**Dossier Présentation :**

**C#**





**Baptiste DELPIERRE**

**Philémon ROUBAUD**

**Louis HOCHART**

**Vincent DUMONT**

**Année 2018/2019**

Sommaire :

**Présentation du C#3**

Introduction au langage3

Historique du langage3

Raisons de sa création4

**Spécificités du langage5**

Indexeur3

Expression Lambda3

Délégué4

Différences entre le JAVA et le C#3

**Conclusion9**

Quel est le but final du C# ? 9

Qu’est ce qui est codé en C# ?9

Bibliographie et sources9

Lexique9

**Présentation du C#**

**Introduction au langage :**

Le C# est un langage de programmation **orienté objet**, commercialisé par Microsoft depuis 2002 et destiné à développer sur la plateforme Microsoft .NET.

Il est dérivé du C++ et très proche du Java dont il reprend la syntaxe générale ainsi que les concepts, y ajoutant des notions telles que la surcharge des **opérateurs**, les **indexeurs** et les **délégués**. Il est utilisé notamment pour développer des applications web sur la plateforme ASP.NET ainsi que des applications de bureau, des services web, des commandes, des widgets.

**Historique du langage :**

James Gosling, qui a créé le langage de programmation Java en 1994, et Bill Joy, cofondateur de Sun Microsystems, l'initiateur de Java, ont appelé C # une "imitation" de Java; Gosling a ajouté que "C # est une sorte de Java dont la fiabilité, la productivité et la sécurité ont été supprimées". Klaus Kreft et Angelika Langer (auteurs d'un livre sur les flux C ++) ont déclaré dans un article de blog que "Java et C # sont des langages de programmation presque identiques. Répétition ennuyeuse sans innovation "," 23 personne ne prétendra que Java ou C # sont des langages de programmation révolutionnaires qui ont changé la façon dont nous écrivons des programmes "et" C # a beaucoup emprunté à Java - et vice-versa Maintenant que C # prend en charge la boxe et le déballage, nous avons une fonctionnalité très similaire en Java. " En juillet 2000, Hejlsberg a déclaré que C # n'était" pas un clone Java "et était" beaucoup plus proche du C ++ ".

Lors du développement du .NET Framework, les bibliothèques de classes étaient initialement écrites à l'aide d'un système de compilation de code appelé Simple Managed C (SMC). En janvier 1999, Anders Hejlsberg a formé une équipe pour créer un nouveau langage appelé Cool, qui correspond à "langage **orienté objet** de type C". [20] Microsoft avait envisagé de conserver le nom "Cool" comme nom définitif de la langue, mais avait choisi de ne pas le faire pour des raisons de marque. Au moment de l'annonce publique du projet .NET lors de la conférence des développeurs professionnels de juillet 2000, le langage avait été renommé C # et les bibliothèques de classes et le runtime ASP.NET avaient été portés en C #.

Hejlsberg est le principal concepteur et architecte principal de C # chez Microsoft. Dans des entretiens et des documents techniques, il a déclaré que les défauts de la plupart des principaux langages de programmation (par exemple, C ++, Java, Delphi et Smalltalk) étaient à la base des principes fondamentaux du **Common Language Runtime** (CLR), ce qui a ensuite conduit la conception du langage C # lui-même.

**Raison de sa création :**

La norme ECMA répertorie ces objectifs de conception pour C #:

* Le langage se veut un langage de programmation simple, moderne, polyvalent et **orienté objet**.
* Le langage et ses implémentations devraient prendre en charge les principes de l’ingénierie logicielle tels que la vérification de **type forte**, la vérification des limites de tableau, la détection des tentatives d’utilisation de variables non initialisées et la collecte automatique des déchets. La robustesse du logiciel, la durabilité et la productivité du programmeur sont importantes.
* La portabilité est très importante pour le code source et les programmeurs, en particulier ceux qui sont déjà familiarisés avec C et C ++.
* C # est conçu pour être adapté à l'écriture d'applications pour les systèmes hébergés et embarqués, des très grands systèmes utilisant des systèmes d'exploitation sophistiqués aux très petits systèmes dotés de fonctions dédiées.
* Bien que les applications C # soient censées être économiques en termes de mémoire et de puissance de traitement, le langage n'était pas destiné à concurrencer directement les performances et la taille avec le langage C ou le langage assembleur.

