**Chaper-1 Testhello.cs console application**

// using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TestHello

{

class Program

{

static void Main(string[] args)

{

System.Console.WriteLine("Hello World!");

}

}

}

**Chapter-2 Promitive Datatypes and Arithmetic operators**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.InteropServices.WindowsRuntime;

using Windows.Foundation;

using Windows.Foundation.Collections;

using Windows.UI.Xaml;

using Windows.UI.Xaml.Controls;

using Windows.UI.Xaml.Controls.Primitives;

using Windows.UI.Xaml.Data;

using Windows.UI.Xaml.Input;

using Windows.UI.Xaml.Media;

using Windows.UI.Xaml.Navigation;

// The Blank Page item template is documented at http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409

namespace PrimitiveDataTypes

{

/// <summary>

/// An empty page that can be used on its own or navigated to within a Frame.

/// </summary>

public sealed partial class MainPage : Page

{

public MainPage()

{

this.InitializeComponent();

}

private void typeSelectionChanged(object sender, SelectionChangedEventArgs e)

{

ListBoxItem selectedType = (type.SelectedItem as ListBoxItem);

switch (selectedType.Content.ToString())

{

case "int":

showIntValue();

break;

case "long":

showLongValue();

break;

case "float":

showFloatValue();

break;

case "double":

showDoubleValue();

break;

case "decimal":

showDecimalValue();

break;

case "string":

showStringValue();

break;

case "char":

showCharValue();

break;

case "bool":

showBoolValue();

break;

}

}

private void showIntValue()

{

int intVar;

intVar = 42;

value.Text = intVar.ToString();

}

private void showLongValue()

{

long longVar;

longVar = 42L;

value.Text = longVar.ToString();

}

private void showFloatValue()

{

float floatVar;

floatVar = 0.42F;

value.Text = floatVar.ToString();

}

private void showDoubleValue()

{

double doubleVar;

doubleVar = 0.42;

value.Text = doubleVar.ToString();

}

private void showDecimalValue()

{

decimal decimalVar;

decimalVar = 0.42M;

value.Text = decimalVar.ToString();

}

private void showStringValue()

{

string stringVar;

stringVar = "forty two";

value.Text = stringVar; // ToString not needed

}

private void showCharValue()

{

char charVar;

charVar = 'x';

value.Text = charVar.ToString();

}

private void showBoolValue()

{

bool boolVar;

boolVar = false;

value.Text = boolVar.ToString();

}

}

}

**Arithmetic Operators**

using System;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

namespace MathsOperators

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void calculateClick(object sender, RoutedEventArgs e)

{

try

{

if ((bool)addition.IsChecked)

{

addValues();

//Exception ex = new Exception("Addion valurecfsds");

//throw ex;

}

else if ((bool)subtraction.IsChecked)

{

subtractValues();

}

else if ((bool)multiplication.IsChecked)

{

multiplyValues();

}

else if ((bool)division.IsChecked)

{

divideValues();

}

}

catch (FormatException fex)

{

result.Text = fex.Message;

// MessageBox.Show(fex.Message);

}

catch (OverflowException oex)

{

MessageBox.Show(oex.Message+"\n Result of muliplication value is very big");

}

catch(Exception ex)

{

MessageBox.Show(ex.Message);

}

finally

{

MessageBox.Show("Must executable statements ");

}

}

private void addValues()

{

int lhs = int.Parse(lhsOperand.Text);

int rhs = int.Parse(rhsOperand.Text);

int outcome = 0;

// TODO: Add rhs to lhs and store the result in outcome

outcome = lhs + rhs;

expression.Text = lhsOperand.Text + " + " + rhsOperand.Text;

result.Text = outcome.ToString();

}

private void subtractValues()

{

int lhs = int.Parse(lhsOperand.Text);

int rhs = int.Parse(rhsOperand.Text);

int outcome = 0;

// TODO: Subtract rhs from lhs and store the result in outcome

outcome = lhs - rhs;

expression.Text = lhsOperand.Text + " - " + rhsOperand.Text;

result.Text = outcome.ToString();

