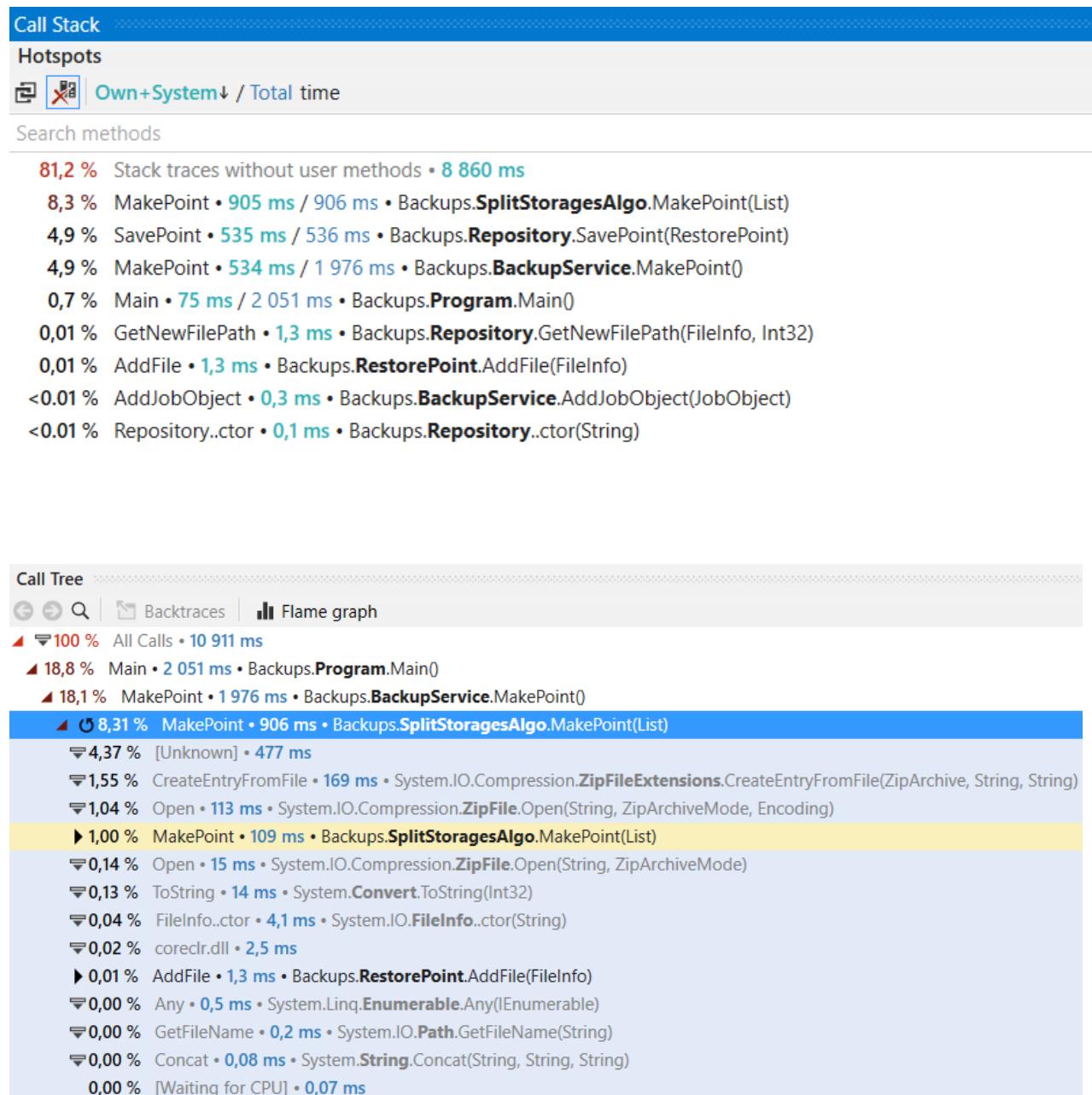
3) Generation 2.

Представляет собой наиболее долгоживущие объекты. Также объекты размером более 85 000 байт автоматически попадают в Large Object Heap и помечаются как Gen2.

Для файловой системы

Timeline

Стек вызова



- ▲ 4,91 % SavePoint • 536 ms • Backups.**Repository**.SavePoint(RestorePoint)
- ☹ 1,99 % MoveFile • 217 ms • System.IO.**FileSystem**.MoveFile(String, String, Boolean)
- ☹ 1,82 % CreateDirectory • 198 ms • System.IO.**Directory**.CreateDirectory(String)
- ☹ 0,53 % [Unknown] • 58 ms
- ☹ 0,36 % Move • 39 ms • System.IO.**File**.Move(String, String)
- ☹ 0,20 % Move • 21 ms • System.IO.**File**.Move(String, String, Boolean)
- ▶ 0,01 % GetNewFilePath • 1,3 ms • Backups.**Repository**.GetNewFilePath(FileInfo, Int32)
- ☹ 0,01 % coreclr.dll • 0,9 ms
- ☹ 0,00 % ToString • 0,3 ms • System.**Convert**.ToString(Object)
- ☹ 0,00 % Concat • 0,01 ms • System.**String**.Concat(String, String, String)
- ☹ 4,81 % [Unknown] • 525 ms
- ☹ 0,08 % coreclr.dll • 8,9 ms
- ☹ 0,18 % GetParent • 20 ms • System.IO.**Directory**.GetParent(String)
- ☹ 0,14 % Create • 15 ms • System.IO.**File**.Create(String)
- ☹ 0,13 % WriteLine • 14 ms • System.**Console**.WriteLine(Int32)
- ☹ 0,11 % CreateDirectory • 11 ms • System.IO.**Directory**.CreateDirectory(String)
- ☹ 0,06 % get_CurrentDirectory • 6,8 ms • System.**Environment**.get_CurrentDirectory()
- ☹ 0,03 % coreclr.dll • 3,6 ms

