

Beuscart Benjamin

Lupa Valentin

Malbranque Louis

Verpoort Alexia

***Découverte***

***Du***

***C#***



***I/ Introduction au C# :***

Le C# est un langage créé en 2002 par Microsoft.

Il utilise le Frameworks.net et utilise donc la combinaison entre les frameworks et les spécificités du langage pour créer une application. Le frameworks.net étant un composant microsoft intégral prenant en charge la création et l’exécution d’applications et de services web. Il est basé sur le langage CLI (Common Langage Structure) et ne dépend donc du langage de programmation. Tous les langages compatibles ont donc accès aux bibliothèques installables dans l’environnement d’exécution. Le but étant de faciliter le développement des applications.

Bien qu’il existe des similitudes avec d’autres langages comme le C et le C++, le C# possède des avantages. En comparaison avec le C, il permet une meilleure comptabilité car l’IDE transforme le code en langage CIL ou MSIL plutôt qu’en binaire. Il est également plus simple que le C++.

Le C# est un langage programmé objet mais possède davantage de features que le langage Java par exemple. C’est également un langage orienté composant. C’est-à-dire une approche plus modulaire de l’architecture d’un projet pour obtenir une meilleure lisibilité et une meilleure maintenance. (Plus d’explication ???)

Le C# est également un langage intégrant des fonctionnalités permettant d’obtenir des applications fiables. Il intègre en effet le **type-safe** permettant de vérifier qu’il n’y a pas d’erreurs propres aux variables (mauvaise indexation de tableaux, mauvaise opération sur un type…). Il intègre également la **Gestion des exceptions** pour permettre une récupération efficace des erreurs générées et enfin le **garbage Collection** qui permet de récupérer la mémoire occupée inutilement.

Compilation C# : empaquement dans assemblys qui implémentent des bibliothèques (.dll) ou des applications (.exe)

***II/ Les bases du C# :***

1. ***Types en C#***

La déclaration d’une variable s’effectue en déclarant le type de la variable directement (float, int…) ou en utilisant var qui permet au compilateur de déduire. Une variable déclarée ne peut pas être assigné à une variable d’un autre type sans utiliser le type-casting qui est une conversion explicite d’un type. Certains cas entrainent cependant une conversion implicite s’il n’y a pas de perte de donnée et cela est effectué par le compilateur.

*Exemple de cast :* int a=10 ;

double b ;

b=(double)a ;

Le C# intègre le principe de **types unifiés**, c’est-à-dire que tous les types héritent du type object et donc il y a un partage d’opérations communes entre les types.

En C# on trouve deux grands types différents, les types valeurs et les types références.

1. ***Le type Valeur***

Une variable de type valeur contient directement la donnée. Une opération sur une variable n’influence pas une autre variable de même type. Le type valeur regroupe les types simples, les struct et les enum. C’est un type scellé, il ne prend donc pas en charge l’héritage, c’est-à-dire qu’aucune classe ne peut hériter d’un struct par exemple.

* Les structures

Les structures peuvent contenir des données et des fonctions membres. Il s’agit d’un conteneur de variables liées.

struct cercle

{

public int rayon, centrex, centrey;

public cercle(int rayon, int centrex, int centrey)

{

this.rayon = rayon;

this.centrex = centrex;

this.centrey = centrey;

}

}

Sur cet exemple nous définissons une structure de type cercle qui contient 3 variables : son rayon et les coordonnées selon x et y de son centre.

* Les énumérations

Une énumération sont des ensembles de constantes. Cela permet de définir un type avec des valeurs discrètes. Par exemple, pour une liste de 3 boissons, on peut créer une énumération de ces trois boissons, puisqu’il n’y en aura pas une infinité plutôt que de créer une classe Boisson et d’en instancier 3 différentes.

enum Vetement

{

Short,

Jupe,

Jean,

Pull

}

D’après ce qui a été dit plus haut, le type Valeur contient directement la donnée, voyons ce que cela entraine sur notre exemple de structure.

cercle cercle1 = new cercle(2, 1, 1);

cercle cercle2 = cercle1;

cercle1.rayon = 1;

Console.WriteLine("cercle1 : Mon rayon est de " + cercle1.rayon);

Console.WriteLine("cercle2 : Mon rayon est de " + cercle2.rayon);

Console.ReadKey();

On observe bien que la modification du rayon du cercle1 n’a pas modifié le rayon du cercle 2.

