# Programming in C#. Fundamentals

SIGMA Software

# *Lesson 2*
# *Classes and their main features*

# Classes and their main features

- Constructors
- Fields and Properties
- Methods
- Events
- Delegates
- Passing Parameters
- Ref and out
- Immutability

# Constructors

Constructors create your object

- Special method in class
- Executed when instance is created
- Named with the class name
- Default constructors has no parameters
- Not required
- No return value (not even Null)

```
Public class Worker{

  public Worker(int age, string name){
    Age = age;
    Name = name;
  }
}
```

# Parameterized Constructors

- Defines parameters to initialize the instance
- Constructor overloading
- Use “this” to invoke another constructors
- Constructors chaining
- Minimizes repeated code

```
public class Worker {

    public Worker(int age, string name) {
        Age = age;
        Name = name;
    }

    // explicit default constructor
    public Worker(){
        Office = 101;
    }
```

# Default Constructor

A constructor with no parameters.

```
public class Worker {

    public string Name { get; } = "Jesse Liberty";

    public int Age { get; } = 40;

    private double salary;
    public double Salary {
        get { return salary; }
        set { salary = value; }
    }

 Worker worker = new Worker();
```

If you don't provide any constructors,
a default constructor is provided for you.

# Fields

Fields are variables of a class

Can hold:

- A built-in value type
- A class instance (a reference)
- A struct instance (actual data)
- An array of class or struct instances (an array is actually a reference)
- An event

# Readonly Fields

Similar to a const, but is initialized at run-time in its declaration or in a constructor

- Once initialized, it cannot be modified

Differs from a constant

- Initialized at run-time (vs. compile-time)
  - Don’t have to re-compile clients
- Can be static or per-instance

# Properties

- A property can define a get and/or set accessor.
- A function body is executed when a property is accessed.
- We can validate a value, limit it to bounds, or even change its internal data type all transparent to the user.

```
private string name;

public string Name{
   get { return  name; }
   set {  name  = value; }
}
```

```
public string Name { get; set; }
```

```
public string Name { get; }
```

```
public string Name { get; } = "Jesse Liberty";
```

# Methods

Methods define behaviour

- Every method has a return type (void if no value returned)
- Every method has zero or more parameters
- Use params keyword to accept a variable number of parameters
- Every method has a signature
- Name of method + parameters

```
public class Employee {
    public int CalculateSum(int firstValue, int secondValue){
        return firstValue + secondValue;
    }
}


int sum = CalculateSum(5, 7);
```

# Method Overloading

Two methods with the same name

```
public double computeSalary(double start){}
public double computeSalary(double start, double bonus){}
public double computeSalary(double start, string level){}
```

## Two Or More Methods May Have The Same Name As Long As...

**They differ in the number of parameters and/or**

**They differ in the type of parameters**

# Methods - Review

## Instance methods versus static methods

- Instance methods invoked via objects, static method via type

## Abstract methods

- Provide no implementation, implicitly virtual

## Virtual methods

- Can override in a derived class

## Partial methods

- Part of a partial class

## Extensions methods

- Add methods to existing types without creating a new derived type

# Ref and out

C# supports call by value and reference for value types.

Reference types can only be passed by reference.

The **ref** and **out** keywords are used to specify a value parameter is to be passed by reference

The difference is that an **out** parameter does not have to have an initial value when the method is called

```
int ValueTypeByOut(out int inputValue)
{
    return inputValue = 1000;
}
int ValueTypeByRef(ref int inputValue)
{
    return inputValue = inputValue + 1000;
}

int NormalCall(int inputValue)
{
    return inputValue = inputValue + 9999;
}
```

# Passing Parameters

Passing a value variable by default refers to the Pass by Value behavior as in Java

```
public static void foo(int a)
{
      a=1;
}

static void Main(string[] args)
{
      int x=3;
      foo(x);
      Console.WriteLine(x);
}
```

This outputs the value of 3 because x is passed by value to method foo, which gets a copy of x's value under the variable name of a.

# Q & A

# *Practice*
# *Lesson 2*

# *Home work*