C# Programming in .NET (code CS-F)

C# and the .NET framework
- The .NET Framework
- Common Language Runtime (CLR)
- What is MSIL
- .NET Framework Class Library (FCL)
- Where does C# fit in
- C# as an object-oriented language
- Assemblies
- Language interop

C# Language
- The start of the application
- Variables and types
- Value types and reference types
- Copying and comparing reference types
- Strings and arrays
- Operators and their precedence
- The Console class
- String formatting
- Statements and flows
- Command-line arguments

Classes & Objects
- Abstract Data Types
- Objects and classes
- Creating and using your own classes
- Data members and member methods
- Accessibility levels
- Constructors
- Method overloading
- *This* keyword

More on classes
- Properties
- Static variables, methods & classes
- Extention Methods
- Object destruction & finalizers
- *ref* and *out* parameters
- Variable length argument lists
- Named and optional arguments
- Constant values
- Enumerations
- Nullable types & coalescing operator
- *var* variables

Inheritance and Polymorphism
- The root of all classes
- Creating derived classes
- Method overriding and hiding
- Polymorphism and virtual functions
- Casting objects
- Abstract classes
- Sealed classes & methods
- More access specifiers

Namespaces, Nested Classes and Conversions
- Why using namespaces
- Using classes in a namespace
- Placing classes in a namespace
- Nested namespaces
- Aliases
- Using assemblies
- Nested classes
- Partial classes
- Implicit conversions and member lookup
- Explicit conversions
- Checked conversions

Interfaces
- What is an interface?
- Creating, implementing and using interfaces
- Interfaces and properties
- The *is* and *as* operators
- Interfaces versus abstract classes
- Cloning objects using the *ICloneable* interface
- Comparing objects using the *Equals* method
- *IDisposable* interface and *using* statement
- Explicit interface implementation
- Implementing *ICloneable* as explicit interface

Structs
- User defined value types
- Structs versus classes
- Boxing and unboxing
- Object Initializers

Operator Overloading
- What is operator overloading?
- Overloading binary operators
- Comparing objects using overloaded == and != operators
- Overloading unary operators
- Prefix and postfix operators
- *true* and *false* operators
- User defined conversions
- Indexers
- Guidelines

Exception Handling

- What are exceptions
- Exceptions in C#
- Build-in exception classes and their members
- Catching exceptions: *try ... catch*
- *finally*
- Nesting try blocks
- Throwing exceptions
- Creating your own exception classes
- Chaining exceptions

Delegates and Events

- Loose coupling using interfaces: *Strategy pattern*
- Delegates: safe function pointers
- Loose coupling using delegates: *Strategy pattern*
- Multicast delegates
- Events
- Publisher-subscribe idiom
- Model-view / observer pattern
- Defining and raising events
- Create event handlers and subscribe
- Anonymous methods

Introduction to the .NET Framework Class Library

- Framework namespaces
- Basic framework functionality and interfaces
- Array sorting and searching
- Mathematics
- Collections: *ArrayList* and *Hashtable*
- Enumerators and *foreach*

Introduction to Windows Forms

- Windows forms library
- Forms and controls
- Creating simple GUI by hand
- Event handling
- Common Dialog Boxes
- GDI+

Introduction to Windows Presentation Foundation

- What is WPF
- XAML
- Code behind files
- Controls
- Graphics

Introduction to Generic Programming

- Traditional .NET object data structures
- Concrete type wrapper classes
- Generic .NET datastructures
- Collection initializers
- Creating generic classes
- Templates versus generics
- Default values
- Multiple generic types
- Generic type alias & *var* variables
- Generic derivation- and constructor constraints
- Generic methods
- Generic delegates and events
- Strategy pattern with generics

Deployment

- Deploying your application to the end user
- CAB setup
- Microsoft Installer (MSI) setup
- Merge modules
- Web setup

Your Trainer

Daniel J. Duffy started the company Datasim in 1987 to promote C++ as a new object-oriented language for developing applications in the roles of developer, architect and requirements analyst to help clients design and analyse software systems for Computer Aided Design (CAD), process control and hardware-software systems, logistics, holography (optical technology) and computational finance. He used a combination of top-down functional decomposition and bottom-up object-oriented programming techniques to create stable and extendible applications (for a discussion, see Duffy 2004 where we have grouped applications into domain categories). Previous to Datasim he worked on engineering applications in oil and gas and semiconductor industries using a range of numerical methods (for example, the finite element method (FEM)) on mainframe and mini-computers.

Daniel Duffy has BA (Mod), MSc and PhD degrees in pure and applied mathematics and has been active in promoting partial differential equation (PDE) and finite difference methods (FDM) for applications in computational finance. He was responsible for the introduction of the Fractional Step (Soviet Splitting) method and the Alternating Direction Explicit (ADE) method in computational finance. He is also the originator of the exponential fitting method for time-dependent partial differential equations.

He is also the originator of two very popular C++ online courses (both C++98 and C++11/14) on www.quantnet.com in cooperation with Quantnet LLC and Baruch College (CUNY), NYC. He also trains developers and designers around the world. He can be contacted dduffy@datasim.nl for queries, information and course venues, in-company course and course dates