

# Введение

# **1. Обзор понятий и методов объектно-ориентированного анализа. Основы использования языка UML.**

## **Сравнение технологий структурного и объектно-ориентированного программирования**

## 2. Применение объектно-ориентированного подхода при программировании методов приближенных вычислений

### 2.1. Цель работы

Необходимо создать приложение на языке C#, в котором выполняется приближенное вычисление интеграла определенной функции разными методами численного интегрирования (методы прямоугольников, метод трапеций и метод Симпсона), выводятся графики функций с указанием интервала интегрирования.

Из предоставленного выбора между графическим и консольным интерфейсами приложения для своей работы я выбрал консольный, т.к. на мой взгляд, он более универсальный для использования в различных операционных системах, например, не только в Windows, но и в Unix-подобных.

По моему варианту нужно работать со следующими двумя функциями:

$$f_1 = \frac{8}{5 + 2^{x^2}}$$

$$f_2 = \frac{x^2}{(x + 8)^{\frac{1}{3}}}$$

Вышеописанные функции должны быть проинтегрированы тремя способами: методом средних прямоугольников, методом трапеций и методом Симпсона «3/8». Также функция интегрируется программой аналитическим способом, то есть, заранее заложенной в ней формулой уже найденной первообразной.

Структура программы включает в себя три модуля:

- Модуль классов, содержащий инструментарий для интегрирования функций;
- Модуль, в котором описаны сами интегрируемые функции;
- «Связующий» модуль, предоставляющий пользовательский интерфейс командной строки и взаимодействие с модулями выше.

## 2.2. Разработка приложения

Приложение разрабатывалось на языке С#. Для написания кода программы использовался интерактивный текстовый редактор Visual Studio Code, а для отладки и тестирования было создано решение function-integration в среде разработки MS Visual Studio 2019 на основе шаблона «Консольное приложение (.NET Framework)», куда были добавлены уже написанные модули.

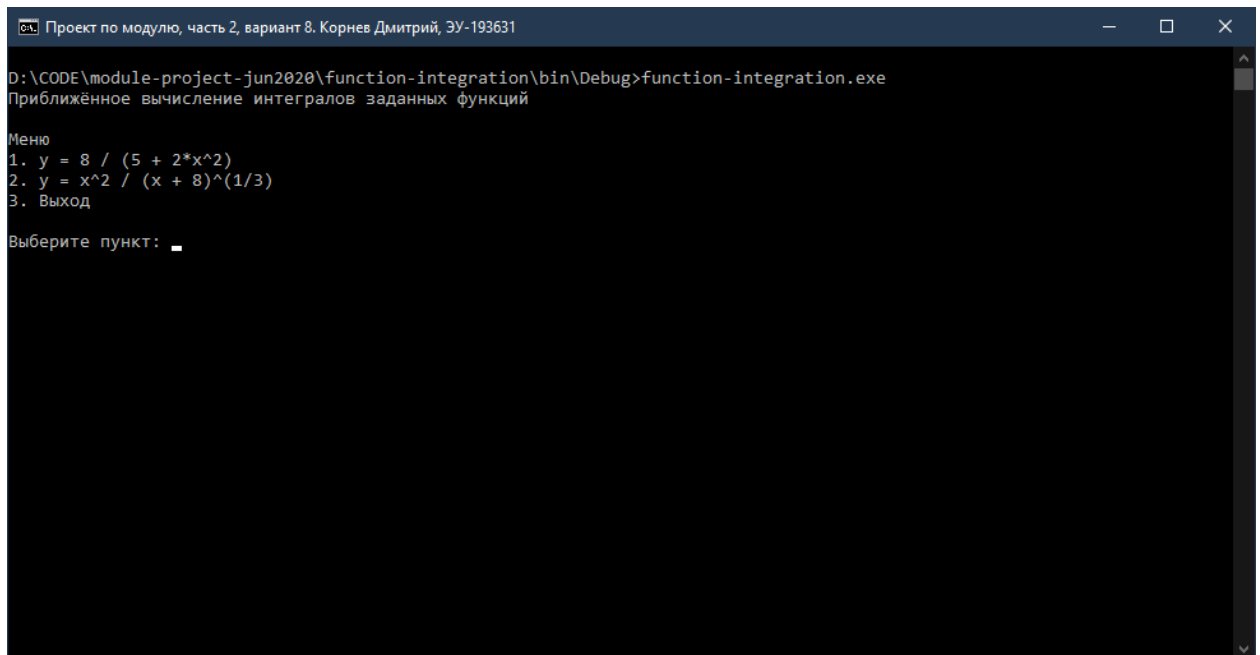


Рисунок 1 - стартовое меню приложения

### Модуль IntegrationMethods.cs

Основополагающий модуль программы, содержащий в себе абстрактный класс IntegrationMethods. Он содержит в себе объявления трёх абстрактных методов, которые при наследовании необходимо переопределить. Они представляют интегрируемую функцию с аргументом  $x$ , её аналитически вычисленную первообразную с аргументами границ интегрирования и удобочитаемое написание (для вывода в консоль):

```
abstract public double Y(double x);  
abstract public double IntY(double low, double hi);  
abstract public string FuncSpelling();
```

