МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**ЛАБОРАТОРНАЯ РАБОТА № 5**

по дисциплине

**«Объектно-ориентированное программирование»**

на тему:

**«Классы, объекты, наследование в C#»**

*Вариант № 8*

Выполнил:

Студент группы

КТбо2-8

Данильченко Т. В.

Проверил:

Тарасов С. А.

Оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Таганрог 2020

# **1 Цель работы**

Ознакомление с основами объектно-ориентированного программирования на С#.

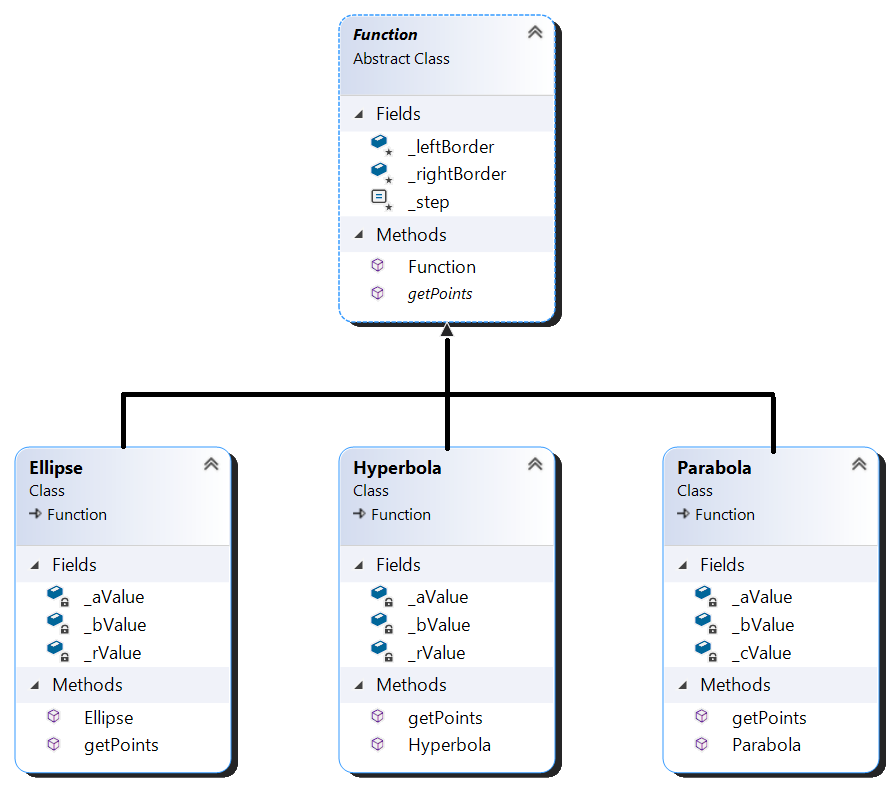
# **2 Задание, вариант № 8**

Создать класс Function (функция) с методами вычисления значения функции y = f(x) в заданной точке х. На его основе определить классы Ellipse, Hiperbola и Parabola, в которых реализуются соответствующие математические зависимости.

# **3 Ход работы**

**3.1 Спецификации классов**

Диаграмма используемых классов приведена на Рисунке 1.

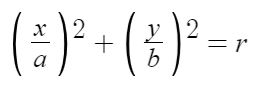


*Рисунок 1 — Диаграмма классов*

В классе Function объявлены 3 переменных, две из которых отвечают за границы рассматриваемой области и константная переменная, отвечающая за шаг на графике. Так же описан абстрактный метод получения точек графика

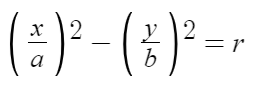
В классах Ellipse, Hyperbola, Parabola объявлены переменные отвечающие за коэффициенты соответствующего графика и так же описан метод получения точек, для каждого графика соответственно

Для класса Ellipse была использована упрощенная математическая формула эллипса, которая изображена на Рисунке 2.



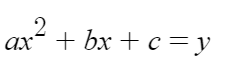
*Рисунок 2 – упрощенная формула эллипса*

Для класса Hyperbola была использована математическая формула гиперболы, которая изображена на Рисунке 3.



*Рисунок 3 –формула гиперболы*

Для класса Parabola была использована математическая формула параболы, которая изображена на Рисунке 4.