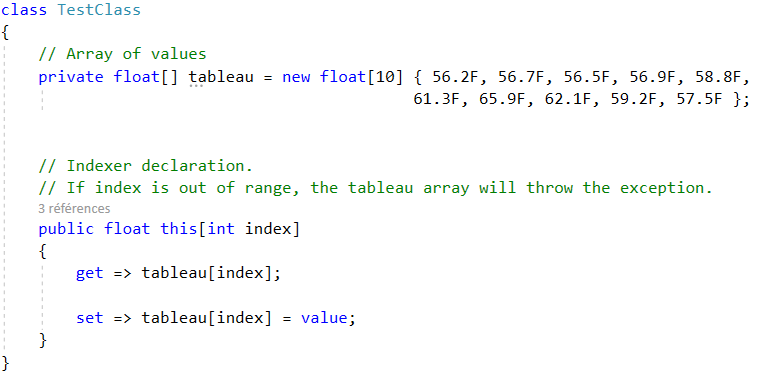
**Spécificité du Langage**

**Indexeur :**

Les indexeurs simplifient la création d’une classe, d’un struct ou d’une interface auxquels les applications clientes peuvent accéder exactement comme à un tableau. Le plus souvent, les indexeurs sont implémentés dans les types dont l’objectif premier est d’encapsuler une collection ou un tableau interne.

Prenons un exemple avec une classe *TestClass*.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Les indexeurs permettent aux objets d’être indexés d’une manière similaire aux tableaux.

L’indexeur est défini à l’aide du mot clé *this*.

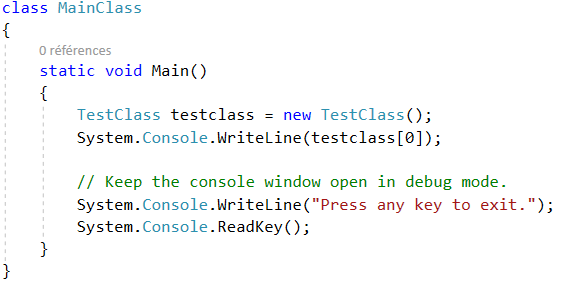
L’indexeur est associé aux accesseurs :

* *get* qui retourne une valeur
* *set* qui affecte la valeur *value*.

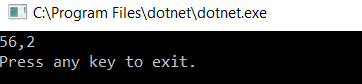
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

L’utilisateur peut donc accéder au tableau *tableau* en utilisant directement l’opérateur [] sur l’instance de l’objet TestClass, sans avoir à interroger directement le tableau.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

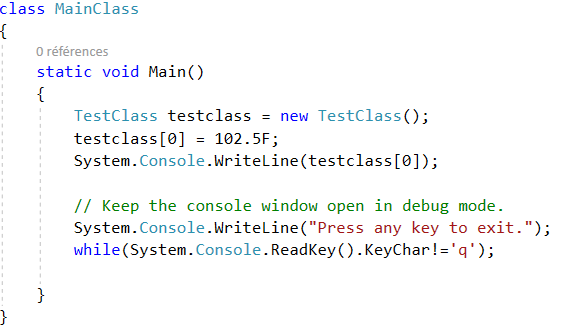
.

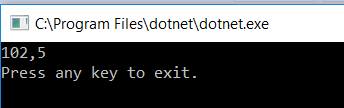
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



Nous obtenons bien la première valeur du tableau *tableau* en appelant *testclass[0]*.

Nous pouvons procéder de même en utilisant l’accesseur *set*.



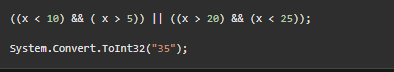


**Expression Lambda :**

Tout d’abord, qu’est-ce qu’une expression ?

Une expression est une séquence d’une ou plusieurs opérandes et opérateurs (ou non) qui peuvent être évalués à une valeur, un objet une méthode ou un espace de noms uniques. Une valeur littérale, une invocation de méthode, une opération, un simple nom de variable sont tous des expressions.

Exemples :



Les expressions peuvent avoir des valeurs de tout type, par exemple, si x et y sont deux int, l’expression x+y est un int, « new MyClass() » est une référence à une nouvelle instance de MyClass.

Les expressions lambdas sont donc une catégorie d’expressions représentant des méthodes anonymes qu’on peut utiliser pour créer des délégués. En utilisant les expressions lambdas, on peut créer des fonctions locales pouvant être passés en paramètres d’autres fonctions.

Voici quelques exemples d’expression lambda :

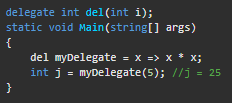
A gauche de la flèche se trouve les paramètres de la méthode, et à droite l’opération effectuée et donc la valeur qui sera retournée lorsqu’on appellera cette méthode.

Lorsque nous voulons préciser qu’il n’y a pas de paramètre, il suffit d’écrire () avant la flèche (exemple 2).

