Программа функционирует при помощи абстрактных типов данных(классов), которые производят определенные операции, которые составляют логику калькулятора, его функционирование.

Разработана иерархия классов, в основе которой лежит класс число(iNimber), от него наследует класс комплексное число (Complex), в классе iNumber все операции над комплексными числами объявлены как абстрактные методы.

public interface iNumber

{

bool isZero();

iNumber Copy();

iNumber Add(iNumber Value);

iNumber Deduct(iNumber Value);

iNumber Mul(iNumber Value);

iNumber Divide(iNumber Value);

iNumber Sqr();

iNumber Rev();

string GetNumberString();

}

Класс Complex(комплексное число) производит операции над комплексными числами, тип хранимого числа – iNumber.

public class Complex : iNumber

{

private double \_re, \_im;

static int \_base;

public bool isZero()

{

if (\_re != 0 || \_im != 0)

return false;

return true;

}

public Complex()

{

\_re = 0;

\_im = 0;

}

public Complex(double Re, double Im)

{

\_re = Re;

\_im = Im;

}

public Complex(string Value, int Base)

{

\_base = Base;

ParseString(Value);

}

private void ParseString(string str)

{

if (str.Length < 4)

throw new FormatException();

string stRe, stIm;

if (str.IndexOf('+',1) != -1)

{

stRe = str.Substring(0, str.IndexOf('+',1));

stIm = str.Substring(str.IndexOf('+', 1) + 2);

}

else

{

stRe = str.Substring(0, str.IndexOf('-', 1));

stIm = '-' + str.Substring(str.IndexOf('-', 1) + 2);

}

\_re = StringToDouble(stRe, \_base);

\_im = StringToDouble(stIm, \_base);

}

public iNumber Copy()

{

return new Complex(\_re, \_im);

}

public iNumber Add(iNumber Value)

{

double Re = \_re + ((Complex)Value).\_re;

double Im = \_im + ((Complex)Value).\_im;

return new Complex(Re, Im);

}

public iNumber Deduct(iNumber Value)

{

double Re = \_re - ((Complex)Value).\_re;

double Im = \_im - ((Complex)Value).\_im;

return new Complex(Re, Im);

}

public iNumber Mul(iNumber Value)

{

double Re = \_re \* ((Complex)Value).\_re - \_im \* ((Complex)Value).\_im;

double Im = \_im \* ((Complex)Value).\_re + ((Complex)Value).\_im \* \_re;

return new Complex(Re, Im);

}

public iNumber Divide(iNumber Value)

{

if (((Complex)Value).\_re == 0 && ((Complex)Value).\_im == 0)

throw new DivideByZeroException();

double den = (Math.Pow(((Complex)Value).\_re, 2) + Math.Pow(((Complex)Value).\_im, 2));

double Re = (\_re \* ((Complex)Value).\_re + \_im \* ((Complex)Value).\_im) / den;

double Im = (((Complex)Value).\_re \* \_im - \_re \* ((Complex)Value).\_im) / den;

return new Complex(Re, Im);

}

}

Абстрактный тип данных Memory(память) может хранить и обрабатывать комплексные числа, тип хранимого числа – iNumber.

public class Memory<T> where T : iNumber

{

private T FNumber;

bool FState;

public Memory()

{

FState = false;

}

public void Set(T E)

{

FNumber = E;

FState = true;

}

public T Get()

{

FState = true;

return FNumber;

}

public void Sum(T E)

{

FState = true;

FNumber = (T)FNumber.Add(E);

}

public void Deduct(T E)

{

FState = true;

FNumber = (T)FNumber.Deduct(E);

}

public void Clear()

{

FNumber = default(T);

FState = false;

}

public string GetState()

{

if (FState)

return "\_On";

return "\_Off";

}

}

Абстрактный тип данных Proc(процессор) обрабатывает комплексные числа, тип хранимого числа – iNumber.

class Proc<T> where T : iNumber

{

private T \_Lop, \_Rop;

private Operation \_operation;

public Proc()

{

ResetProc();

}

public void ResetProc()

{

\_operation = Operation.None;

}

public void ResetOprtn()

{

\_operation = Operation.None;

}

public void RunOprtn()

{

switch (\_operation)

{

case Operation.Add:

\_Lop = (T)\_Lop.Add(\_Rop);

break;

case Operation.Deduct:

