Platforma .NET și limbajul de programare C#

### Modulul 1. Platforma .NET. Istoria și noțiuni introductive

#### Introducere în platforma .NET

1. Istoria și etapele dezvoltării tehnicilor de programare

* Programarea nestructurată - (un program simplu, ce utilizează numai variabile globale); complicaţiile apar când prelucrarea devine mai amplă, iar datele se multiplică şi se diversifică.
* Programarea procedurală - (program principal deservit de subprograme cu parametri formali, variabile locale şi apeluri cu parametri efectivi); se obţin avantaje privind depanarea şi reutilizarea codului şi se aplică noi tehnici privind transferul parametrilor şi vizibilitatea variabilelor; complicaţiile apar atunci când la program sunt asignaţi doi sau mai mulţi programatori care nu pot lucra simultan pe un acelaşi fişier ce conţine codul sursă.
* Programarea modulară - (gruparea subprogramelor cu funcţionalităţi similare în module, implementate şi depanate separat); se obţin avantaje privind independenţa şi încapsularea (prin separarea zonei de implementare, păstrând vizibilitatea numai asupra zonei de interfaţă a modulului) şi se aplică tehnici de asociere a procedurilor cu datele pe care le manevrează, stabilind şi diferite reguli de acces la date şi la subprograme. Se observă că modulele sunt ”centrate” pe proceduri, acestea gestionând şi setul de date pe care le prelucrează (date+date1 din figură). Daca, de exemplu, dorim să avem mai multe seturi diferite de date, toate înzestrate comportamental cu procedurile din modulul module1, această arhitectură de aplicaţie nu este avantajoasă.
* Programarea orientată obiect - (programe cu noi tipuri ce integrează atât datele, cât şi metodele asociate creării, prelucrării şi distrugerii acestor date); se obţin avantaje prin abstractizarea programării (programul nu mai este o succesiune de prelucrări, ci un ansamblu de obiecte care prind viaţă, au diverse proprietăţi, sunt capabile de acţiuni specifice şi care interacţionează în cadrul programului); intervin tehnici noi privind instanţierea, derivarea şi polimorfismul tipurilor obiectuale.

1. Platforma Microsoft .NET. Motivele și istoria apariției.

.NET este un framework de dezvoltare software care permite scrierea, distribuirea şi rularea aplicaţiilor Windows, WEB și mobile.

Platforma .NET a început să fie dezvoltată la sfârșitul anilor 90 în cadrul unui proiect intern Microsoft cu denumirea Next Generation Windows Services. În 2001 a fost lansată versiunea beta apoi în Februarie 2002 - versiunea completă.

La etapa inițială C# era foarte similar cu Java(o mulțime din aceste similarități s-au păstrat până în prezent), din motiv că scopurile creării limbajului erau de a crea: ”un limbaj de programare simplu, modern, orientat pe obiect care să poată fi utilizat pentru o multitudine de scopuri, iar Java reprezenta toate aceste standarde în acea perioadă.

1. Avantajele și dezavantajele platformei

**Avantaje:**

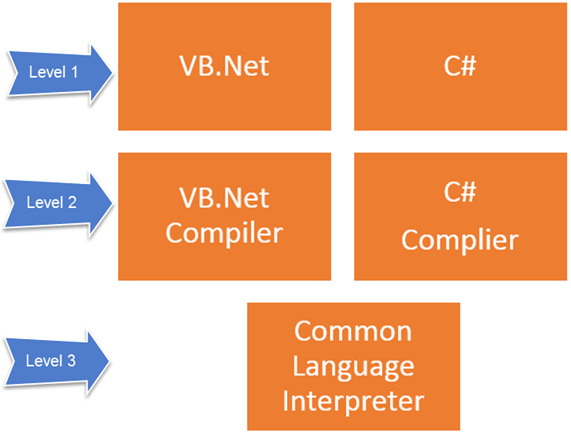
* Orientat pe obiect
* Alocarea memoriei automată
* Cross platform
* Foarte scalabil
* Comunitate largă(multă informație) - locul 2 framework după Node.js
* Librărie extinsă de clase
* MS Intermediate Language
* Compatibilitate între diferite versiuni .NET
* Gamă largă de servicii disponibile(aplicații desktop, aplicații web, suport pentru cloud features etc.)

**Dezavantaje:**

* Dependența de Microsoft (nu este un limbaj open-source)
* Alocarea memoriei automată (nu oferă posibilitatea de a gestiona complet alocarea și curățarea memoriei)

#### Noțiuni introductive în platforma .NET

1. Arhitectura platformei Microsoft .NET



Nivelul 1 - cod(C#, F#, Visual Basic)

Nivelul 2 - compilator( individual pentru orice limbaj de programare)

Nivelul 3 - CLR(common language runtime). Nivelul final care are funcția de a face legătura dintre nivelul 1 și 2 cu funcțiile low-end și executa și derula programul scris.

1. Mediul de execuție universal CLR (common language runtime)

**Mediul de execuție universal (Common Language Runtime, CLR)** - este un mediu de execuție care face parte din platforma .NET. CLR gestionează execuția programelor scrise în diferite limbaje care sunt acceptate în .NET, apoi transformă acest cod în **CIL(Common Intermediate Language - Limbaj universal intermediar)** care poate fi citit de calculator.