}

private void multiplyValues()

{

int lhs = int.Parse(lhsOperand.Text);

int rhs = int.Parse(rhsOperand.Text);

int outcome = 0;

// TODO: Multiply lhs by rhs and store the result in outcome

outcome =checked(lhs \* rhs);

expression.Text = lhsOperand.Text + " \* " + rhsOperand.Text;

result.Text = outcome.ToString();

}

private void divideValues()

{

int lhs = int.Parse(lhsOperand.Text);

int rhs = int.Parse(rhsOperand.Text);

int outcome = 0;

// TODO: Divide lhs by rhs and store the result in outcome

outcome = lhs / rhs;

expression.Text = lhsOperand.Text + " / " + rhsOperand.Text;

result.Text = outcome.ToString();

}

}

}

**Cahapter-3 dailyRate.cs Optional Parameters**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DailyRate

{

class OptionalParameters

{

public void optMethod(int first=1,double second =0.45,string third = "C#",int fourt =200)

{

Console.WriteLine("First {0},Second {1},third {2},fourth {3}", first, second, third, fourt);

}

public void optMethod(int first = 1, double second = 0.45, string third = "C#")

{

Console.WriteLine("First {0},Second {1},third {2}", first, second, third);

}

public int optMethod(int a, int b) => a \* b;

//static void Main()

//{

// int value = (new OptionalParameters()).optMethod(2, 4);

//}

}

}

**Chapter -6 Exceptions**

using System;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

namespace MathsOperators

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void calculateClick(object sender, RoutedEventArgs e)

{

try

{

if ((bool)addition.IsChecked)

{

addValues();

}

else if ((bool)subtraction.IsChecked)

{

subtractValues();

}

else if ((bool)multiplication.IsChecked)

{

multiplyValues();

}

else if ((bool)division.IsChecked)

{

divideValues();

}

else

{

throw new InvalidOperationException("No operator selected");

}

}

catch (FormatException fEx)

{

result.Text = fEx.Message;

}

catch (OverflowException oEx)

{

result.Text = oEx.Message;

}

catch (InvalidOperationException ioEx)

{

result.Text = ioEx.Message;

}

catch (Exception ex)

{

result.Text = ex.Message;

}

}

private void addValues()

{

int lhs = int.Parse(lhsOperand.Text);

int rhs = int.Parse(rhsOperand.Text);

int outcome = 0;

outcome = lhs + rhs;

expression.Text = lhsOperand.Text + " + " + rhsOperand.Text;

result.Text = outcome.ToString();

}

private void subtractValues()

{

int lhs = int.Parse(lhsOperand.Text);

int rhs = int.Parse(rhsOperand.Text);

int outcome = 0;

outcome = lhs - rhs;

expression.Text = lhsOperand.Text + " - " + rhsOperand.Text;

result.Text = outcome.ToString();

}

private void multiplyValues()

{

int lhs = int.Parse(lhsOperand.Text);

int rhs = int.Parse(rhsOperand.Text);

int outcome = 0;

outcome = checked(lhs \* rhs);

expression.Text = lhsOperand.Text + " \* " + rhsOperand.Text;

result.Text = outcome.ToString();

}

private void divideValues()

{

int lhs = int.Parse(lhsOperand.Text);

int rhs = int.Parse(rhsOperand.Text);

int outcome = 0;

outcome = lhs / rhs;

expression.Text = lhsOperand.Text + " / " + rhsOperand.Text;

result.Text = outcome.ToString();

}

}

}

**Chapter -7 Classes Example.**

using System;

using System.Collections.Generic;

using System.Text;

namespace Classes

{

class Point

{

private int x, y;

private static int objectCount = 0;

public Point()

{

this.x = -1;

this.y = -1;

objectCount++;

}

public Point(int x, int y)

{

this.x = x;

this.y = y;

objectCount++;

}

public double DistanceTo(Point other)