Также класс содержит наследуемые методы MidRectMethod, TrapeziaMethod и Simpson2Mehod, которые вычисляют значение интеграла методом средних треугольников, трапеции и Симпсона «3/8» соответственно, и возвращают его. Для расчёта значений использовались формулы с сайта <https://www.cleverstudents.ru>.

Эти методы представлены ниже:

```
public double MidRectMethod(double low, double intStep, double stepCount)
{ // интегрирование методом трапеций
    double result = 0, x = low;

    for (int i = 1; i < stepCount; i++)
    {
        result += Y(x - (intStep / 2)) * intStep;
        x += intStep;
    }

    return result;
}

public double TrapeziaMethod(double low, double intStep, double stepCount)
{ // интегрирование методом трапеций
    double result = 0, x = low;

    for (int i = 1; i < stepCount; i++)
    {
        result += (Y(x) + Y(x - intStep)) / 2 * intStep;
        x += intStep;
    }

    return result;
}

public double Simspon2Method(double low, double intStep, double stepCount)
{ // метод для интегрирования методом Симпсона 3/8
    double result = 0, x = low;

    for (int i = 1; i < stepCount; i++)
    {
        /*
            для удобства числитель разбит на
            три дополнительные переменные
        */
        double a = Y(x - intStep);
        double b = 3 * Y((2 * (x - intStep) + x) / 3);
        double c = 3 * Y((x - intStep + 2 * x) / 3);

        result += (a + b + c + Y(x)) / 8 * intStep;
        x += intStep;
    }

    return result;
}
```

## Модуль FunctionData.cs

Модуль, хранящий в себе описания интегрируемых функций по варианту, их аналитически вычисленные первообразные и строковое представление.

Сами функции хранятся в статическом классе Funcs в виде методов Function1 и Function2:

```
public static class Funcs
{
    public static double Function1(double x)
    {
        return 8 / (5 + 2 * x * x);
    }

    public static double Function2(double x)
    {
        return (x * x) / Math.Pow(x + 8, (double)1 / 3);
    }
}
```

Первообразные функций хранятся в классе FuncInts в таком же виде, только у методов в именах присутствует суффикс Int:

```
public static class FuncInts
{
    public static double FunctionInt1(double x)
    {
        return 8 * Math.Atan(Math.Sqrt(2) * x / Math.Sqrt(5)) / Math.Sqrt(10);
    }

    public static double FunctionInt2(double x)
    {
        return (Math.Pow(x + 8, (double)2 / 3) *
            (15 * x * x - 144 * x + 1728)) / 40;
    }
}
```

Чтобы функции можно было вызывать в пользовательском интерфейсе, они должны унаследовать класс IntegrationMethods и переопределить его абстрактные методы под возвращение соответствующих значений. Для этого я объявил для каждой функции дочерние классы Function1 и Function2 соответственно и в каждом из них переопределил методы Y(), IntY() и FuncSpelling() с помощью ключевого слова override.

```

public class Function1 : IntegrationMethods
{
    public override double Y(double x)
    {
        return Funcs.Function1(x);
    }

    public override double IntY(double low, double hi)
    {
        return FuncInts.FunctionInt1(hi) - FuncInts.FunctionInt1(low);
    }

    public override string FuncSpelling()
    {
        return "y = 8 / (5 + 2*x^2)";
    }
}

public class Function2 : IntegrationMethods
{
    public override double Y(double x)
    {
        return Funcs.Function2(x);
    }

    public override double IntY(double low, double hi)
    {
        return FuncInts.FunctionInt2(hi) - FuncInts.FunctionInt2(low);
    }

    public override string FuncSpelling()
    {
        return "y = x^2 / (x + 8)^(1/3)";
    }
}

```

## Модуль Program.cs

Основной модуль, обеспечивающий связь вышеописанных операционных модулей с пользовательским интерфейсом командной строки. В сгенерированном по умолчанию классе Program объявлено статическое поле function, являющееся экземпляром абстрактного класса IntegrationMethods.

```

public static IntegrationMethods function;

```

Чаще всего объявляются экземпляры дочерних классов с уже уточнённым и переопределённым функционалом, но в данном случае использование «родительского» экземпляра оправдано — в начале работы программы возможен выбор интегрируемой

функции посредством меню, и неизвестно, с какой функцией решит работать пользователь. Поэтому этот универсальный экземпляр можно инициализировать любым из потомков его класса, что и будет сделано во время подготовки к вычислениям.