Il peut également y avoir plusieurs paramètres (exemple 3), il est également possible de spécifier les types des paramètres (exemple 4).

Pour résumer, une expression lambda est une méthode locale effectuant des opérations simples.

Une expression lambda permet de créer des délégués.



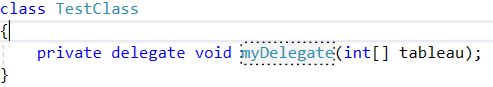
**Délégué :**

Les délégués (« delegate ») en anglais permettent de créer des variables spéciales.

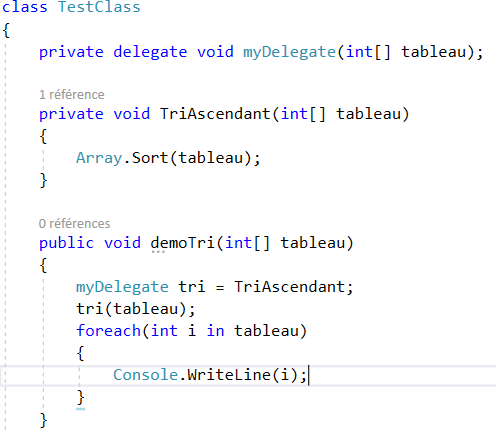
Le délégué permet en quelque sorte de définir la signature d’une méthode et le programmeur peut donc, avec ce délégué, pointer vers n’importe quelle méthode qui respecte cette signature.

Un délégué est en général utilisé lorsque le programmeur veut passer une méthode en paramètre d’une autre méthode.

Un délégué est déclaré en utilisant le mot clé *delegate* entre la visibilité et le type de retour de la méthode.



Ici, un délégué privé est créé dans la classe *TestClass* et il permettra de pointer vers des méthodes qui ne retourne rien (*void*) et qui acceptent un tableau d’entier en paramètre.



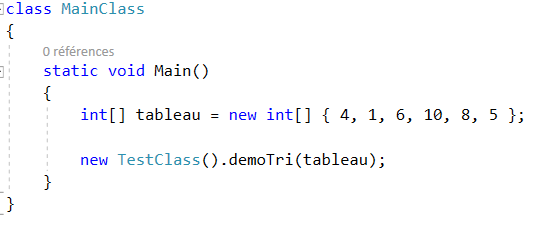
Dans ce code, nous avons implémenté une méthode *private void TriAscendant(int[] tableau)*, qui a la même signature que notre délégué.

Dans la méthode demoTri, nous commençons par déclarer une variable du type du délégué *myDeleguate*, qui est le délégué que nous avons créé.

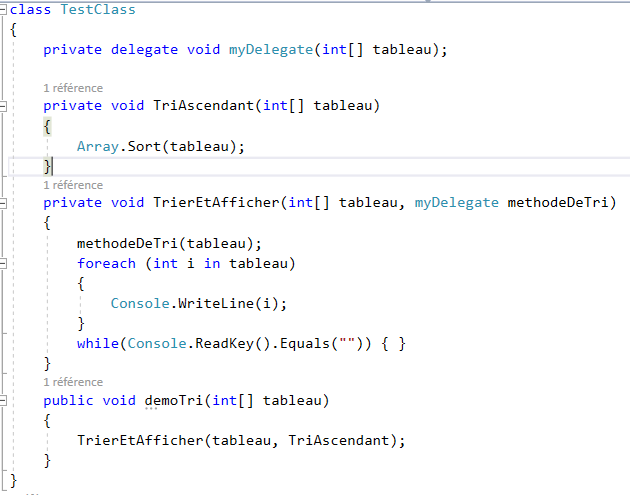
Puis nous faisons pointer cette variable vers la méthode *TriAscendant*.

On exécute la fonction Main() et on obtient bien le tableau trié.





L’intérêt ici est que le délégué est très souple et va permettre de réorganiser le code (on parle également de **refactoriser du code**).



En ajoutant la méthode *TrierEtAfficher*, qui prend un paramètres le tableau et un délégué du type de celui créé précédemment, nous simplifions grandement la méthode *demoTri.*

Nous avons passé le délégué en paramètre, il est ensuite utilisé pour invoquer la méthode que nous lui auront passé (ici, TriAscendant) dans demoTri

À noter que le fait d’utiliser le mot-clé delegate revient en fait à créer une classe qui dérive de *System.Delegate* et qui implémente la logique de base d’un délégué. Le C# nous masque tout ceci afin d’être au maximum efficace.