{

int xDiff = this.x - other.x;

int yDiff = this.y - other.y;

double distance = Math.Sqrt((xDiff \* xDiff) + (yDiff \* yDiff));

return distance;

}

public static int ObjectCount()

{

return objectCount;

}

}

}

**Chapter -8 Ref/out parameter Example**

using System;

using System.Collections.Generic;

using System.Text;

#endregion

namespace Parameters

{

class Program

{

static void doWork()

{

int i = 0;

Console.WriteLine(i);

Pass.Value(ref i);

Console.WriteLine(i);

WrappedInt wi = new WrappedInt();

Console.WriteLine(wi.Number);

Pass.Reference(wi);

Console.WriteLine(wi.Number);

}

static void Main(string[] args)

{

try

{

doWork();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

**Chapter -9 Structs and enums example**

using System;

namespace StructsAndEnums

{

enum Month

{

January, February, March, April,

May, June, July, August,

September, October, November, December

}

struct Date

{

private int year=1900;

private Month month;

private int day=1;

public Date(int ccyy, Month mm, int dd)

{

this.year = ccyy;

this.month = mm;

this.day = dd;

}

//public override string ToString()

//{

// string data = String.Format("{0} {1} {2}", this.month, this.day+1 ,

// this.year+1900);

// return data;

//}

public void AdvanceMonth()

{

this.month++;

if (this.month == Month.December + 1)

{

this.month = Month.January;

this.year++;

}

}

}

}

**Chapter -11 params keyword Example**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ParamsArray

{

class Util

{

public static int Sum(params int[] paramList)

{

Console.WriteLine("Using parameter list");

if (paramList == null)

{

throw new ArgumentException("Util.Sum: null parameter list");

}

if (paramList.Length == 0)

{

throw new ArgumentException("Util.Sum: empty parameter list");

}

int sumTotal = 0;

foreach (int i in paramList)

{

sumTotal += i;

}

return sumTotal;

}

public static int Sum(int param1 = 0, int param2 = 0, int param3 = 0, int param4 = 0)

{

Console.WriteLine("Using optional parameters");

int sumTotal = param1 + param2 + param3 + param4;

return sumTotal;

}

}

class Program

{

static void doWork()

{

// Console.WriteLine(Util.Sum(10, 9, 8, 7, 6, 5, 4, 3, 2, 1));

Console.WriteLine(Util.Sum(2, 4, 6, 8, 10));

}

static void Main()

{

try

{

doWork();

}

catch (Exception ex)

{

Console.WriteLine("Exception: {0}", ex.Message);

}

}

}

}

**Chapter -12 Inheritance example.**

using System;

using System.Text;

using System.Threading.Tasks;

namespace Vehicles

{

class Vehicle

{

public void StartEngine(string noiseToMakeWhenStarting)

{

Console.WriteLine("Starting engine: {0}", noiseToMakeWhenStarting);

}

public void StopEngine(string noiseToMakeWhenStopping)

{

Console.WriteLine("Stopping engine: {0}", noiseToMakeWhenStopping);

}

public virtual void Drive()

{

Console.WriteLine("Default implementation of the Drive method");

}

}

class Car : Vehicle

{

public void Accelerate()

{

Console.WriteLine("Accelerating");

}

public void Brake()

{

Console.WriteLine("Braking");

}

public override void Drive()

{

Console.WriteLine("Motoring");

}

}

class Airplane : Vehicle

{

public void TakeOff()

{

Console.WriteLine("Taking off");

}

public void Land()

{

Console.WriteLine("Landing");

}

public override void Drive()

{

Console.WriteLine("Flying");

}

}

}

class Program

{

static void doWork()

{

Console.WriteLine("Journey by airplane:");

Airplane myPlane = new Airplane();

myPlane.StartEngine("Contact");

myPlane.TakeOff();

myPlane.Drive();

myPlane.Land();

myPlane.StopEngine("Whirr");

Console.WriteLine("\nJourney by car:");