*Рисунок 4 –формула параболы*

**3 Листинг**

//Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace DanilchenkoTV\_Lab5

{

static class Program

{

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new FunctionCalculator());

}

}

}

//Function.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DanilchenkoTV\_Lab5

{

abstract class Function

{

protected double \_leftBorder;

protected double \_rightBorder;

protected const double \_step = 0.01;

public Function() { }

abstract public List<KeyValuePair<double, double>> getPoints();

}

}

//Ellipse.cs

using System;

using System.Collections.Generic;

namespace DanilchenkoTV\_Lab5

{

class Ellipse:Function

{

private double \_aValue;

private double \_bValue;

private double \_rValue;

public Ellipse(double a, double b, double r)

{

\_leftBorder = 0;

\_rightBorder = 0;

\_aValue = a;

\_bValue = b;

\_rValue = r;

}

public override List<KeyValuePair<double, double>> getPoints()

{

List<KeyValuePair<double, double>> points = new List<KeyValuePair<double, double>>();

if (\_aValue == 0 || \_bValue == 0)

{

return points;

}

double y;

double x = -\_aValue;

while (x <= \_aValue)

{

y = (\_bValue \* Math.Sqrt(\_aValue \* \_aValue - x \* x)) / \_aValue;

points.Add(new KeyValuePair<double, double>(x, y));

x += \_step;

}

x = \_aValue;

while (x >= -\_aValue)

{

y = (\_bValue \* Math.Sqrt(\_aValue \* \_aValue - x \* x)) / \_aValue;

points.Add(new KeyValuePair<double, double>(x, -y));

x -= \_step;

}

x = -\_aValue;

y = (\_bValue \* Math.Sqrt(\_aValue \* \_aValue - x \* x)) / \_aValue;

points.Add(new KeyValuePair<double, double>(x, y));

return points;

}

}

}

//Hyperbola.cs

using System;

using System.Collections.Generic;

namespace DanilchenkoTV\_Lab5

{

class Hyperbola:Function

{

private double \_aValue;

private double \_bValue;

private double \_rValue;

public Hyperbola(double lb, double rb, double a, double b, double r)

{

\_leftBorder = lb;

\_rightBorder = rb;

\_aValue = a;

\_bValue = b;

\_rValue = r;

}

public override List<KeyValuePair<double, double>> getPoints()

{

List<KeyValuePair<double, double>> points = new List<KeyValuePair<double, double>>();

if (\_aValue == 0 || \_bValue == 0)

{

return points;

}

if(\_leftBorder > -\_rValue && \_rightBorder > \_rValue)

{

return points;

}

double y;

double x = \_leftBorder;

while (x <= -\_rValue)

{

y = (Math.Sqrt(-1 \* (\_bValue \* \_bValue) \* (\_aValue \* \_aValue \* \_rValue - x \* x))) / \_aValue;

points.Add(new KeyValuePair<double, double>(x, y));

x += \_step;

}

x = -\_rValue;

while (x >= \_leftBorder)

{

y = (Math.Sqrt(-1 \* (\_bValue \* \_bValue) \* (\_aValue \* \_aValue \* \_rValue - x \* x))) / \_aValue;

points.Add(new KeyValuePair<double, double>(x, -y));

x -= \_step;

}

x = \_rightBorder;

while (x >= \_rValue)

{

y = (Math.Sqrt(-1 \* (\_bValue \* \_bValue) \* (\_aValue \* \_aValue \* \_rValue - x \* x))) / \_aValue;

points.Add(new KeyValuePair<double, double>(x, -y));

x -= \_step;

}

x = \_rValue;

while (x <= \_rightBorder)

{

y = (Math.Sqrt(-1 \* (\_bValue \* \_bValue) \* (\_aValue \* \_aValue \* \_rValue - x \* x))) / \_aValue;

points.Add(new KeyValuePair<double, double>(x, y));

x += \_step;

}

return points;

}

}

}

//Parabola.cs

using System;

using System.Collections.Generic;

namespace DanilchenkoTV\_Lab5

{

class Parabola:Function

{

private double \_aValue;

private double \_bValue;

private double \_cValue;

public Parabola(double lb, double rb, double a, double b, double c)

{

\_leftBorder = lb;

\_rightBorder = rb;

\_aValue = a;

\_bValue = b;

\_cValue = c;

}

public override List<KeyValuePair<double, double>> getPoints()

{

List<KeyValuePair<double, double>> points = new List<KeyValuePair<double, double>>();

double x = \_leftBorder;

double y;

while (x <= \_rightBorder)

{

y = \_aValue \* x \* x + \_bValue \* x + \_cValue;

points.Add(new KeyValuePair<double, double>(x, y));

x += \_step;

}

return points;

}

}

}