Important :

* Un délégué est en général utilisé lorsque le programmeur veut passer une méthode en paramètre d’une autre méthode.
* Un délégué est déclaré en utilisant le mot clé *delegate* entre la visibilité et le type de retour de la méthode.
* L’intérêt d’un délégué est qu’il est très souple et va permettre de réorganiser le code.

**Différence entre le JAVA et le C# :**

C# et Java sont des langages similaires, ils sont [**typé**](https://en.wikipedia.org/wiki/Type_system)**s statiquement**, **fortement**, et [**manifeste**](https://en.wikipedia.org/wiki/Manifest_typing). C’est deux langages sont **orientés-objet** , et désigné avec une **semi-interprétation** ou avec un runtime [**Just-in-time compilation**](https://en.wikipedia.org/wiki/Just-in-time_compilation), enfin les deux sont dit **[curly brace langages](https://en.wikipedia.org/wiki/Curly_bracket_programming_language" \o "Curly bracket programming language)**, comme le [C](https://en.wikipedia.org/wiki/C_(programming_language)) et le [C++](https://en.wikipedia.org/wiki/C%2B%2B).

Java et C# partagent un ensemble de fonctionnalités communes :

* Compilation dans un langage intermédiaire indépendant de la machine et exécution dans un environnement dédié (une machine virtuelle)
* Gestion automatique de la mémoire grâce à un ramasse-miettes
* Introspection pour manipuler dynamiquement les objets
* Toutes les classes héritent d'une même classe (Object) et sont allouées sur le tas
* Pas de support de l'héritage multiple mais utilisation d'interfaces
* Tout doit être encapsulé dans une classe : il n'existe pas de fonctions ou constantes globales
* Gestion des erreurs grâce aux exceptions

Java et C# possèdent cependant des différences :

* Java possède des exceptions vérifiées (traitement ou déclaration de la propagation de ces exceptions)
* C# propose le mode unsafe qui permet de manipuler la mémoire (hors du contrôle de l'environnement managé)
* C# propose les propriétés, les indexeurs, les délégués et les événements
* C# propose la surcharge des opérateurs
* C# possède les structures qui sont des types valeurs
* La génération de la documentation est différente : Javadoc génère une documentation au format HTML, C# génère des fichiers XML
* C# propose l'instruction goto

Chaque langage possède des fonctionnalités importantes qui lui sont propres notamment :

* En C# : passage par référence, delegates, événements, indexeurs, objets de type valeur, surcharge des opérateurs, pointeurs (dans des portions de code non managée), préprocesseur, l'instruction goto, ...
* En Java : la portabilité inter plate-forme, les exceptions vérifiées (checked exception), les classes internes anonymes, la documentation automatique du code (Javadoc), ...

Il est intéressant de remarquer que certaines fonctionnalités d'un des langages sont incorporées dans l'autre et vice versa au fur et à mesure de leurs nouvelles versions (exemple : les fonctionnalités boxing/unboxing, enumération, parcours de collections et les generics ajoutés dans Java 5.0, les classes anonymes ajoutées dans C# 2.0 ou les classes anonymes internes ajoutées dans C# 3.0)

**CONCLUSION**

**Le but final du C# :**

Il y a de multiples raisons d’apprendre le C# :

* C’est un langage très utilisé en entreprise avec beaucoup plus de demande de developpeurs par les entreprises que d’offre.
* La raison pour laquelle il est très demandé en entreprise est parce qu’il est versatile :
* On peut faire des sites web rapidement avec C# ASP NET MVC
* On peut faire des applications de bureau avec **WPF**
* On peut faire des applications mobiles multiplateformes (Apple + Android) avec Xamarin.
* Des technologies comme EntityFramework ou LINQ permettent de manipuler facilement des données de base de données avec du C#
* Développer des jeux vidéo avec Unity
* C’est un langage qui a bien pris en compte les problématiques actuelles du développement d’où sa popularité. Développer du code maintenable et évolutif est relativement aisé.
* Du fait de sa popularité, il y a beaucoup de questions répondues sur Stackoverflow et Google, il est donc facile de résoudre des erreurs en C#.

