Пусть требуется создать класс для реализации методов решения квадратного уравнения *ax*2+*bx*+*c*=0. Уравнение задается набором коэффициентов (от 1 до 3). Если при создании указывается иное количество коэффициентов, то квадратное уравнение определить нельзя, поэтому выдается предупреждение об ошибке. В классе должны быть предусмотрены средства для решения уравнений, в которых *a*=0, *b*=0 или *c*=0. Тогда уравнение может стать линейным (0*\*x*2+5*x*+2=0), обратиться в тождество (0 = 0) или стать неразрешимым (6 = 0).

1. Определение полей класса.

61

Поля класса определяют его структурные свойства, т.е. описывают состав объекта класса. Для определения квадратного уравнения требуется указать его коэффициенты: a, b и c. Также можно ввести в класс поля, характеризующие решение

уравнения – количество корней (count) и сами корни (максимальное количество корней – 2) – x1 и x2. Эти поля нужны, в частности, чтобы не осуществлять повторное решение уравнения.

class Equation

{

// поля класса

// константа для обозначения

// бесконечного количества корней

const int infinity = Int32.MaxValue;

// переменные для хранения коэффициентов уравнения double a,b,c;

// поле для хранения количества корней уравнения // (-1 означает, что вычисление корней

// уравнения пока не осуществлялось)

int count=-1;

// переменные для хранения корней уравнения double x1,x2;

. . .

}

Далее будет указываться только код отдельных элементов, которые добавляются к классу Equation.

2. Определение конструктора класса.

Для создания объекта класса Equation требуется задать его коэффициенты. Поскольку количество коэффициентов может быть переменным (от 1 до 3), конструктор можно сделать методом с переменным числом параметров (определенным с ключевым словом params). Если будет задано недопустимое количество коэффициентов, сгенерируется соответствующее исключение:

62

// конструктор объекта-уравнения

public Equation(params double [] coef)

{

// в зависимости от количества параметров (длины массива) // получаем различные виды уравнений

switch (coef.Length)

{

case 3:

// квадратное уравнение

a = coef[0];

b = coef[1];

c = coef[2];

break;

case 2:

// линейное уравнение

a = 0.0;

b = coef[0];

c = coef[1];

break;

case 1:

// тождество или неразрешимое уравнение

a = 0.0;

b = 0.0;

c = coef[0];

break;

default:

// генерация исключения при некорректном

// наборе коэффициентов

throw new Exception("Данный набор коэффициентов не может определять квадратное уравнение");

}

}

3. Определение методов решения уравнения.

Для определения разрешимости уравнения сначала должен вызываться метод Solve(). В его задачу входит определение порядка уравнения (квадратное – уравнение 2-ого порядка, линейное – уравнение 1-ого порядка, тождество или неразрешимое – уравнение 0-ого порядка).

public void Solve()

{

if (a == 0)

//линейное уравнение

if (b == 0)

// уравнение нулевого порядка

if (c == 0)

63

{

// тождество

count = infinity;

}

else

{

// неразрешимое уравнение

count = 0;

}

else

// линейное уравнение

LinSolve();

else

//квадратное уравнение

QSolve();

}

Для решения линейных и квадратных уравнений дополнительно вызываются методы: QSolve () – для уравнений 2-ого порядка, LinSolve() – для уравнений 1-ого порядка.

// метод решения квадратного уравнения

public void QSolve()

{

// вычисление дискриминанта

double disc = b \* b - 4 \* a \* c;

if (disc < 0.0)

count = 0; // вещественных корней нет else if (disc == 0.0)

{

// уравнение имеет один корень

count = 1;

x1 = x2 = -b / (2 \* a);

}

else

{

// уравнение имеет два корня

count = 2;

x1 = (-b + Math.Sqrt(disc)) / (2 \* a); x2 = (-b - Math.Sqrt(disc)) / (2 \* a); }

}

// метод решения линейного уравнения

public void LinSolve()

{

count = 1;

x1 = -c / b;

}

64

4. Определение свойств для доступа к коэффициентам уравнения и получения количества его корней.

Согласно принципу инкапсуляции данные класса являются закрытыми (private). К ним можно обращаться только из методов самого класса. В случае, когда требуется изменить или получить значения тех или иных данных класса следует определять специальные методы доступа к данным (обычно называются get- и set-методы). Как правило, эти методы оформляются в виде свойств класса.