Всё действие происходит в методе `Main()` – в данном случае можно себе позволить такую нагрузку, т.к. весь функционал вынесен в отдельные модули, и программа лишь оперирует уже готовым алгоритмом действий.

В начале метода объявляются вещественные переменные для границ интегрирования, количества шагов и байтовая переменная для хранения выбранного пункта в меню (тип данных `byte` выбран для экономии памяти).

```
double lowerBound = 0, upperBound = 1, stepCount = 20;  
byte choice = 0;
```

Далее задаётся заголовок окна консоли и выводится название работы:

```
Console.Title = "Проект по модулю, часть 2, вариант 8. Корнев Дмитрий, ЭУ-  
193631";  
Console.WriteLine("Приближённое вычисление интегралов заданных функций");
```

После этого запускается главный цикл программы, выход из которого обеспечивается через меню – после чего программа закрывается. В начале цикла выводится непосредственно текстовое меню с вариантами выбора двух функций, либо выхода из приложения:

```
Console.Write("\nМеню\n" +  
    "1.  $y = 8 / (5 + 2 \cdot x^2)$ \n" +  
    "2.  $y = x^2 / (x + 8)^{(1/3)}$ \n" +  
    "3. Выход\n\n");
```

Чтобы при неверно введённом значении (например, 8 или «а») программа не завершалась с ошибкой, введена логическая переменная `inputFail`, которая хранит ложное значение, пока не происходит верного ввода, и блок `try-catch`, проверяющий верность типа данных. В случае несоответствия типа блок отправляет в меню значение 0, что тоже является неверным, поэтому цикл начинается заново и пользователь повторяет ввод.

Если получено верное значение (от 1 до 3) выбирается один из пунктов меню с помощью блока `switch-case` и программа работы продолжается. Ниже представлен полностью код всего цикла проверки, т.к. разрывать его на части было не совсем правильно:



```

while (inputFail)
{
    Console.Write("Выберите пункт: ");
    try
    {
        choice = Convert.ToByte(Console.ReadLine());
    }
    catch
    {
        choice = 0;
    }

    switch (choice)
    {
        case 1:
            function = new Function1();
            inputFail = false;
            break;
        case 2:
            function = new Function2();
            inputFail = false;
            break;
        case 3:
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.WriteLine("До свидания!\n");
            Console.ResetColor();
            return;
        default:
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("Неверный выбор.\n");
            Console.ResetColor();
            break;
    }
}

```

Следующим шагом переменным `lowerBound`, `upperBound` и `stepCount` присваиваются значения нижней и верхней границ и количества шагов интегрирования соответственно. Проверка значений контролируется блоком `try-catch` в комбинации с циклом `while` во избежание необработанных исключений (состояние ввода контролирует всё та же переменная `inputFail`):

```

Console.Write("\nНижняя граница = ");
lowerBound = Convert.ToDouble(Console.ReadLine());
Console.Write("Верхняя граница = ");
upperBound = Convert.ToDouble(Console.ReadLine());
Console.Write("Количество шагов = ");
stepCount = Convert.ToDouble(Console.ReadLine());

```

Далее вычисляется шаг интегрирования в объявленной для его хранения переменной:

```
double intStep = (upperBound - lowerBound) / stepCount;
```

После того, как шаг интегрирования вычислен, можно посчитать интегралы разными методами, в том числе и аналитическим:

```
var midRectsResult = function.MidRectMethod(lowerBound, intStep, stepCount);  
var trapeziaResult = function.TrapeziaMethod(lowerBound, intStep, stepCount);  
var simpson2Result = function.Simpson2Method(lowerBound, intStep, stepCount);  
var analyticResult = function.IntY(lowerBound, upperBound);
```

