Advanced C# for Computational Finance and Derivatives' Pricing) (code CSCF)

Part I: Essential C#

Overview of the .NET framework

- Type Safety and Memory Management
- Classes and Interfaces
- Namespaces and their use in .NET
- Common Language Runtime (CLR)

Object-Oriented Programming in C#

- Classes, fields and methods
- Pass-by-value and pass-by-reference
- Constructors and object initializers
- Structs
- Properties

Advanced Classes

- Inheritance
- Polymorphism and casting
- Virtual function members
- Abstract classes and abstract methods
- Boxing and unboxing

Interfaces and Components

- Interfaces and contracts
- Implementing interfaces; the different scenarios
- Should we use an abstract class or an interface?
- Why interfaces are essential for applications

Delegates

- Motivation: function pointers and callbacks
- Delegate types and delegate instances
- Writing *plug-in methods* with delegates
- Multi-cast delegates
- Delegates versus interfaces

Events

- Broadcasts and notification patterns
- Event accessors and modifiers
- Events and Windows programming
- Anonymous methods

Generics in C#

- Generic types
- Generic methods and parameters
- Generic delegates

Advanced Generics

- Generic constraints
- Subclassing of generic types
- Self-referencing generic declarations
- Generic events

Application in .NET, Part 1

- One-factor Monte Carlo option pricing
- Scoping the system
- System and component interfaces
- System Decomposition

Application in .NET, Part 2

- Choosing between interfaces and delegates
- Implementing classes
- Design and system patterns (*Builder*, *Mediator*)
- Creating a single-threaded solution
- Multi-threaded solution

Part II: Core Libraries and their Applications in Finance

Fundamental Data Structures

- Strings and text handling
- Dates and Times; time zone
- Financial dates: day-count convention
- Formatting and parsing
- Regular expressions

Collections

- Defining and iterating in containers
- The Array Class
- Linked lists
- Hash tables and sorted dictionaries

Creating your own Containers

- Combining inheritance and generics
- Composition as an alternative to inheritance
- Data structures: vectors and numeric matrices
- Multi-dimensional data structures
- Applications to finance

Streams and I/O

- Stream architecture
- Stream class' members
- File, memory and pipe streams
- Stream adapters

Serialization
- Serialization engines
- Data contract serializer
- Binary serializer
- XML serializer
- Creating and applying serializers

Design by Contract
- Background (Eiffel Programming language)
- Supplier and client: rights and responsibilities
- What is reliable and correct software?
- Correctness and Hoare triples
- Preconditions, postconditions and assertions
- Imperative and declarative statements

Code Contracts in .NET
- Overview of Code Contract
- The binary rewriter
- The Contract class
- Implementing contract by design
- Contracts on interfaces and abstract classes
- Dealing with contract failure

Disposal and Garbage Collection(GC)
- IDisposable, Dispose and Close
- Finalisers
- Automatic garbage collection
- GC internals
- Memory leaks

Part III: Design and Integration Techniques

Threads and Parallel Programming
- A quick introduction to multi-threading concepts
- Thread class
- Synchronisation
- Asynchronous delegates
- Locking
- Thread safety

Assemblies
- No more DLL hell!
- The structure of an assembly
- Private and shared assemblies
- Global Assembly Cache (GAC)
- Assemblies and their importance for component design

Reflection and Metadata
- What is reflection and why is it useful?
- Reflection applied to types
- Reflection applied to assemblies
- Reflection and member invocation
- Application configuration
- Dynamic code generation

Part IV: Creating Computational Finance Applications

Threads and Data
- Implementing the shared data metaphor
- Passing data to a thread
- Mutex and Semaphore in C#
- Thread safety

Synchronisation I
- Blocking methods
- Sleep and EndInvoke
- Join
- Spinning

Synchronisation II
- Locking
- Mutex and Semaphore, again
- Nested locking
- Locking and atomicity

Parallel design patterns
- SPMD pattern
- Master/Worker pattern
- Loop parallelism pattern
- Shared data and shared queues patterns

C# and Design Patterns
- Why design patterns and which ones to use
- Implementing patterns in C#
- Ready-made patterns in C#
- Applications in finance

Major Patterns
- Builder and Factory Method
- Adapter, Bridge
- Visitor, Strategy, Template Method
- Builder

Monte Carlo Engine in C#
- Modular decomposition
- Design of engine (Produce-consumer)
- Random number generators
- Parallel programming

Finite Difference Method (FDM)
- One-factor models
- Plain and barrier options
- Early exercise features
- The Crank Nicolson method
- Comparing FDM with trinomial method

Alternating Direction Explicit (ADE) Method
- Background and motivation
- ADE for one-factor models
- ADE for nonlinear pricing models
- Advantages of ADE

Two-Factor Model
- ADI and Splitting Methods
- Craig-Sneyd method
- Mixed derivatives and Janenko method
- ADE for two-factor problems

Fixed Income Application

- Creating a structured product in C#
- Class design
- Bootstrapping and yield curves
- Linear and cubic spline interpolation
- Worksheet functions
- Add-ins

Monte Carlo Simulator

- Class Design
- Random number generators
- Stochastic equations and finite difference methods
- Using multi-threading to improve performance

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