1. ***Le type Référence***

Une variable de type référence contient la référence à leur donnée qui est appelé objet. Une opération sur un variable peut influencer une autre variable de même type. Les types références sont des classes, interfaces, délégué ou tableaux. Ce type est initialisé à null et nécessite la création grâce à l’objet new.

* Les classes

Les classes sont des structures de données améliorées contenant à la fois les données et les méthodes liées.

public class cercle

{

public int rayon, centrex, centrey;

public cercle(int rayon, int centrex, int centrey)

{

this.rayon = rayon;

this.centrex = centrex;

this.centrey = centrey;

}

}

* Les interfaces

 Ressemble à une classe de type abstraite. Elle ne peut être instanciée directement, c’est-à-dire qu’on ne peut créer un objet directement du type de l’interface. Les interfaces ne contiennent pas d’implémentation des méthodes.

Une classe peut implémenter plusieurs interfaces.

interface GérerFigure

{

void Dessiner();

}

Cette interface pourrait être implémantée par la classe cercle créée plus haut.

* Les tableaux

Comme en C, les tableaux sont indexés sur zéro, avec n éléments allant de 0 à n-1. Les tableaux sont de tous types y compris du type tableau. Les valeurs par défaut des éléments de tableau sont définies sur zéro et les éléments de référence sont définis sur Null. Un tableau peut être unidimensionnel, multidimensionnel comme en C ou encore en escalier. La particularité de l’escalier est que les éléments du tableau principal n’ont pas les mêmes dimensions entre eux :

cercle[][] a = new cercle[3][];

a[0] = new cercle[3];

Tableau en escalier

a[1] = new cercle[5];

a[2] = new cercle[1];

* Les délégués

Les délégués sont des variables qui pointent vers des méthodes. Ils sont déclarés avec **delegate** et permettent la mise en place du principe d’ouvert/fermé, autrement dit, ils permettent de passer des méthodes comme argument lors des appels. Ils sont en quelque sorte un moyen de modifier le comportement d’une application sans modifier son code source et donc de réorganiser le code.

Leur effet est lié à celui de la catégorie de classes **abstract**.

Les délégués peuvent être déclarés dans la partie **namespace** ou alors au sein même d’une classe.

Le mot clé **delegate** permet également d’instancier un délégué avec une méthode quelconque qui correspond à test2 dans l’exemple suivant.

EXEMPLE PRATIQUE :

namespace UsingDelegate

{

//Creation d’un délégué qui prend en paramètre un string et renvoie un string

delegate string ActionOnString(string text);

class Program

{

static string affichageClassique(string text)

{

return text;

}

static void Main(string[] args)

{

//Instancie un délégué pointant vers affichageClassique

ActionOnString test1 = affichageClassique;

//appel du délégué test1

//affichageClassique

string result1 = test1("Première phrase de test.");

Console.WriteLine(result1 + "\n");

//Instancie un délégué vers une méthode quelconque

//All in uppercase

ActionOnString test2 = delegate (string text)

{

string tmp = text.ToUpper();

return tmp;

};

string result2 = test2("Deuxième phrase de test.");

Console.WriteLine(result2 + "\n");

//Instancie un délégué avec une expression lambda

//qui concatène deux fois le même string

ActionOnString test3 = s => s + s;

string result3 = test3("Troisième phrase de test.");

Console.WriteLine(result3 + "\n");

Console.ReadKey();

}

}

Dans l’exemple précédent on a inséré l’utilisation des expressions Lambda (explication expression lambda)

Il existe également des délégués dits génériques qui sont les délégués Action et Func.

Action permet de pointer vers une méthode qui ne renvoie rien et prend en argument un tableau de 16 types maximum.

Func pointe au contraire vers une méthode qui retourne quelque chose.

D’après ce qui a été dit plus haut, le type Référence contient la référence de la donnée et non pas directement la donnée, voyons ce que cela entraine sur notre exemple de classe.

cercle cercle1 = new cercle(2, 1, 1);

cercle cercle2 = cercle1;

cercle1.rayon = 1;

Console.WriteLine("cercle1 : Mon rayon est de " + cercle1.rayon);

Console.WriteLine("cercle2 : Mon rayon est de " + cercle2.rayon);

Console.ReadKey();

La modification d’un champ de cercle1 a également modifié celui de cercle2 car leur référence est identique.

