Structural programming

Introduction to Computer Programming

Maciej Piernik

Review of Lecture 4

- Arrays
  - Creating arrays
  - Accessing elements
  - Looping through elements
- Reference vs value types

Outline

- Methods: procedures and functions
- Returning values
- Passing parameters
  - Reference vs value types… again…
  - Output parameters
  - Varying number of parameters
  - Default values
- Namespaces
- Structures

Methods

- In C# each instruction is executed in the context of some method.
- A method is a named block of code which consists of a series of instructions.
- The program executes the instructions in the method when the method is called.

Procedures

```
class Program
{
    static void Main(string[] args)
    {
        //Procedure call
        NameOfTheProcedure();
    }

    //Procedure definition
    static void NameOfTheProcedure() //Signature
    {
        //Body
    }
}
```

Procedures – example

```
class Program
{
    static void Main(string[] args)
    {
        HelloWorld();
    }

    static void HelloWorld()
    {
        Console.WriteLine("Hello, World!");
    }
}
```

Procedures with parameters

```
class Program
{
    static void Main(string[] args)
    {
        NameOfTheProcedure(argument1, argument2, ...);
    }

    static void NameOfTheProcedure(type1 parameter1, type2 parameter2, ...)
    {
        type1 variable = parameter1;
    }
}
```

Procedures with parameters Example

```
class Program
{
    static void Main(string[] args)
    {
        Addition(1, 2);
    }

    static void Addition (int a, int b)
    {
        Console.WriteLine("Addition");
        Console.WriteLine(a + " + " + b + " = " + (a + b));
    }
}
```

Functions

```
class Program
{
    static void Main(string[] args)
    {
        type variable = NameOfTheFunction();
    }

    static type NameOfTheFunction()
    {
        return value_of_type_type;
    }
}
```

Functions – example

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Addition");
        Console.WriteLine(Addition(1, 2));
    }

    static int Addition(int a, int b)
    {
        return a + b;
    }
}
```

Reference vs value types

```csharp
static void Main(string[] args)
{
    int i = 0;
    int[] t = { 0 };

    Modify(i);
    Modify(t);

    Console.WriteLine(i);
    Console.WriteLine(t[0]);
}

static void Modify(int a)
{
    a = 5;
}

static void Modify(int[] a)
{
    a[0] = 5;
}
```

Reference vs value types

```
static void Main(string[] args)
{
    int i = 0;
    int[] t = { 0 };

    Modify(ref i);
    Modify(t);

    Console.WriteLine(i);
    Console.WriteLine(t[0]);
}

static void Modify(ref int a)
{
    a = 5;
}

static void Modify(int[] a)
{
    a[0] = 5;
}
```

Output parameters

```
class Program
{
    static void Main(string[] args)
    {
        int x = 1;
        Console.WriteLine(x);
        ModifyingProcedure(out x);
        Console.WriteLine(x);

    }

    static void ModifyingProcedure(out int a)
    {
        a = 23;
    }
}
```

Default values

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine(FunctionWithDefaultValues());
        Console.WriteLine(FunctionWithDefaultValues(2));
        Console.WriteLine(FunctionWithDefaultValues(2, 3));
        Console.WriteLine(FunctionWithDefaultValues(b: 3));
    }

    static int FunctionWithDefaultValues(int a = 1, int b = 23)
    {
        return a + b;
    }
}
```

Varying number of parameters

```
class Program
{
    static void Main(string[] args)
    {
        VaryingParameters();
        VaryingParameters(1);
        VaryingParameters(1, 2);
        VaryingParameters(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        int[] array = new int[] { 1, 2, 3, 4, 5 };
        VaryingParameters(array);
    }

    static void VaryingParameters(params int[] parameters)
    {
        foreach (int param in parameters)
        {
            Console.Write(param + " ");
        }
        Console.WriteLine();
    }
}
```

Classes, namespaces, and libraries

- Classes, namespaces and libraries help us organize our code.
- Classes aggregate related methods.
- Namespaces aggregate related classes.
- Libraries aggregate related namespaces and allow us to use the same methods in different projects.
- The visibility of certain methods can be restricted using access modifiers: `public` and `private`.

Classes mean something completely different in object-oriented programming!

Namespaces and access modifiers

```
namespace ns2
{
    class SomeMethods
    {
        public static void Method1()
        {
            Method2();
        }

        private static void Method2()
        {
        }
    }

    class SomeOtherMethods
    {
    }
}
```

```
using ns2;

namespace ns1
{
    class Program
    {
        static void Main(string[] args)
        {
            SomeMethods.Method1();
            SomeMethods.Method2(); //ERROR
        }
    }
}
```

Structures

- A structure is a value type which aggregates related variables and methods.

```
public struct Rectangle
{
    public int Width;
    public int Height;

    public Rectangle(int w, int h)
    {
        this.Width = w;
        this.Height = h;
    }

    public int Area()
    {
        return this.Height * this.Width;
    }
}
```

```
static void Main(string[] args)
{
    Console.Write("Enter width: ");
    int width = int.Parse(Console.ReadLine());
    Console.Write("Enter height: ");
    int height = int.Parse(Console.ReadLine());

    Rectangle r = new Rectangle(width, height);
    Console.WriteLine("Width: " + r.Width);
    Console.WriteLine("Height: " + r.Height);
    Console.WriteLine("Area: " + r.Area());
}
```

Summary

- Methods: procedures and functions
- Returning values
- Passing parameters
  - Reference vs value types… again…
  - Output parameters
  - Varying number of parameters
  - Default values
- Namespaces
- Structures