**C# : Les bases**

Flux de control / Flux d’exécution : Ordre dans lequel sont exécutées les instructions du programme et dépendant de la réaction de celui-ci à l’entrée qui lui est donnée.

Instructions :

* De déclaration : Déclaration d’une nouvelle variable ou constante. Assignation d’une valeur à celles-ci ;
* D’expression : Expression qui calcule la valeur à stocker dans une variable ;
* De sélection : Permettent de rediriger le flux d’exécution vers de nouvelles parties du programme selon certaines conditions ;
* D’itération : Expression permettant d’exécuter une boucle d’instruction ;
* De saut : Permettent de rediriger le flux d’exécution vers une autre section du programme ;
* De gestion des exceptions : Gère les exceptions qui rend possible leur récupération au cours de l’exécution ;
* De checked et unchecked : Permet de vérifier les opérations autorisées à effectuer un dépassement de capacité ou non lors du stockage d’une valeur dans une variable trop petite ;

Type :

* **Lors de la déclaration de la variable :** Famille de la variable étudiée ;
* **Lors de la déclaration d’une fonction :** Famille des paramètres utilisé dans la fonction et famille de la variable retournée par la fonction.

CLR : Common Langage Runtime.

Class : type de référence. Lors de la création d’un objet, la variable à laquelle l’objet est affecté conserve uniquement une référence à son adresse mémoire. Lorsque l’adresse est affectée à une nouvelle variable, celle-ci fait toujours référence à l’objet d’origine. Les modifications apportées à une des deux variables sont répercutées sur la seconde puisqu’elles font référence au même objet.

Struct : type de valeur. Lors de la création d’un struct, les valeurs réelles de de celui-ci sont assignées à la variable. Si l’on affecte le struct à une nouvelle variable, il est copié. Les deux variables contiennent donc deux copies distinctes de l’objet et peuvent être modifiées indépendamment l’une de l’autre.

Délégué : Type de variable permettant d’instancier une méthode possédant une signature avec les mêmes types de variables de renvoie et arguments d’entrées que la méthode instanciée grâce à elle. Cela crée une sorte de Classe permettant de faire appel au type de fonction décrit précédemment. Cela représente un type d’encapsulation des méthodes. Cela permet également de passer cette méthode en argument d’une autre pour y faire appel simplement. Cette encapsulation fonctionne également pour des méthodes dites anonymes. Les méthodes anonymes ne permettent pas d’utiliser les méthodes de saut d’instructions telles que Goto, Break et Continue, depuis l’extérieur vers l’intérieur de la méthode anonyme.

Caractères d’échappement : Ce sont les caractères inclus dans les chaînes de caractères précédés par un « \ » et qui sont interprétés par le compilateur puis convertis en déplacements ou caractères d’organisation de texte comme pour les tabulations ou les retours chariots à la ligne. Certains de ces caractères sont ajoutés par le compilateur lui-même lors de l’affichage des chaînes sur la console et peuvent être visibles sur la fenêtre d’un Espion.

StringBuilder : C’est une classe de variable permettant de rendre les chaînes de caractères mutables. Grâce à celle-ci, il est possible de modifier directement les caractères dans d’une chaîne sans en créer de nouvelle. Il est possible d’y ajouter de nouvelles chaînes de caractères grâce à une mémoire tampon lui permettant d’adapter sa taille mémoire en fonction de la longueur de la chaîne. Elle ne peut pas être hérité. Il n’est pas toujours possible de remplacer automatiquement une String par un StringBuilder, c’est pourquoi sont utilisation est intéressante si elle permet une amélioration significative des performances du système. Les performances dépendent de la taille de la chaîne, du système utilisé, de la quantité de mémoire allouée et du type d’opérations effectuées. La longueur d’un StringBuilder peut augmenter jusqu’à atteindre la StringBuilder.Capacity. Si cette valeur est dépassée alors la quantité de mémoire est doublée jusqu’à atteindre la StringBuilder.MaxCapacity. Un dépassement de cette valeur entraine la levée d’une exception.

Requêtes LINQ : Les requêtes sur les bases de données peuvent se faire sous deux formes : la forme Query et la forme Method. Ce type d’instructions permet d’accéder au contenue de la base de données ciblée suivant certaines conditions qui peuvent être implémentées dans le script du code.

Expressions Lambda : Ce sont des fonctions anonymes. Ce sont des expressions comprenant des opérateurs « => » qui utilisent une variable d’entrée placée à la gauche de ce symbole et l’expression ou le bloc d’expressions, qui peut aussi être un appel de fonction (sauf pour les arborescences d’expressions en dehors du Framework .NET), à droite. Cette variable implémente une interface IEnumerable<T>. Grâce à elles, il est possible d’écrire des fonctions locales qui peuvent être placées en argument d’une méthode et dont la valeur de retour sera prise en compte par la fonction appelante. On s’en sert beaucoup pour effectuer des requêtes LINQ. Ce type d’expression n’est pas autoriser du coté gauche de l’opérateur « is » ou « as ».