Car myCar = new Car();

myCar.StartEngine("Brm brm");

myCar.Accelerate();

myCar.Drive();

myCar.Brake();

myCar.StopEngine("Phut phut");

Console.WriteLine("\nTesting polymorphism");

Vehicle v = myCar;

v.Drive();

v = myPlane;

v.Drive();

}

static void Main()

{

try

{

doWork();

}

catch (Exception ex)

{

Console.WriteLine("Exception: {0}", ex.Message);

}

}

}

**Chapter -13 Inheritance with Interfaces**.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Media;

using System.Windows.Shapes;

using System.Windows.Controls;

namespace Drawing

{

abstract class DrawingShape

{

protected int size;

protected int locX = 0, locY = 0;

protected Shape shape = null;

public DrawingShape(int size)

{

this.size = size;

}

public void SetLocation(int xCoord, int yCoord)

{

this.locX = xCoord;

this.locY = yCoord;

}

public void SetColor(Color color)

{

if (this.shape != null)

{

SolidColorBrush brush = new SolidColorBrush(color);

this.shape.Fill = brush;

}

}

public virtual void Draw(Canvas canvas)

{

if (this.shape == null)

{

throw new InvalidOperationException("Shape is null");

}

this.shape.Height = this.size;

this.shape.Width = this.size;

Canvas.SetTop(this.shape, this.locY);

Canvas.SetLeft(this.shape, this.locX);

canvas.Children.Add(this.shape);

}

}

interface IColor

{

void SetColor(Color color);

}

interface IDraw

{

void SetLocation(int xCoord, int yCoord);

void Draw(Canvas canvas);

}

class Square : DrawingShape, IDraw, IColor

{

public Square(int sideLength)

: base(sideLength)

{

}

public override void Draw(Canvas canvas)

{

if (this.shape != null)

{

canvas.Children.Remove(this.shape);

}

else

{

this.shape = new Rectangle();

}

base.Draw(canvas);

}

}

}

**Chapter-14 Garbage Collection Exmple.. Dispose() and Distructor**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GarbageCollectionDemo

{

class Calculator : IDisposable

{

private bool disposed = false;

public Calculator()

{

Console.WriteLine("Calculator being created");

}

~Calculator()

{

Console.WriteLine("Calculator being finalized");

this.Dispose();

}

public int Divide(int first, int second)

{

return first / second;

}

public void Dispose()

{

if (!this.disposed)

{

Console.WriteLine("Calculator being disposed");

}

this.disposed = true;

GC.SuppressFinalize(this);

}

}

class Program

{

static void Main(string[] args)

{

using (Calculator calculator = new Calculator())

{

Console.WriteLine("{0} / {1} = {2}", 120, 0, calculator.Divide(120, 0));

}

Console.WriteLine("Program finishing");

}

}

}

**Chapter 15 Properties Example..**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Media;

using System.Windows.Shapes;

using System.Windows.Controls;

namespace Drawing

{

abstract class DrawingShape

{

protected int \_size;

protected int \_x = 0, \_y = 0;

protected Shape shape = null;

public DrawingShape(int size)

{

this.\_size = size;

}

public int X

{

get { return this.\_x; }

set { this.\_x = value; }

}

public int Y

{

get { return this.\_y; }

set { this.\_y = value; }

}

public Color Color

{

set

{

if (this.shape != null)

{

SolidColorBrush brush = new SolidColorBrush(value);

this.shape.Fill = brush;

}

}

}

public virtual void Draw(Canvas canvas)

{

if (this.shape == null)

{

throw new InvalidOperationException("Shape is null");

}

this.shape.Height = this.\_size;

this.shape.Width = this.\_size;

Canvas.SetTop(this.shape, this.\_y);

Canvas.SetLeft(this.shape, this.\_x);

canvas.Children.Add(this.shape);

}

}

interface IColor

{

Color Color { set; }

}

interface IDraw

{

int X { get; set; }

int Y { get; set; }

void Draw(Canvas canvas);