1. Sistema standardă de tipuri CTS (common type system)

Sistema standardă de tipuri (Common Type System, CTS), definește cum sunt tipurile de date declarate, folosite și gestionate de CRL(Mediul de execuție universal), ceea ce o face vitală pentru funcționarea mediului de execuție universal.

Funcțiile CTS:

* Stabilește framework-ul necesar pentru combinarea mai multor limbaje de programare, asigură performanța și previne erorile în cadrul combinării acestora.
* Definește regulile de interacțiune între diferite limbaje pentru a asigura funcționarea corectă

1. Specificația standardă de limbaje CLS (common language specification)

Specificația standardă de limbaje definește regulile de combinare a diferitor limbaje de programare. (Pentru a evita neînțelegerea, CTS răspunde tipurile standarde în timp ce CLS de regulile de interacțiune între aceste tipuri).

1. Biblioteca claselor .NET:
2. FCL - Framework Class Library(clasa librăriilor de framework)
3. BCL - Base Class Library(clasa librăriilor de bază)

Platforma .NET conține o multitudine de librării care conțin clasele necesare pentru funcționarea programelor scrise.

**FCL sau clasa librăriilor de framework** reprezintă ansamblul complet de biblioteci necesare pentru realizarea aplicațiilor în .NET.

Părțile componente ale Clasei librăriilor de framework sunt:

* BCL, clasa librăriilor de bază
* LINQ
* XML
* ASP.Net
* etc

**BCL sau Clasa librăriilor de bază**, precum reiese din denumire conține librăriile fundamentale și cele mai simple, precum: System.String, System.DateTime, System.Diagnostics , System.Globalization, System.Resources , System.Text , System.Runtime.Serialization and System.Data etc.

1. Limbajele platformei Microsoft .NET

* **C#**
* **Visual Basic .NET**
* **C++**
* **J#**
* **JScript .NET**
* **Windows PowerShell**
* **IronPython (Implementarea .NET a limbajului Python)**
* **IronRuby (Implementarea .NET a limbajului Ruby)**
* **F#**

**Alte limbaje adiționale pot fi folosite de asemenea prin integrarea unor librării externe.**

1. Limbajul Intermediar Microsoft (MSIL, Microsoft Intermediate Language)

Limbajul intermediar Microsoft sau MSIL este un set de instrucțiuni care nu depind de platforma pe care se derulează și sunt generate de un compilator specific în dependență de limbajul codului scris de programator.

De exemplu codul scris în C# va fi procesat de compilatorul C# și transformat în limbaj intermediar.

Limbajul intermediar permite platformei .NET să compileze o multitudine de limbaje și să le combine în același set de instrucțiuni care sunt executate comun, fără conflicte și eficient.

1. Schema compilației și execuției aplicațiilor .NET:
2. Compilarea codului scris în Microsoft Intermediate Language (MSIL sau IL).
3. CLR include o maşină virtuală (asemănătoare cu o maşină Java), ce execută instrucţiunile limbajului intermediar.
4. Se folosește un compilator care analizează codul intermediar corespunzător şi produce codul maşină adecvat şi eficient.
5. Codul mașină este rulat și rezultatele prezentate utilizatorului.

#### Reflecție și Dotfuscatoare

1. Reflecție.

Reflection provides objects (of type [Type](https://docs.microsoft.com/en-us/dotnet/api/system.type)) that describe assemblies, modules, and types. You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties. If you are using attributes in your code, reflection enables you to access them. For more information, see [Attributes](https://docs.microsoft.com/en-us/dotnet/standard/attributes/).

1. Dotfuscator.

**Dotfuscator** is a tool performing a combination of code [obfuscation](https://en.wikipedia.org/wiki/Obfuscation_(software)), [optimization](https://en.wikipedia.org/wiki/Program_optimization), shrinking, and hardening on [.NET](https://en.wikipedia.org/wiki/.NET_Framework), [Xamarin](https://en.wikipedia.org/wiki/Xamarin) and [Universal Windows Platform](https://en.wikipedia.org/wiki/Universal_Windows_Platform) apps. Ordinarily, .NET executables can easily be [reverse engineered](https://en.wikipedia.org/wiki/Reverse_engineering) by free tools (such as ILSpy, dotPeek and JustDecompile), potentially exposing algorithms and [intellectual property](https://en.wikipedia.org/wiki/Intellectual_property) (trade secrets), licensing and security mechanisms. Also, code can be run through a debugger and its data inspected. Dotfuscator can make all of these things more difficult.

Dotfuscator was developed by PreEmptive Solutions. A free version of the .NET Obfuscator, called the Dotfuscator Community Edition, is distributed as part of Microsoft's [Visual Studio](https://en.wikipedia.org/wiki/Visual_Studio).[[1]](https://en.wikipedia.org/wiki/Dotfuscator#cite_note-1)[[2]](https://en.wikipedia.org/wiki/Dotfuscator#cite_note-2)[[3]](https://en.wikipedia.org/wiki/Dotfuscator#cite_note-3)[[4]](https://en.wikipedia.org/wiki/Dotfuscator#cite_note-4)[[5]](https://en.wikipedia.org/wiki/Dotfuscator#cite_note-5) However, the current version is free for personal, non-commercial use only.