L’utilisateur peut créer ses propres types à l’aide des constructions struct, classe, interface, enum.

1. ***Utilisation de la programmation objet***

La manière de gérer la programmation objet est semblable aux langages tels que le C++ ou le java, la majeure différence avec ce dernier étant par exemple la syntaxe.

Pour commencer, lors de la création d’une classe, l’héritage par rapport à une classe mère se déroule de la sorte :

public class Salade : Aliment

Ici notre classe Salade dérive de la classe aliment, elle détient donc les mêmes caractéristiques que la classe Aliment. Supposons que la classe aliment soit défini de la manière suivante :



La classe Salade possèdera donc également les variables nom, prix… et aura accès au constructeur et à la méthode descriptionAliment.

Pour utiliser le constructeur ou les méthodes, on passe par l’utilisation du mot-clé **base**. Ce mot clé permet notamment d’overrider les méthodes.



Ici Salade a un constructeur qui reprend les précédents éléments du constructeur de Aliment en prenant pour argument « Salade » pour le nom, 1 pour le prix…

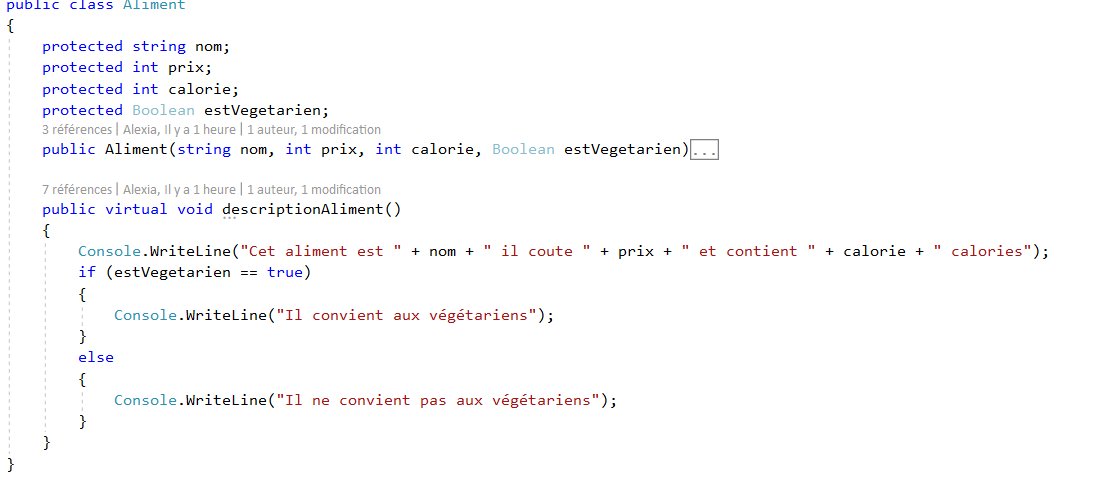
On peut ainsi créer plusieurs classes qui pourront potentiellement avoir chacune leurs caractéristiques mais qui posséderont chacune les caractéristiques de la classe aliment.

Il est également possible de créer une interface qui sera implémentée par les classe de la même manière que l’héritage. Les classes qui l’implémenteront devront alors implémenter les fonctions déclarées.