}

class Square : DrawingShape, IDraw, IColor

{

public Square(int sideLength) : base(sideLength)

{

}

public override void Draw(Canvas canvas)

{

if (this.shape != null)

{

canvas.Children.Remove(this.shape);

}

else

{

this.shape = new Rectangle();

}

base.Draw(canvas);

}

}

}

**Chapter-16 Indexers Example.**

using System;

namespace Indexers

{

sealed class PhoneBook

{

private int used;

private Name[] names;

private PhoneNumber[] phoneNumbers;

public PhoneBook()

{

int initialSize = 0;

this.used = 0;

this.names = new Name[initialSize];

this.phoneNumbers = new PhoneNumber[initialSize];

}

public void Add(Name name, PhoneNumber number)

{

enlargeIfFull();

this.names[used] = name;

this.phoneNumbers[used] = number;

this.used++;

}

public Name this[PhoneNumber number]

{

get

{

int i = Array.IndexOf(this.phoneNumbers, number);

if (i != -1)

{

return this.names[i];

}

else

{

return new Name();

}

}

}

public PhoneNumber this[Name name]

{

get

{

int i = Array.IndexOf(this.names, name);

if (i != -1)

{

return this.phoneNumbers[i];

}

else

{

return new PhoneNumber();

}

}

}

private void enlargeIfFull()

{

if (this.used == this.names.Length)

{

int bigger = used + 16;

Name[] moreNames = new Name[bigger];

this.names.CopyTo(moreNames, 0);

PhoneNumber[] morePhoneNumbers = new PhoneNumber[bigger];

this.phoneNumbers.CopyTo(morePhoneNumbers, 0);

this.names = moreNames;

this.phoneNumbers = morePhoneNumbers;

}

}

}

}

**Chapter 17 Binary Tree Example Using generics**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BinaryTree

{

public class Tree<TItem> where TItem : IComparable<TItem>

{

public TItem NodeData { get; set; }

public Tree<TItem> LeftTree { get; set; }

public Tree<TItem> RightTree { get; set; }

public Tree(TItem nodeValue)

{

this.NodeData = nodeValue;

this.LeftTree = null;

this.RightTree = null;

}

public void Insert(TItem newItem)

{

TItem currentNodeValue = this.NodeData;

if (currentNodeValue.CompareTo(newItem) > 0)

{

// Insert the new item into the left subtree

if (this.LeftTree == null)

{

this.LeftTree = new Tree<TItem>(newItem);

}

else

{

this.LeftTree.Insert(newItem);

}

}

else

{

// Insert the new item into the right subtree

if (this.RightTree == null)

{

this.RightTree = new Tree<TItem>(newItem);

}

else

{

this.RightTree.Insert(newItem);

}

}

}

public string WalkTree()

{

string result = "";

if (this.LeftTree != null)

{

result = this.LeftTree.WalkTree();

}

result += String.Format(" {0} ", this.NodeData.ToString());

if (this.RightTree != null)

{

result += this.RightTree.WalkTree();

}

return result;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using BinaryTree;

namespace BinaryTreeTest

{

class Program

{

static void Main(string[] args)

{

Tree<int> tree1 = new Tree<int>(10);

tree1.Insert(5);

tree1.Insert(11);

tree1.Insert(5);

tree1.Insert(-12);

tree1.Insert(15);

tree1.Insert(0);

tree1.Insert(14);

tree1.Insert(-8);

tree1.Insert(10);

tree1.Insert(8);

tree1.Insert(8);

string sortedData = tree1.WalkTree();

Console.WriteLine("Sorted data is: {0}", sortedData);

Tree<string> tree2 = new Tree<string>("Hello");

tree2.Insert("World");

tree2.Insert("How");

tree2.Insert("Are");

tree2.Insert("You");

tree2.Insert("Today");

tree2.Insert("I");

tree2.Insert("Hope");

tree2.Insert("You");

tree2.Insert("Are");

tree2.Insert("Feeling");

tree2.Insert("Well");

tree2.Insert("!");

sortedData = tree2.WalkTree();

