//C# 3.0

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using ExtensionMethods;

namespace CSC2008CS3

{

delegate int Delegates(int i);

class Program

{

//Also here

//delegate int Delegates(int i);

//2. Implicitly Typed Arrays

object ImplicitlyTypedArray = new[] { 1, 10, 100, 1000 }; //int[]

static void Main()

{

Program P = new Program();

P.Method();

Console.ReadKey();

}

private void Method()

{

/\*var - 1. Implicitly Typed Local Variables\*/

Console.WriteLine("2. Implicitly Typed Arrays ");

var ImplicitlyTypedIntArray = new[] { 1, 10, 100, 1000 }; //int[]

var ImplicitlyTypedStringArray = new[] { "hello", null, "world" }; //string[]

int[,] TwoDimensionArray = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 } };

int[,] ImplicitlyTypedTwoDimensionArray = new[,] { { 1, 2 }, { 3, 4 }, { 5, 6 } };

Console.WriteLine("Print 5: " + ImplicitlyTypedTwoDimensionArray[2, 0]);

//single-dimension jagged array of integers

var ImplicitlyTypedSingleDimensionJaggedArray = new[]

{

new[]{1,2,3},

new[]{4,5,6,7},

};

Console.WriteLine("Print 5: " + ImplicitlyTypedSingleDimensionJaggedArray[1][1]);

//two-dimension jagged array of strings

var ImplicitlyTypedTwoDimensionJaggedArray = new[]

{

new[,]{{"Bill", "Gates"}, {"Anders", "Heilsberg"}},

new[,]{{"Larry", "Page"}, {"Mark", "Zuckerberg"}, {"Jack", "Dorsey"}},

};

Console.WriteLine("To print Zuckerberg: " + ImplicitlyTypedTwoDimensionJaggedArray[1][1, 1]);

Console.WriteLine();

Console.WriteLine("3. Anonymous Types");

//new - 4. Object Initializers

var AnonymousType = new { Name = "Anonymous", Number = 0 };

Console.WriteLine("Name = {0}, Number = {1}", AnonymousType.Name, AnonymousType.Number);

//new - 4. Collection Initializers

//() is optional

//using System.Collections.Generic;

List<Product> Products = new List<Product>()

{

new Product(){Color = "Red", Flag = -1},

new Product(){Color = "Amber", Flag = 0},

new Product(){Color = "Green", Flag = 1},

};

//P - 3. Anonymous Types

//new - 4. Collection Initializers

//using System.Linq;

var Rowset = from P in Products

select new { P.Color, Semaphore = P.Flag };

foreach (var Row in Rowset)

{

Console.WriteLine("Color = {0}, Semaphore = {1}", Row.Color, Row.Semaphore);

}

//P - 3. Anonymous Types //var ImplicitlyTypedLocalVaraibleArray = new[] { 1, 10, 100, 1000 }; //int[]

var AnonymousTypeArray = new[] { new { Fruit = "Apple", Price = 10 }, new { Fruit = "Grape", Price = 5 } };

foreach (var Array in AnonymousTypeArray)

{

Console.WriteLine("Fruit = {0}, Flag = {1:C}", Array.Fruit, Array.Price);

}

for (var Index = 0; Index < AnonymousTypeArray.Length; Index++)

{

Console.WriteLine("Flag = {0:C}, Fruit = {1}, ", AnonymousTypeArray[Index].Price, AnonymousTypeArray[Index].Fruit);

}

Console.WriteLine();

//new - 4. Collection Initializers

var Integers = new List<int>()

{

1, 2, 3, 4, 5,

};

foreach (var Integer in Integers)

{

Console.Write(Integer + " ");

}

Console.WriteLine();

int N = 2;

//new - 4. Collection Initializers

var IntergerList = new List<int> { -2 + 12, 40 / 2, 150 % 40, 10 \* GetSquare(N), 50 };

foreach (var Integer in IntergerList)

{

Console.Write(Integer + " ");

}

Console.WriteLine("\n");

int[] Numbers = { 1, 5, 4, 8, 9, 2, 7, 3, 6 };

Console.WriteLine("5. Extension Methods");

//using System.Linq;

var NumbersOrderBy = Numbers.OrderBy(l => l);

foreach (var i in NumbersOrderBy)

{

Console.Write(i + " ");

}

Console.WriteLine();

string Line = "Hello World!";

Console.WriteLine("Number of words in \"{0}\" = {1}", Line, Line.WordCount());

Console.WriteLine();

Console.WriteLine("6. Lambda Expressions");

Delegates DelegatesObject = x => x \* x;

Console.WriteLine(DelegatesObject(5));

//using System.Linq.Expressions;

Expression<Delegates> ExpressionTreeType = y => y \* y \* y;

// Compiling the expression tree into a delegate.

var ExpressionTree = ExpressionTreeType.Compile();

DelegatesObject = ExpressionTreeType.Compile();

// Invoking the delegate and writing the result to the console.