И когда всё, наконец, вычислено, можно выводить результат:

```
Console.WriteLine("Результаты численного интегрирования\n" +  
"-----\n" +  
$"Функция {choice}. {function.FuncSpelling()}; " +  
$"шаг {intStep:f2}; a={lowerBound}; b={upperBound}; s = {analyticResult:f8}");  
  
Console.WriteLine($"Метод средних прямоугольников | {midRectsResult,11:f8}, " +  
$"погрешность: {Math.Abs(analyticResult - midRectsResult),-11:f8}\n" +  
$"Метод трапеций | {trapeziaResult,11:f8}, " +  
$"погрешность: {Math.Abs(analyticResult - trapeziaResult),-11:f8}\n" +  
$"Метод Симпсона 3/8 | {simpson2Result,11:f8}, " +  
$"погрешность: {Math.Abs(analyticResult - simpson2Result),-11:f8}");  
  
Console.WriteLine(  
"-----");
```

После вывода результата программа возвращается в начало основного цикла и предлагает повторно выбрать функцию для вычислений с новыми значениями, либо выйти из программы.

## 2.3. Диаграмма классов

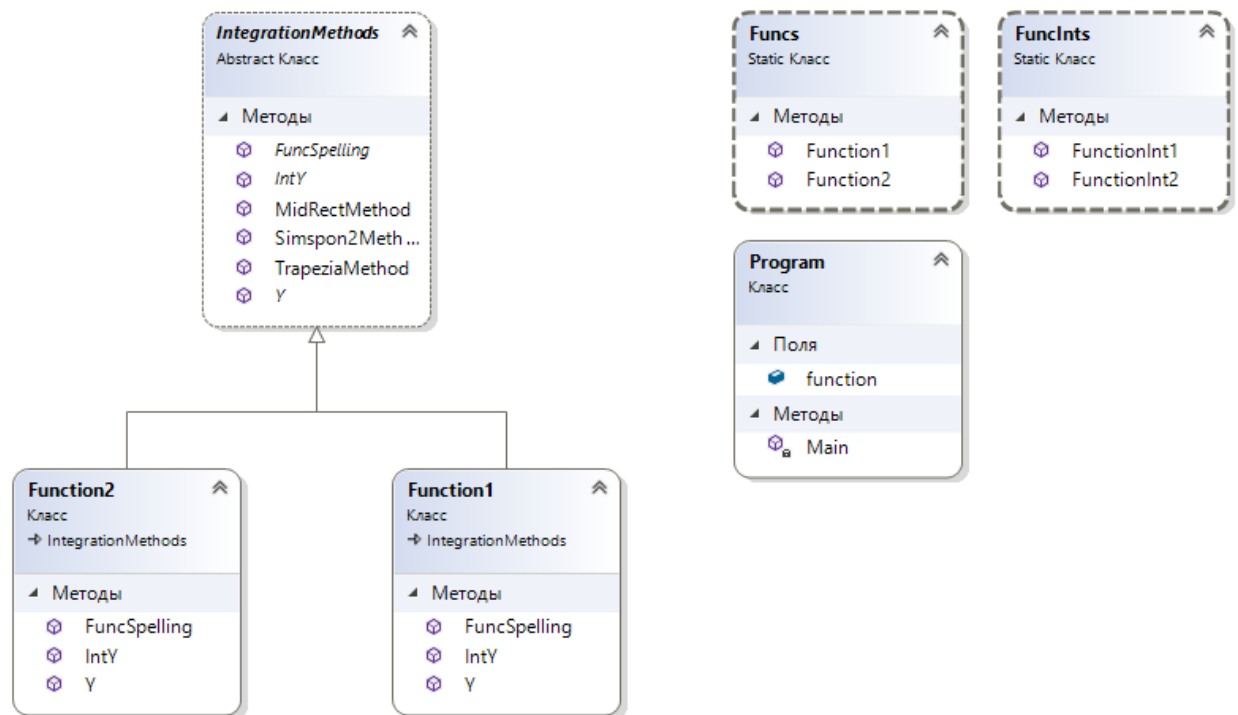


Рисунок 2 - Развёрнутая диаграмма классов проекта function-integration

## 2.4. Тестирование

При запуске программы в командную строку выведется меню с тремя предложенными вариантами. Если ввести вариант не из списка – будет вызвано исключение, но программа не завершится, а предложит повторить ввод:

```