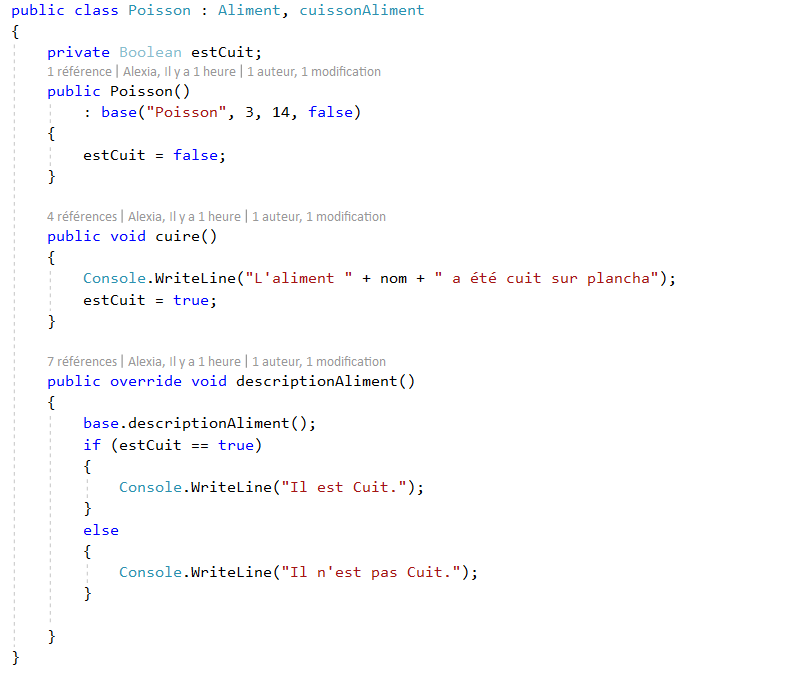
Cette interface pourrait être implémentée par un Aliment qui peut être cuit, ce n’est donc pas nécessaire que Salade implémente cette interface.

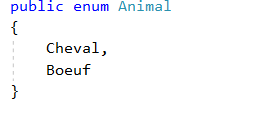
Voyons l’exemple complet qui peut être créé à l’aide la classe Aliment :





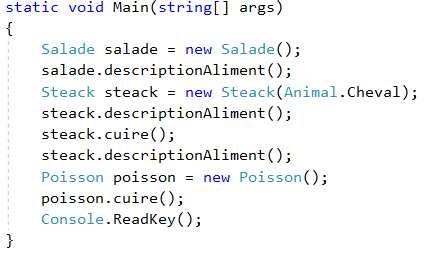
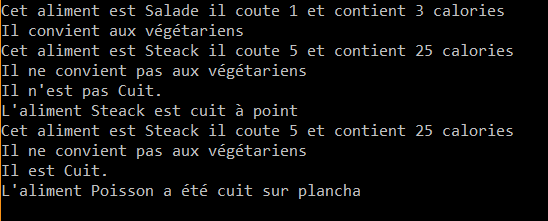








L’exemple suivant nous permet d’obtenir le résultat console suivant :



***EXPLICATIONS SUP ???***

***III/ Concepts Clés du C# :***

*Evenements :*

Permet à un objet qui est appelé publieur d’avertir à d’autres classes, les abonnés lorsqu’un événement important se produit.

Le choix du déclenchement est effectué par le publieur, c’est-à-dire que le publieur choisit quelle action génère un événement et les abonnés choisissent que faire à la réception de l’événement.

Un évènement nécessite un abonné pour pourvoir être déclenché.

Ce sont les event handler qui permettent d’avertir lorsque l’évènement a été déclenché en l’attachant à un évènement.

EXEMPLE PRATIQUE

*Attributs :*

A la base une entité peut être qualifée de private, public, protected ou internal selon son degré d’accessibilité en dehors et dans la classe.

La majeure différence en C# est de pouvoir créer des attributs qui permettent eux aussi de qualifier l’entité, les attributs permettent de donner des informations complémentaires sur la classe ou les méthodes.

Pour créer un attribut, il faut créer une classe héritant de la classe Attribute.

Une fois la classe créée, il suffit de placer l’attribut entre crochet juste avant la déclaration de classe par exemple ou encore de méthode ou de variable. Il est possible de cantonner l’utilisation de l’attribut uniquement aux classes par exemple.

Certains attributs provoquent des messages qui s’écrivent directement dans le compilateur c’est le cas des attributs de type Obsolete qui sont préexistants dans le frameworks.NET. D’autres à l’image de certains attributs tels que les descriptions de classe sont utiles en utilisant la réflexion.

La réflexion permet en fait d’obtenir des informations sur les types

QUELQUES MOTS SUR LA REFLEXION

Le programme Attributs illustre une utilisation de la description des attributs.

<https://docs.microsoft.com/fr-fr/dotnet/csharp/language-reference/language-specification/introduction>

https://openclassrooms.com/fr/courses/1526901-apprenez-a-developper-en-c/2866796-tableaux-listes-et-enumerations