Приведем определения свойств для доступа к коэффициентам квадратного уравнения:

// свойства для обеспечения доступа

// к коэффициентам уравнения

public double A

{

get { return a; }

set { a = value; }

}

public double B

{

get { return b; }

set { b = value; }

}

public double C

{

get { return c; }

set { c = value; }

}

Для получения количества корней уравнения создадим также свойство Сount. В отличие от свойств, осуществляющих доступ к коэффициентам уравнения, Count доступно только для чтения, так как количество корней должно вычисляться, а не определяться пользователем.

// свойство для получения количества корней уравнения public int Count

{

get { return count; }

}

65

5. Определение индексатора для получения корней уравнения по номеру.

Отличием индексаторов от свойств является то, что индексатор осуществляет доступ к отдельному элементу из упорядоченного набора значений. Упорядочение в данном случае понимается как соответствие каждого элемента определенному набору индексов (индексов может быть несколько и они могут задаваться с помощью различных типов данных – целые числа, строки, символы, объекты какого-то класса и пр.).

В класс Equation введем индексатор для получения корня уравнения по его номеру. Очевидно, что в качестве индекса здесь используется целое число (1 или 2), соответствующее номеру корня. В случаях, когда корня с заданным номером не существует или уравнение еще не решено, будет сгенерирована исключительная ситуация. Данный индексатор должен предоставлять доступ только для чтения, поскольку корни уравнения должны вычисляться.

// индексатор для получения корня уравнения по его номеру public double this[int i]

{

get

{

if (count == -1)

throw new Exception("Уравнение еще не решено"); if ((count == 1 || count==2) && i == 1) return x1;

if (count == 2 && i == 2)

return x2;

throw new Exception("Уравнение не имеет корня с номером "+i);

}

}

6. Метод распечатки корней уравнения.

Для того чтобы распечатать корни уравнения, добавим в класс специальный метод:

66

// метод распечатки корней уравнения

public void PrintSolution()

{

// сообщение определяется значением количества // корней уравнения

switch(count)

{

case 2: Console.WriteLine("x1={0}, x2={1}",x1,x2); break;

case 1: Console.WriteLine("x={0}",x1);

break;

case 0: Console.WriteLine("Корней нет"); break;

case Int32.MaxValue:

Console.WriteLine("Любое x является решением");

break;

default:

Console.WriteLine("Уравнение еще не решено");

break;

}

}

7. Пример использования класса Equation.

Простая последовательность действий при создании и решении уравнения такова (Рис.4.7):

try

{

// создание объекта-уравнения

Equation e = new Equation(2.0,5.0,-9.0);

// решение уравнения

e.Solve();

// распечатка полученного решения

e.PrintSolution();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

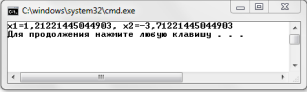


Рис.4.7. Результат решения квадратного уравнения

67

Сначала создается объект-уравнение, далее для него вызываются метод Solve() (решение уравнения) и метод PrintSolution() (печать корней). Поскольку в конструкторе может быть сгенерировано исключение, эти три команды помещаются в блок try. Соответствующий ему блок catch печатает сообщение об ошибке.

Если мы хотим изменить значение какого-то коэффициента уравнения, можно воспользоваться свойствами A, B или С. В следующем фрагменте кода после решения уравнения меняеться его коэффициент B и уравнение решается заново (Рис.4.8):

try

{

Equation e = new Equation(2.0,5.0,-9.0);

e.Solve();

e.PrintSolution();

**// изменение коэффициента B уравнения и его решение e.B = -3.0;**

**e.Solve();**

**e.PrintSolution();**

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

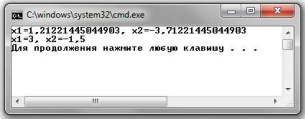


Рис.4.8. Результат решения квадратного уравнения

до и после изменения его коэффициентов

Для демонстрации использования индексатора распечатаем корни уравнения без использования функции PrintSolution().

Equation e = new Equation(2,5);

e.Solve();

68

try

{

if (e.Count == Int32.MaxValue) Console.WriteLine("Корней много"); // печать первого корня уравнения Console.WriteLine("x1 = {0}", **e[1]**);

// печать второго корня уравнения Console.WriteLine("x2 = {0}", **e[2]**); }

catch (Exception ex)

{

Console.WriteLine(ex.Message); }