Проект по модулю, часть 2, вариант 8. Корнев Дмитрий, ЭУ-193631

D:\CODE\module-project-jun2020\function-integration\bin\Debug>function-integration.exe
Приближенное вычисление интегралов заданных функций

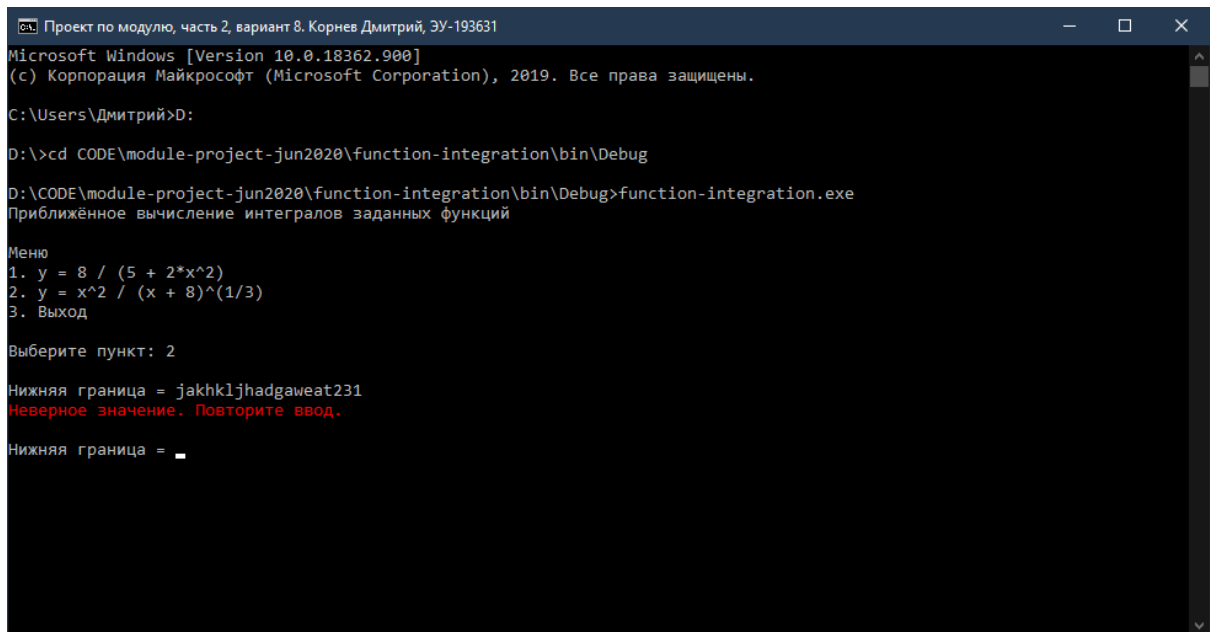
Меню
1. y = 8 / (5 + 2*x^2)
2. y = x^2 / (x + 8)^(1/3)
3. Выход

Выберите пункт: 4
Неверный выбор.

Выберите пункт:
```

Рисунок 3 - обработанное исключение неверного выбора пункта меню

Если выбрать одну из двух функций и ошибиться во введённом значении, например, нижней границы, программа так же на это среагирует и предложит начать ввод параметров заново (но уже в пределах выбранной функции):



```
Проект по модулю, часть 2, вариант 8. Корнев Дмитрий, ЭУ-193631
Microsoft Windows [Version 10.0.18362.900]
(c) Корпорация Майкрософт (Microsoft Corporation), 2019. Все права защищены.

C:\Users\Дмитрий>D:
D:\>cd CODE\module-project-jun2020\function-integration\bin\Debug
D:\CODE\module-project-jun2020\function-integration\bin\Debug>function-integration.exe
Приближённое вычисление интегралов заданных функций

Меню
1. y = 8 / (5 + 2*x^2)
2. y = x^2 / (x + 8)^(1/3)
3. Выход

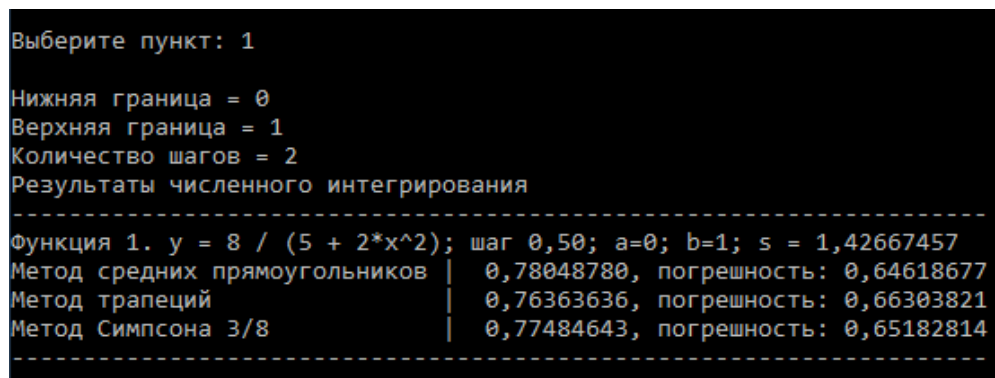
Выберите пункт: 2

Нижняя граница = jakhkljhadgaweat231
Неверное значение. Повторите ввод.

Нижняя граница = _
```