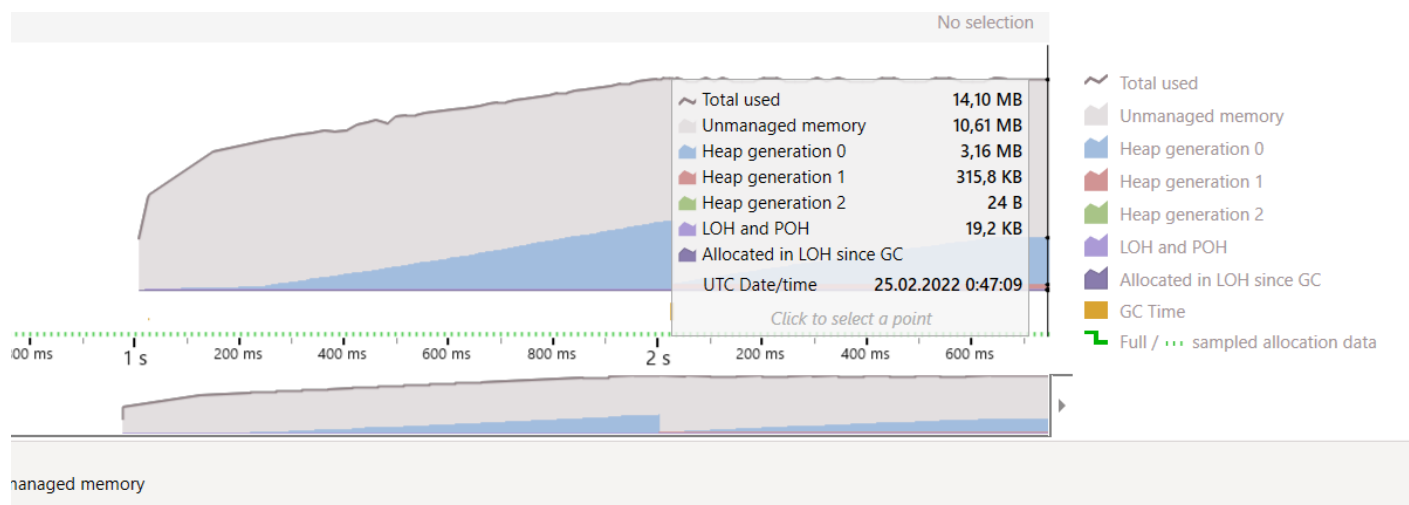
▼ Events ?	
<input checked="" type="radio"/> Not Selected	10 911 ms
<input type="radio"/> .NET Memory Allocations	7,8 MB
<input type="radio"/> Debug Output	1 event
<input type="radio"/> Garbage Collection	1,9 ms
<input type="radio"/> JIT Compilation	441 ms
<input type="radio"/> File Operations	322 ms

10,5 % CPU

Трассировка

Subsystem	Payload
System code	87,83% 1 408 ms
Interop+Kernel32.MoveFileExPrivate	35,46% 569 ms
Other	18,02% 289 ms
Interop+Kernel32.CreateFilePrivate	13,67% 219 ms
Interop+Kernel32.GetFileAttributesExPrivate	10,35% 166 ms
Interop+Kernel32.CreateDirectoryPrivate	10,34% 166 ms
File I/O	7,54% 121 ms
String	2,19% 35 ms
Collections	1,18% 19 ms
User code	1,12% 18 ms
Backups.BackupService.MakePoint	0,39% 6,2 ms
Backups.SplitStoragesAlgo.MakePoint(List)	0,18% 3,0 ms
Backups.Program.Main	0,14% 2,3 ms
Backups.Repository.GetNewFilePath(FileInfo, Int32)	0,14% 2,2 ms
Backups.Repository.SavePoint(RestorePoint)	0,13% 2,0 ms
Backups.RestorePoint.AddFile(FileInfo)	0,07% 1,1 ms
Backups.RestorePoint..ctor(String)	0,02% 0,4 ms
Backups.JobObject.get_Path	0,02% 0,3 ms
Backups.RestorePoint.get_Files	<0,01% 0,1 ms
Backups.BackupService.AddJobObject(JobObject)	<0,01% 0,1 ms
GC Wait	0,07% 1,1 ms
Linq	0,07% 1,1 ms

DotMemory



Анализ полученных результатов

Параметр	Без фс	С фс	Комментарий
Общая память	9.24MB	14.10 MB	Понятно, что для хранения 500 zip-архивов уходит достаточное количество памяти
Heap0	311KB	3.16MB	В целом, опять же понятно, объекты стали весить больше, соответственно, некоторые стали попадать в более высокие поколения
Heap1	864B	3.115KB	
Heap2	24B	24B	

При переходе к файловой системе выросло общее время выполнения программы с 1407ms до 10911ms, что соответствует росту в 7.8 раз или же на 675,5%.

За счет чего получилось такое увеличение во времени работы?

1) MakePoint() выросло с 25ms (1.99% от общего времени) до 1976ms (18.1%), те выросло в 79 раз.

2) Main() выросло с 72ms (5.11%) до 2051ms (18,8%)

3) Исходя из данных, полученных в результате трассировки можно сделать вывод о том, что среднее время выполнения метода увеличилось почти в 10 раз.

В результате выполнения данного задания было выяснено, что данная реализация не является эффективной и требует дальнейших доработок или переработок.