dstruct Point: IComparable<Point>, ICloneable {

public double x, y;

public override string ToString() {

return "Point";

}

public int CompareTo(Point other) {

return 1;

}

public object Clone() {

return this;

}

}

class Program {

static void Main(string[] args)

{

Point p = default(Point);

((IComparable<Point>) p).CompareTo(p);

p.ToString();

p.GetHashCode();

p.Clone();

Console.ReadKey();

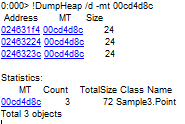
}

}

При вызове команды !DumpHeap –stat найдём строку с нашим типом

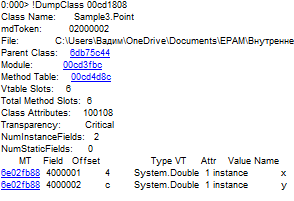


Выведем подробную информацию об объектах нашего типа

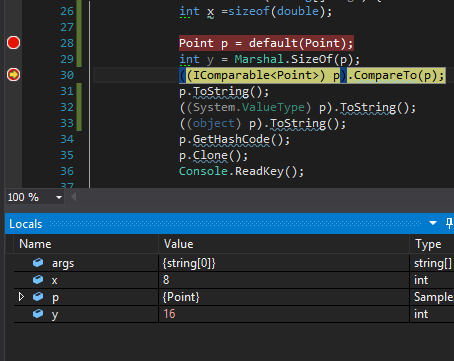
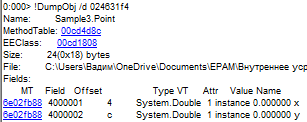


Создание 3 объектов для данного куска кода, говорит нам о том, что значимые типы копируются при передаче в качестве параметров методов, возврате методов.

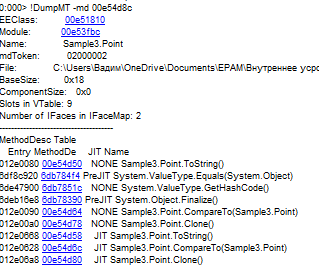
Просмотрим информацию о нашем классе



Просмотрим один из созданных объектов. Как видим размер объекта составляет 24 байта (2 поля по 8 байт равно 16 байтам, откуда берутся ещё 8 байт непонятно ). Во время выполнения кода размер объекта составил 16 байт



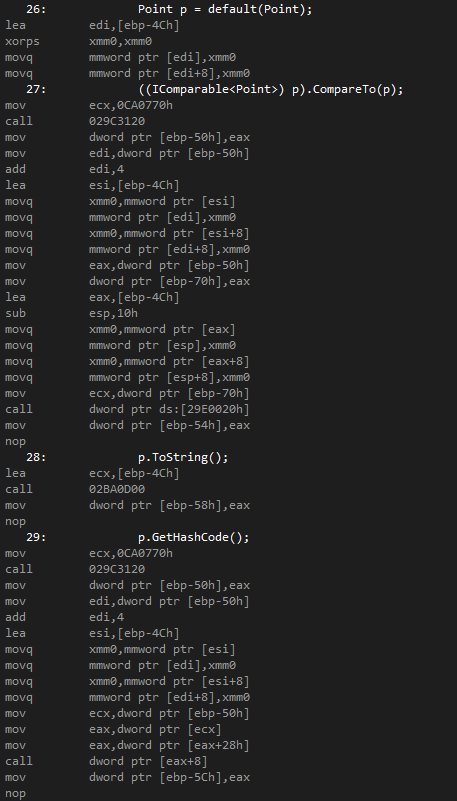
Просмотрим таблицу методов типа Point. Как мы можем заметить в таблице методов присутствуют методы типа ValueType и метод типа System (Finalize). Метод GetHashCode не предкомпилирован. Взглянув в Reflector можно заменить, что данный метод помечен как extern, поэтому данный метод и не компилировался JIT компилятором.



Просмотрим домены нашего приложения командой !DumpDomain



Исследуем данный код с помощь дизассемблера в среде Visual Studio



Вызов ToString и GetHashCode на объекте типа Point существенно отличаются. Как можно заметить, метод ToString вызывается как обычный метод, а не виртуальный, а для метода GetHashCode сгенерировался для динамической диспетчеризации. Также для объектов значимых типов первой командой является команда lea.