Рисунок 4 - Обработанное исключение неверного значения границы

Перейдём к вычислениям интегралов. Если выбрать малое количество шагов, например, 2 – погрешность будет огромной:

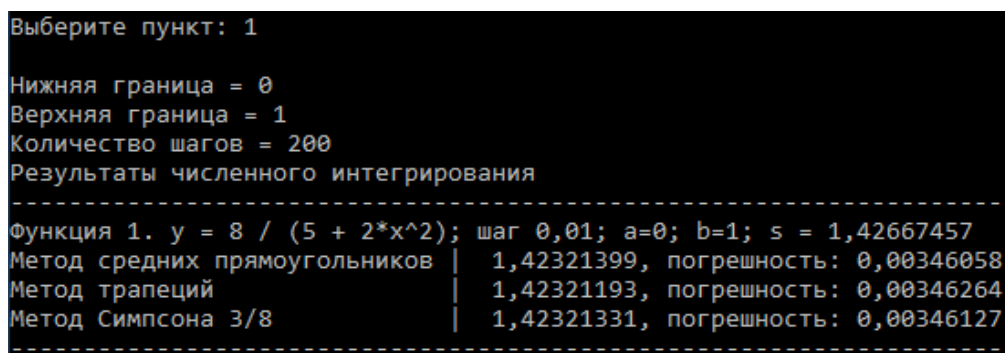


```
Выберите пункт: 1

Нижняя граница = 0
Верхняя граница = 1
Количество шагов = 2
Результаты численного интегрирования
-----
Функция 1. y = 8 / (5 + 2*x^2); шаг 0,50; a=0; b=1; s = 1,42667457
Метод средних прямоугольников | 0,78048780, погрешность: 0,64618677
Метод трапеций | 0,76363636, погрешность: 0,66303821
Метод Симпсона 3/8 | 0,77484643, погрешность: 0,65182814
-----
```

Рисунок 5 - Пример вычислений для первой функции с двумя шагами

Если увеличить количество шагов, то погрешность, естественно, сокращается:



```
Выберите пункт: 1

Нижняя граница = 0
Верхняя граница = 1
Количество шагов = 200
Результаты численного интегрирования
-----
Функция 1. y = 8 / (5 + 2*x^2); шаг 0,01; a=0; b=1; s = 1,42667457
Метод средних прямоугольников | 1,42321399, погрешность: 0,00346058
Метод трапеций | 1,42321193, погрешность: 0,00346264
Метод Симпсона 3/8 | 1,42321331, погрешность: 0,00346127
-----
```

Отмечу, что самым точным остаётся метод средних прямоугольников, метод Симпсона 3/8 занимает второе место, а дальше всех от идеала оказался метод трапеций.

Для второй функции повторяю те же самые операции (кстати, округленное до 8 знаков после запятой аналитическое значение совпало со значением в таблице вариантов – 0,1617797191):

```
Нижняя граница = 0
Верхняя граница = 1
Количество шагов = 20
Результаты численного интегрирования
-----
Функция 2.  $y = x^2 / (x + 8)^{(1/3)}$ ; шаг 0,05; a=0; b=1; s = 0,16177972
Метод средних прямоугольников | 0,11820069, погрешность: 0,04357903
Метод трапеций                  | 0,11848322, погрешность: 0,04329650
Метод Симпсона 3/8              | 0,11829487, погрешность: 0,04348485
-----
```

Рисунок 6 - Пример для второй функции, 20 шагов

```
Нижняя граница = 0
Верхняя граница = 1
Количество шагов = 2000
Результаты численного интегрирования
-----
Функция 2.  $y = x^2 / (x + 8)^{(1/3)}$ ; шаг 0,00; a=0; b=1; s = 0,16177972
Метод средних прямоугольников | 0,16129943, погрешность: 0,00048029
Метод трапеций                  | 0,16129946, погрешность: 0,00048026
Метод Симпсона 3/8              | 0,16129944, погрешность: 0,00048028
-----
```

Рисунок 7 - Пример для второй функции, 2000 шагов

Вычисления выполнялись на отрезке  $[0, 1]$ , и здесь лидером по точности оказался метод трапеций, метод Симпсона 3/8 остался на втором месте, а третье занял метод средних прямоугольников.