\_Lop = (T)\_Lop.Deduct(\_Rop);

break;

case Operation.Division:

\_Lop = (T)\_Lop.Copy().Divide(\_Rop.Copy());

break;

case Operation.Mul:

\_Lop = (T)\_Lop.Mul(\_Rop);

break;

}

}

public void RunFunc(Func func)

{

switch (func)

{

case Func.Revers:

\_Rop = (T)\_Rop.Rev();

break;

case Func.Sqr:

\_Rop = (T)\_Rop.Sqr();

break;

}

}

public T GetLop()

{

return (T)\_Lop.Copy();

}

public void SetLop(T E)

{

\_Lop = (T)E.Copy();

}

public T GetRop()

{

return (T)\_Rop.Copy();

}

public void SetRop(T E)

{

\_Rop = (T)E.Copy();

}

public Operation GetState()

{

return \_operation;

}

public void SetState(Operation oprtn)

{

\_operation = oprtn;

}

}

Класс ControllerManager управляет калькулятором.

public void MAdd()

{

iNumber n = Init();

if (\_mem.Get() is Complex)

{

if (n is Complex)

\_mem.Sum(n);

}

}

public void MDeduct()

{

iNumber n = Init();

if (\_mem.Get() is Complex)

{

if (n is Complex)

\_mem.Deduct(n);

}

}

public void Compute(Operation \_operation)

{

if (\_isTotal)

{

strHistory = "";

\_isTotal = false;

}

if (\_proc.GetState() == Operation.None)

{

iNumber n = Init();

\_proc.SetLop(n);

\_proc.SetRop(\_proc.GetLop());

\_proc.SetState(\_operation);

}

else

{

iNumber n = Init();

n = Convert(n, \_proc.GetLop().GetType());

if (n == null)

{

Debug.WriteLine("3");

throw new FormatException("Can't continue");

}

\_proc.SetRop(n);

\_proc.RunOprtn();

\_proc.SetState(\_operation);

}

\_isUsing = true;

strHistory += \_proc.GetRop().GetNumberString();

strCalculate = \_proc.GetLop().GetNumberString();

switch (\_operation)

{

case Operation.Add: strHistory += "+"; break;

case Operation.Deduct: strHistory += "-"; break;

case Operation.Division: strHistory += "/"; break;

case Operation.Mul: strHistory += "\*"; break;

}

}

Были написаны тесты:

///Тест для Add

[TestMethod()]

public void ComplexAddTest()

{

Complex expected = new Complex(5,5);

Complex target = new Complex(2,4);

Complex Value = new Complex(3,1);

Complex actual;

actual = (Complex)target.Add(Value);

Assert.AreEqual(expected.GetNumberString(), actual.GetNumberString());

}

///Тест для Deduct

[TestMethod()]

public void ComplexDeductTest()

{

Complex expected = new Complex(3,1);

Complex target = new Complex(5,4);

Complex Value = new Complex(2,3);

Complex actual;

actual = (Complex)target.Deduct(Value);

Assert.AreEqual(expected.GetNumberString(), actual.GetNumberString());

}

///Тест для Mul

[TestMethod()]

public void ComplexMulTest()

{

Complex expected = new Complex(2,0);

Complex target = new Complex(1,1);

Complex Value = new Complex(1,-1);

Complex actual;

actual = (Complex)target.Mul(Value);

Assert.AreEqual(expected.GetNumberString(), actual.GetNumberString());

}

///Тест для Divide

[TestMethod()]

public void ComplexDivideTest()

{

Complex expected = new Complex(2.5,-0.5);

Complex target = new Complex(3,2);

Complex Value = new Complex(1,1);

Complex actual;

actual = (Complex)target.Divide(Value);

Assert.AreEqual(expected.GetNumberString(), actual.GetNumberString());

}

///Тест для Rev

[TestMethod()]

public void ComplexRevTest()

{

Complex expected = new Complex(0.5,0.5);

Complex target = new Complex(1,1);

Complex actual;

actual = (Complex)target.Rev();

Assert.AreEqual(expected.GetNumberString(), actual.GetNumberString());

}

///Тест для Sqr

[TestMethod()]

public void ComplexSqrTest()

{

Complex expected = new Complex(5,12);

Complex target = new Complex(3,2);

Complex actual;

actual = (Complex)target.Sqr();

Assert.AreEqual(expected.GetNumberString(), actual.GetNumberString());

}