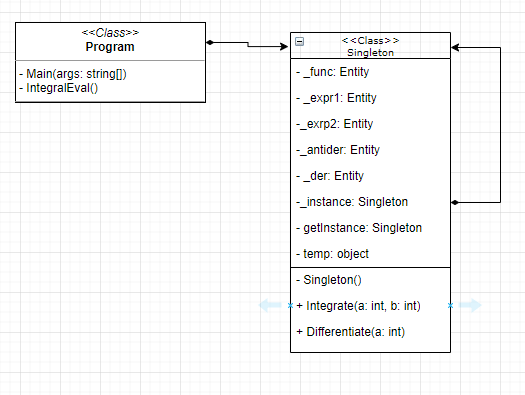
Реализация паттерна Singleton.

В данной программе реализован паттерн Singleton в его безопасной для многопоточности версии. Данная программа в разных потоках (parent, child) производит математические вычисления (интеграл и производная).

Диграмма классов:



Программный код:

using System;  
using System.Threading;  
using AngouriMath;  
  
namespace Singleton  
{  
 class Program  
 {  
   
 static void Main(string[] args)  
 {  
   
 Singleton forDifferentiation = Singleton.getInstance;  
   
  
   
 Thread myThread = new Thread(new ThreadStart(IntegralEval));  
 myThread.Start();  
   
 var rnd = new Random();  
 for (int i = 0; i < 5; i++)  
 {  
 Console.WriteLine("In parent thread");  
 forDifferentiation.Differentiate(rnd.Next(1, 10));  
 Thread.Sleep(400);  
 }  
  
   
 }  
  
 private static void IntegralEval()  
 {  
 var rnd = new Random();  
   
 Singleton forIntegration = Singleton.getInstance;  
 for (int i = 0; i < 5; i++)  
 {  
 Console.WriteLine("In child thread");  
 forIntegration.Integrate(rnd.Next(1, 10), rnd.Next(1, 10));  
 Thread.Sleep(300);  
 }  
  
 }  
 }

public sealed class Singleton  
{  
 private static Entity \_func;  
 private static Singleton \_instance = null;  
   
 private static Entity \_expr1;  
 private static Entity \_expr2;  
   
 private static Entity \_antider;  
 private static Entity \_der;  
  
 private static object temp = new Object();  
 public static Singleton getInstance  
 {  
 get  
 {  
 if (\_instance == null)  
 {  
 lock (temp)  
 {  
 if (\_instance == null)  
 {  
 \_instance = new Singleton();   
 }  
 }  
 }  
 return \_instance;  
 }  
 }  
   
 private Singleton()  
 {  
 \_func = "x^2+ln(x)";  
 \_antider = \_func.Integrate("x").InnerSimplified;  
 \_der = \_func.Differentiate("x").Simplify();  
 Console.WriteLine($"Function body: {\_func}\n" +  
 $"Antiderivative: {\_antider}\n" +  
 $"Derivative: {\_der}");  
 }  
  
 public void Integrate(int a, int b)  
 {  
 Console.WriteLine($"Evaluating integral from a={a} to b={b}");  
 double newtonLeibniz = (double)(\_antider.Substitute("x",a).EvalNumerical() -   
 \_antider.Substitute("x", b).EvalNumerical());  
 Console.WriteLine($"Result={Math.Round(newtonLeibniz, 3)}\n");  
 }  
  
 public void Differentiate(int a)  
 {  
 Console.WriteLine($"Evaluating derivative at a={a}");  
 Entity result = \_der.Substitute("x", a).EvalNumerical();  
 Console.WriteLine($"Result={result.ToString()}\n");  
 }  
  
}

}