После того, как я протестировал обе функции, я вышел из программы с помощью пункта 3 в меню:

```
Меню
1.  $y = 8 / (5 + 2*x^2)$ 
2.  $y = x^2 / (x + 8)^{(1/3)}$ 
3. Выход

Выберите пункт: 3
До свидания!

D:\CODE\module-project-jun2020\function-integration\bin\Debug>
```

Рисунок 8 - Выход из программы

## 2.5. Вывод

В ходе разработки приложения я следовал парадигме объектно-ориентированного программирования. Я закрепил навыки работы с классами, абстракциями, наследованием и использовал различные базовые инструменты языка C#, такие как форматированный вывод и математические функции. Также на практике выяснил, какой из представленных в варианте работы методов вычисления интеграла эффективнее для конкретной функции.

### 3. Объектно-ориентированная разработка приложения для работы с одномерным массивом

#### 3.1. Цель работы

В среде Microsoft Visual C# необходимо создать приложение, в котором создаются, наследуются и применяются классы и объекты при работе с одномерным числовым массивом, а также используются графические средства для визуализации числовых данных массива и операций, выполняемых с этими данными.

Код программы было решено разделить на три модуля вместо двух:

- ArrayKit.cs – модуль классов, обеспечивающих работу с целочисленным массивом;
- Form1.cs – модуль формы, предоставляющей графический интерфейс приложения и управление компонентами программы;
- Histogram.cs – отдельный модуль для статического класса, отвечающего за построение гистограммы в форме.

По моему варианту с массивом необходимо выполнить следующие операции:

№	Задание для одномерного массива	Размещение чисел в файле	Способ сортировки
8.	Получить новый массив из сумм соседних элементов исходного массива	В одной строке через пробел	Selection2

## 3.2. Разработка приложения

Разработка программы велась на языке C#. Приложение создавалось с помощью нескольких средств разработки: код модуля классов разрабатывался в редакторе кода Visual Studio Code, графический интерфейс формы создавался в среде разработки Visual Studio 2019. Все файлы проекта были объединены в решение one-dim-array.

