**Différence entre Java & 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), 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).

**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.