**Les applications codées en C# :**

Quelques exemples d’applications/logiciels codé en C# :

* C’est un langage très utilisé en entreprise avec beaucoup plus de demande de developpeurs par les entreprises que d’offre.
* La raison pour laquelle il est très demandé en entreprise est parce qu’il est versatile :
* On peut faire des sites web rapidement avec C# ASP NET MVC
* On peut faire des applications de bureau avec WPF
* On peut faire des applications mobiles multiplateformes (Apple + Android) avec Xamarin.
* Des technologies comme EntityFramework ou LINQ permettent de manipuler facilement des données de base de données avec du C#
* Développer des jeux vidéo avec Unity

**Bibliographie et Sources :**

[**https://fr.quora.com/Pourquoi-apprendre-le-C-en-2017**](https://fr.quora.com/Pourquoi-apprendre-le-C-en-2017)

[**https://fr.wikipedia.org/wiki/C\_sharp**](https://fr.wikipedia.org/wiki/C_sharp)

[**https://www.jmdoudoux.fr/java/dej/chap-java\_et\_csharp.htm**](https://www.jmdoudoux.fr/java/dej/chap-java_et_csharp.htm)

[**http://wiip.fr/content/differences-entre-csharp-et-java**](http://wiip.fr/content/differences-entre-csharp-et-java)

[**https://docs.microsoft.com/fr-fr/dotnet/csharp/programming-guide/delegates/**](https://docs.microsoft.com/fr-fr/dotnet/csharp/programming-guide/delegates/)

[**https://docs.microsoft.com/fr-fr/dotnet/csharp/programming-guide/statements-expressions-operators/lambda-expressions**](https://docs.microsoft.com/fr-fr/dotnet/csharp/programming-guide/statements-expressions-operators/lambda-expressions)

**https://docs.microsoft.com/fr-fr/dotnet/csharp/programming-guide/indexers/**

**Lexique :**

[**Typé**](https://en.wikipedia.org/wiki/Type_system): Dans les langages de programmation, un système de types est un ensemble de règles qui attribue une propriété appelée type aux différentes constructions d'un programme informatique, telles que des variables, des expressions, des fonctions ou des modules. L'objectif principal d'un système de types est de réduire les risques de bug en définissant par exemple des interfaces entre différentes parties d'un programme, puis en vérifiant que les parties ont été connectées de manière cohérente.

**Statiquement**: Le [typage](https://fr.wikipedia.org/wiki/Type_(informatique)) statique est une technique utilisée pour associer à un [symbole](https://fr.wikipedia.org/wiki/Symbole) dénotant une [variable](https://fr.wikipedia.org/wiki/Variable_(informatique)) le type de la valeur dénotée par la variable. (Un [compilateur](https://fr.wikipedia.org/wiki/Compilateur) de langage à typage statique détecte les erreurs de types avant que le programme ne soit exécuté, on obtient ainsi la sûreté du typage)

**Fortement** : Le langage fortement typé a des règles de typage plus strictes au moment de la compilation, ce qui implique que les erreurs et les exceptions sont plus susceptibles de se produire lors de la compilation. La plupart de ces règles affectent l'affectation des variables, les valeurs renvoyées et l'appel des fonctions.

**Manifest** : En informatique, le typage manifeste est l’identification explicite par le programme du type de chaque variable déclarée. Par exemple: si la variable X va stocker des entiers, son type doit être déclaré en tant qu'entier.

**Langage orienté-objet** : Un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre. Il possède une structure interne et un comportement, et il sait interagir avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ; l'interaction entre les objets via leurs relations permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes.

**Semi-interprétation** : Cela signifie un langage qui est compilé en un code intermédiaire, qui est ensuite interprété. Cela signifie que le langage peut être utilisé pour déployer un programme sur de nombreuses machines différentes sans recompilation. Le seul inconvénient est une réduction des performances par rapport à la compilation directe.

**Just-in-time compilation** : En informatique, la compilation Just-in-time (JIT) est une manière d'exécuter du code informatique qui implique la compilation pendant l'exécution d'un programme - au moment de l'exécution - plutôt qu'avant l'exécution. Le plus souvent, il s’agit de la conversion de code source en code machine, qui est ensuite exécutée directement. Un système implémentant un compilateur JIT analyse généralement en continu le code en cours d’exécution et identifie des parties du code où l’accélération tirée de la compilation ou de la recompilation l'emporterait sur le temps système nécessaire à la compilation de ce code.

[**Curly brace langages**](https://en.wikipedia.org/wiki/Curly_bracket_programming_language): Ce sont le groupement des langages qui utilisent des accolades pour compiler ensemble les blocs. Ceux-ci ont été inspirés par le langage C.

[**Common**](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) **Language Runtime**: C’est le nom choisi par Microsoft pour le composant de machine virtuelle du Framework .NET. Il s'agit de l'implémentation par Microsoft du standard Common Language Infrastructure (CLI) qui définit l'environnement d'exécution des codes de programmes. Le CLR fait tourner une sorte de bytecode nommé Common Intermediate Language (CIL)

**WPF**: C’est la spécification graphique de Microsoft .NET 3.0. Il intègre le langage descriptif XAML qui permet de l'utiliser d'une manière proche d'une page HTML pour les développeurs.