Console.WriteLine("Sorted data is: {0}", sortedData);

}

}

}

**Chapter 22 operator Overloading Complex Number Example.**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ComplexNumbers

{

class Complex

{

public int Real { get; set; }

public int Imaginary { get; set; }

IEnumerable

public Complex(int real, int imaginary)

{

this.Real = real;

this.Imaginary = imaginary;

}

public Complex(int real)

{

this.Real = real;

this.Imaginary = 0;

}

public static implicit operator Complex(int from)

{

return new Complex(from);

}

public static explicit operator int(Complex from)

{

return from.Real;

}

public override string ToString()

{

return String.Format("({0} + {1}i)", this.Real, this.Imaginary);

}

public static Complex operator +(Complex lhs, Complex rhs)

{

return new Complex(lhs.Real + rhs.Real, lhs.Imaginary + rhs.Imaginary);

}

public static Complex operator -(Complex lhs, Complex rhs)

{

return new Complex(lhs.Real - rhs.Real, lhs.Imaginary - rhs.Imaginary);

}

public void Display()

{

Console.WriteLine("{0}+{1}i",this.Real,this.Imaginary);

}

public static Complex operator \*(Complex lhs, Complex rhs)

{

return new Complex(lhs.Real \* rhs.Real - lhs.Imaginary \* rhs.Imaginary,

lhs.Imaginary \* rhs.Real + lhs.Real \* rhs.Imaginary);

}

public static Complex operator /(Complex lhs, Complex rhs)

{

int realElement = (lhs.Real \* rhs.Real + lhs.Imaginary \* rhs.Imaginary) /

(rhs.Real \* rhs.Real + rhs.Imaginary \* rhs.Imaginary);

int imaginaryElement = (lhs.Imaginary \* rhs.Real - lhs.Real \* rhs.Imaginary) /

(rhs.Real \* rhs.Real + rhs.Imaginary \* rhs.Imaginary);

return new Complex(realElement, imaginaryElement);

}

public static bool operator ==(Complex lhs, Complex rhs)

{

return lhs.Equals(rhs);

}

public static bool operator !=(Complex lhs, Complex rhs)

{

return !(lhs.Equals(rhs));

}

public override bool Equals(Object obj)

{

if (obj is Complex)

{

Complex compare = (Complex)obj;

return (this.Real == compare.Real) &&

(this.Imaginary == compare.Imaginary);

}

else

{

return false;

}

}

public override int GetHashCode()

{

return base.GetHashCode();

}

}

class Program

{

static void doWork()

{

Complex first = new Complex(10, 4);

Complex second = new Complex(5, 2);

Console.WriteLine("first is {0}", first);

Console.WriteLine("second is {0}", second);

Complex temp = first + second;

temp.Display();

//Console.WriteLine("Add: result is {0}", temp);

temp = first - second;

temp.Display();

// Console.WriteLine("Subtract: result is {0}", temp);

temp = first \* second;

Console.WriteLine("Multiply: result is {0}", temp);

temp = first / second;

Console.WriteLine("Divide: result is {0}", temp);

if (temp == first)

{

Console.WriteLine("Comparison: temp == first");

}

else

{

Console.WriteLine("Comparison: temp != first");

}

if (temp == temp)

{

Console.WriteLine("Comparison: temp == temp");

}

else

{

Console.WriteLine("Comparison: temp != temp");

}

Console.WriteLine("Current value of temp is {0}", temp);

if (temp == 2)

{

Console.WriteLine("Comparison after conversion: temp == 2");

}

else

{

Console.WriteLine("Comparison after conversion: temp != 2");

}

temp += 2;

Console.WriteLine("Value after adding 2: temp = {0}", temp);

int tempInt = (int)temp;

Console.WriteLine("Int value after conversion: tempInt == {0}", tempInt);

}

static void Main()

{

try

{

doWork();

}

catch (Exception ex)

{

Console.WriteLine("Exception: {0}", ex.Message);

}

}

}

}