Console.WriteLine(ExpressionTree(5));

Console.WriteLine(DelegatesObject(5));

Console.WriteLine();

Console.WriteLine("7. Expression Trees");

// Add the following using directive to your code file:

// using System.Linq.Expressions;

// Manually build the expression tree for

// the lambda expression num => num < 5.

ParameterExpression numParam = Expression.Parameter(typeof(int), "num");

ConstantExpression five = Expression.Constant(5, typeof(int));

BinaryExpression numLessThanFive = Expression.LessThan(numParam, five);

Expression<Func<int, bool>> lambda1 = Expression.Lambda<Func<int, bool>>(numLessThanFive, new ParameterExpression[] { numParam });

// Add the following using directive to your code file:

// using System.Linq.Expressions;

// Create an expression tree.

Expression<Func<int, bool>> exprTree = num => num < 7;

// Decompose the expression tree.

ParameterExpression param = (ParameterExpression)exprTree.Parameters[0];

BinaryExpression operation = (BinaryExpression)exprTree.Body;

ParameterExpression left = (ParameterExpression)operation.Left;

ConstantExpression right = (ConstantExpression)operation.Right;

// This code produces the following output:

// Decomposed expression: num => num LessThan 7

Console.WriteLine("Decomposed expression: {0} => {1} {2} {3}", param.Name, left.Name, operation.NodeType, right.Value);

// Creating an expression tree.

Expression<Func<int, bool>> expr = num => num < 5;

// Compiling the expression tree into a delegate.

Func<int, bool> result = expr.Compile();

// Invoking the delegate and writing the result to the console.

Console.WriteLine(result(4)); //Prints True.

// You can also use simplified syntax

// to compile and run an expression tree.

// The following line can replace two previous statements.

Console.WriteLine(expr.Compile()(4)); //Also prints True.

Console.WriteLine();

Console.WriteLine("10. Partial Classes and Methods");

CoOrdinates CoOrds = new CoOrdinates(5, 10);

CoOrds.PrintCoOrdinates();

}

private int GetSquare(int N)

{

return N \* N;

}

}

class Product

{

//8. Automatically Implemented Properties

public string Color

{

get;

set;

}

//8. Automatically Implemented Properties

public sbyte Flag

{

get;

set;

}

}

}

namespace ExtensionMethods

{

static class Extension //Extension methods must be defined in a non-generic static class

{

//5. Extension Methods

public static int WordCount(this string Message)

{

return Message.Split(new char[] { ' ', '!' }, StringSplitOptions.RemoveEmptyEntries).Length;

}

}

}

//10. Partial Classes (Partial Classes Structs and Interfaces were available in C# 2.0) and Methods

//It is possible to split the definition of a class or a struct, an interface or a method over two or more source files.

//Each source file contains a section of the type or method definition, and all parts are combined when the application is compiled.

public partial class CoOrdinates //public partial struct CoOrdinates //All should be structs

{

private int x;

private int y;

public CoOrdinates(int x, int y)

{

this.x = x;

this.y = y;

}

//A partial method must be declared within a partial class or partial struct

//Signatures in both parts of the partial type must match.

//The method must return void.

//No access modifiers or attributes are allowed. Partial methods are implicitly private.

//Can be static - both partial method declarations must be static or neither may be static

//The 'partial' modifier can only appear immediately before 'class', 'struct', 'interface', or 'void'

//A partial method may not have multiple implementing declarations - Max 2 - 1 declaration - MUST (can be called without implementation) and 1 implementation

partial void PartialMedthod();

}

//This part can be in a separate file.

public partial class CoOrdinates //public partial struct CoOrdinates //All should be structs

{

public void PrintCoOrdinates()

{

Console.WriteLine("CoOrds: {0},{1}", x, y);

PartialMedthod();

StaticPartialMedthod();

}

partial void PartialMedthod()

{

Console.WriteLine("This is partial method!");

}

static partial void StaticPartialMedthod();

}

//This part can be in a separate file.

public partial class CoOrdinates //public partial struct CoOrdinates //All should be structs

{

static partial void StaticPartialMedthod()

{

Console.WriteLine("This is static partial method!");

}

}

partial interface IInterface

{

void NonPartailMedthod();

}

//This part can be in a separate file.

partial interface IInterface

{

//Must be different or overloaded

void NonPartailMedthod(string s);

}

/\* Output

2. Implicitly Typed Arrays

Print 5: 5

Print 5: 5

To print Zuckerberg: Zuckerberg

3. Anonymous Types

Name = Anonymous, Number = 0

Color = Red, Semaphore = -1

Color = Amber, Semaphore = 0

Color = Green, Semaphore = 1

Fruit = Apple, Flag = $10.00

Fruit = Grape, Flag = $5.00

Flag = $10.00, Fruit = Apple,

Flag = $5.00, Fruit = Grape,

1 2 3 4 5

10 20 30 40 50

5. Extension Methods

1 2 3 4 5 6 7 8 9

Number of words in "Hello World!" = 2

6. Lambda Expressions

25

125

125

7. Expression Trees

Decomposed expression: num => num LessThan 7

True

True

10. Partial Classes and Methods

CoOrds: 5,10

This is partial method!

This is static partial method!

\*/