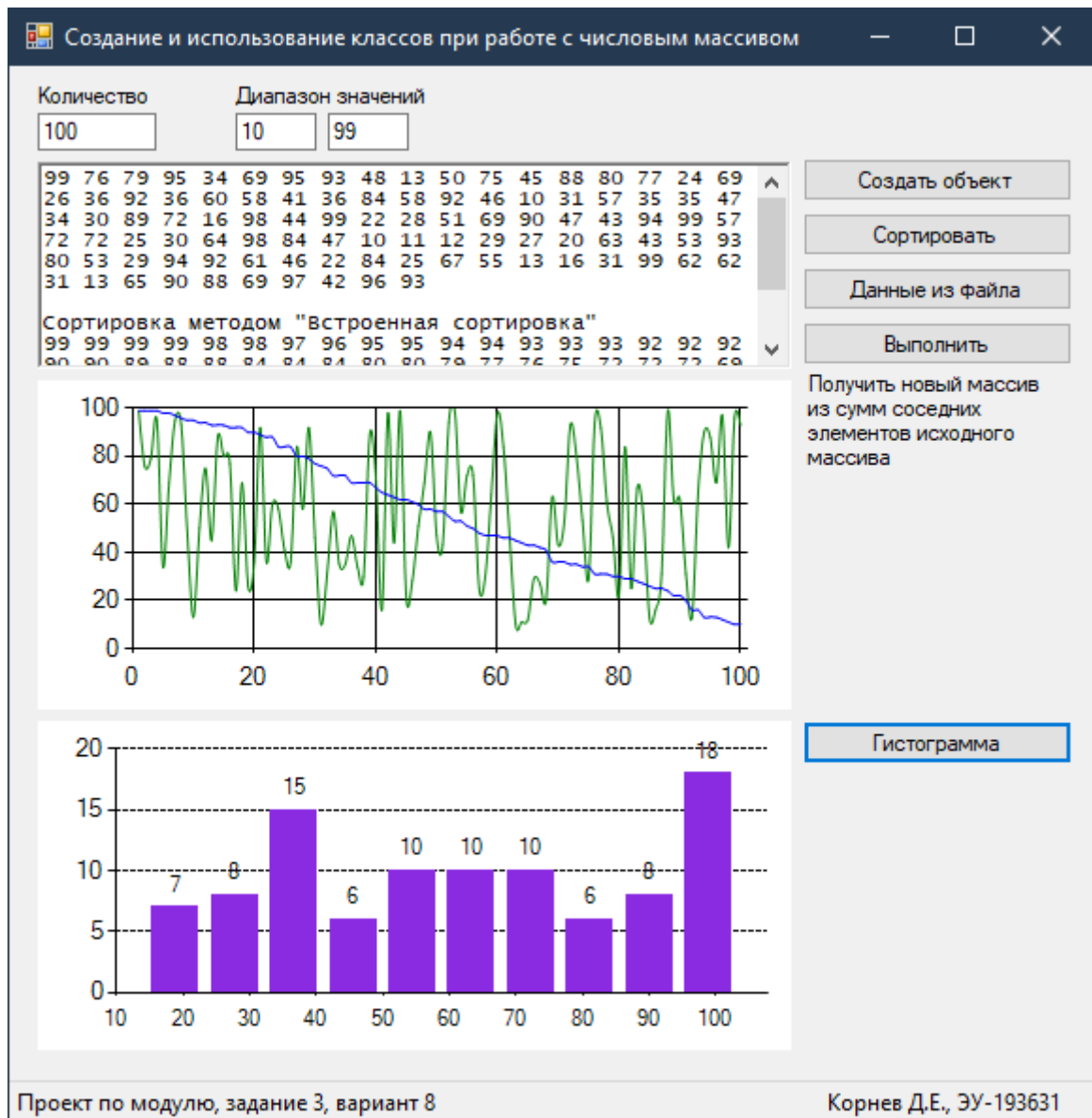


Рисунок 9 - Внешний вид формы Form1.cs

При создании пользовательского интерфейса приложения использовались следующие элементы Windows Forms:

- `MaskedTextBox` – для ввода количества элементов верном формате (в форме – `elementsCount`);
- `TextBox` – для ввода диапазона значений (в форме -- `lowerBoundBox` и `upperBoundBox`);



- RichTextBox – для вывода массива и результатов операций с ними;
- Button – для вызова различных функций в форме, например, создания нового массива и его сортировки (в форме – createButton, sortButton, readFromFileButton, plotHistogramButton и executeButton);
- Label – для отображения пользователю различных пояснений;
- Chart – для визуализации массива в виде графиков Spline и численных интервалов в виде гистограммы;
- StatusStrip – для вывода информации о номере, варианте и авторе работы с помощью двух текстовых элементов.

Далее будет рассмотрено содержание и назначение каждого из модулей программы.

## Модуль ArrayKit.cs

В этом модуле находятся три класса. Родительский CustomArrayBase описывает целочисленный массив в качестве защищённого поля и различные базовые операции с ним в виде методов. Свойствами класса определены Size (размер массива) и SortMode (способ сортировки).

```
protected int[] array;
protected int Size { get; set; }
public string SortMode { get; protected set; }
```

В этом классе так же описан стандартный конструктор, который хоть и не используется в приложении, но может быть полезен для тестирования базовых функций класса – он инициализирует массив десятью заранее заданными числами и записывает в свойство Size его размер.

```
public CustomArrayBase()
{
    array = new int[] {5, 8, 10, 12, 24, 11, 9, 5, 8, 1};
    Size = array.Length;
}
```

Метод GetArray() возвращает массив, объявленный в качестве поля класса array:

```
public int[] GetArray()
{
    return array;
}
```

Метод `Sort()` с ключевым словом `virtual` (делает метод доступным для последующего переопределения) записывает в свойство `SortMode` способ сортировки массива (в данном случае – встроенная сортировка из средств стандартного класса `Array`) и сортирует массив этим способом:

```
public virtual void Sort()
{
    SortMode = "Встроенная сортировка";
    Array.Sort(array);
    Array.Reverse(array);
}
```

Метод `NeighbouringSum()` отвечает за выполнение задания по варианту: находит суммы соседних элементов массива, на основе которых создаёт ещё один массив и возвращает его:

```
public int[] NeighbouringSum()
{
    int[] neighbouringSum = new int[Size - 1];

    for (var i = 0; i < Size - 1; i++)
    {
        neighbouringSum[i] = array[i] + array[i + 1];
    }

    return neighbouringSum;
}
```

В модуле `ArrayKit.cs` также присутствуют два дочерних класса, которые уже непосредственно используются в приложении.

Первый такой класс – `CustomArray`, унаследованный от `CustomArrayBase`. В нём появляются два новых закрытых поля, хранящие в себе нижнюю и верхнюю границы чисел в массиве:

```
private int lowerBound, upperBound;
```

## **Заключение**

