Environnement de développement .Net

Mehdi@Louizi.com

Plan du cours

- Rappels (POO)
- Le Framework .NET
- Les langages de développement
- Applications Windows .NET
- Méthodologie de développement
- XML – XSL et XML avec .NET
- ASP .NET
- SQL SERVER
- Web services
- ADO .NET
- C #

Rappels

Le trio <entité, attribut, valeur>

- Il est possible dans tous les langages informatiques de stocker et de manipuler des objets en mémoire, comme autant d'ensembles de couples attribut/valeur.
- Exemple: <voiture, couleur, rouge>, <voiture, marque, peugeot>, <passant, taille, grande>, <passant, âge, 50>…
- Ce sont les entités et non les attributs qui sont mis en valeur: « objet est intéressant »

Le trio <entité, attribut, valeur>

- La voiture a une taille comme le passant.
- L'arbre a une couleur comme la voiture…
- Le monde des attributs est beaucoup moins diversifié que le monde des objets.
- Les objets se partagent plusieurs attributs.
- Il faut une structure qui puissent les regrouper: c'est la classe et les sous classes.

Stockage des objets en mémoire

- Un objet occupe un espace mémoire dans l'ordinateur lors de sa création.
- Les attributs ont leurs propres types: « types primitifs »
- les types primitifs: int, float, char…
- Les objets seront structurellement décrits par un premier ensemble d'attributs de type primitif.
- Le calcul de l'espace mémoire à occuper est facilement déterminé.

Objet dans le monde

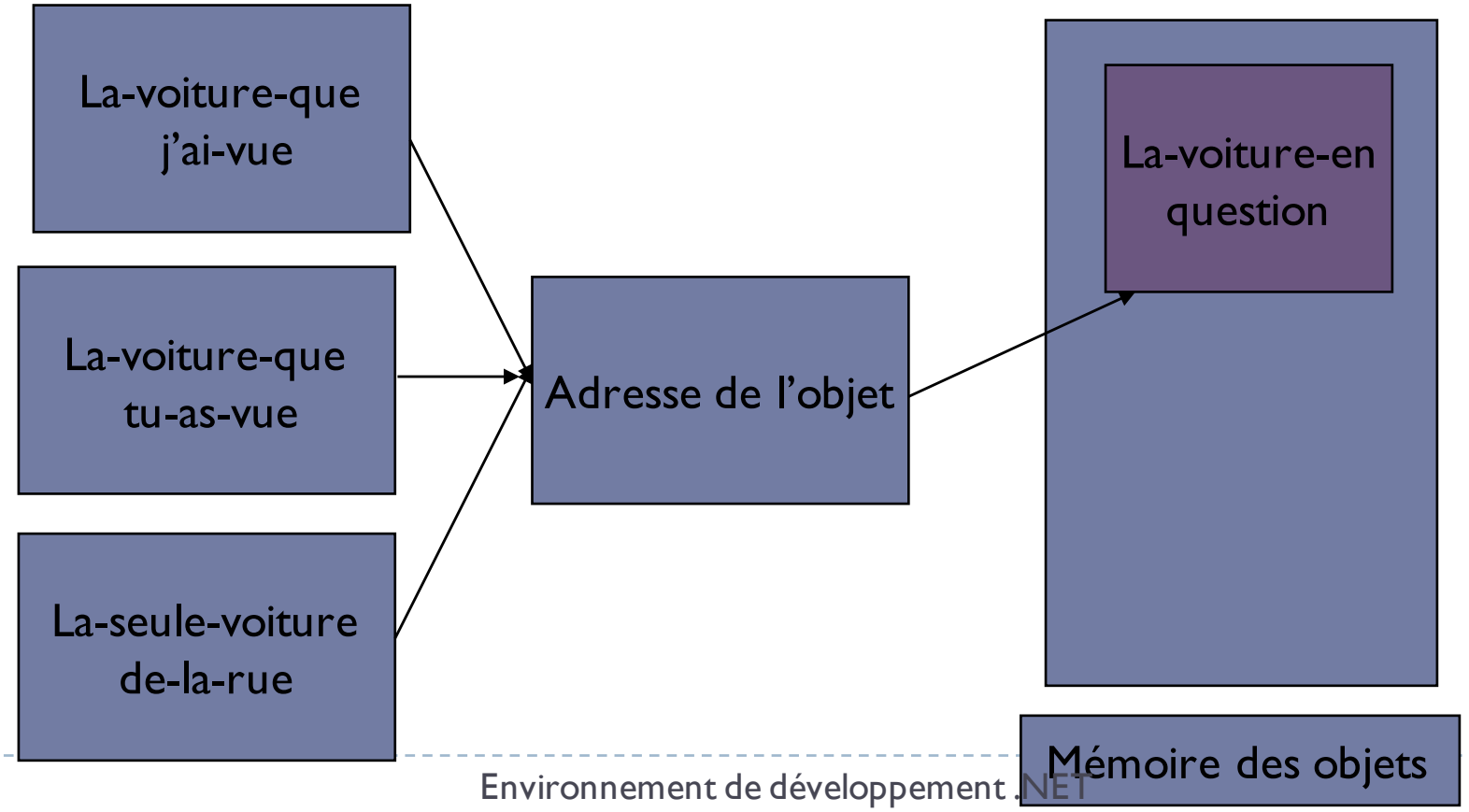
- Point caractérisé par ses coordonnés: x, y et z.
- Un atome caractérisé par son nombre atomique
- La santé d'un pays est caractérisée par son PIB/habitant.

Le référent d'un objet

- Le nom de l'objet est son seul et unique identifiant.
- C'est son adresse physique.
- Un objet existe en mémoire tant qu'il est possible de le référer. Sinon inaccessible.
- Peut on avoir plusieurs référents pour un même objet?

Adressage indirect

- Le même objet peut être désigné par plusieurs noms.



Relation entre objets

- Entre eux les objets peuvent entrer dans une relation de type composition,
- Certains se trouvent contenus dans d'autres et ne sont accessibles qu'à partir de ces autres
- Leur existence dépend entièrement de celle des objets qui les contiennent.
- Exemple: objet voiture et objet moteur

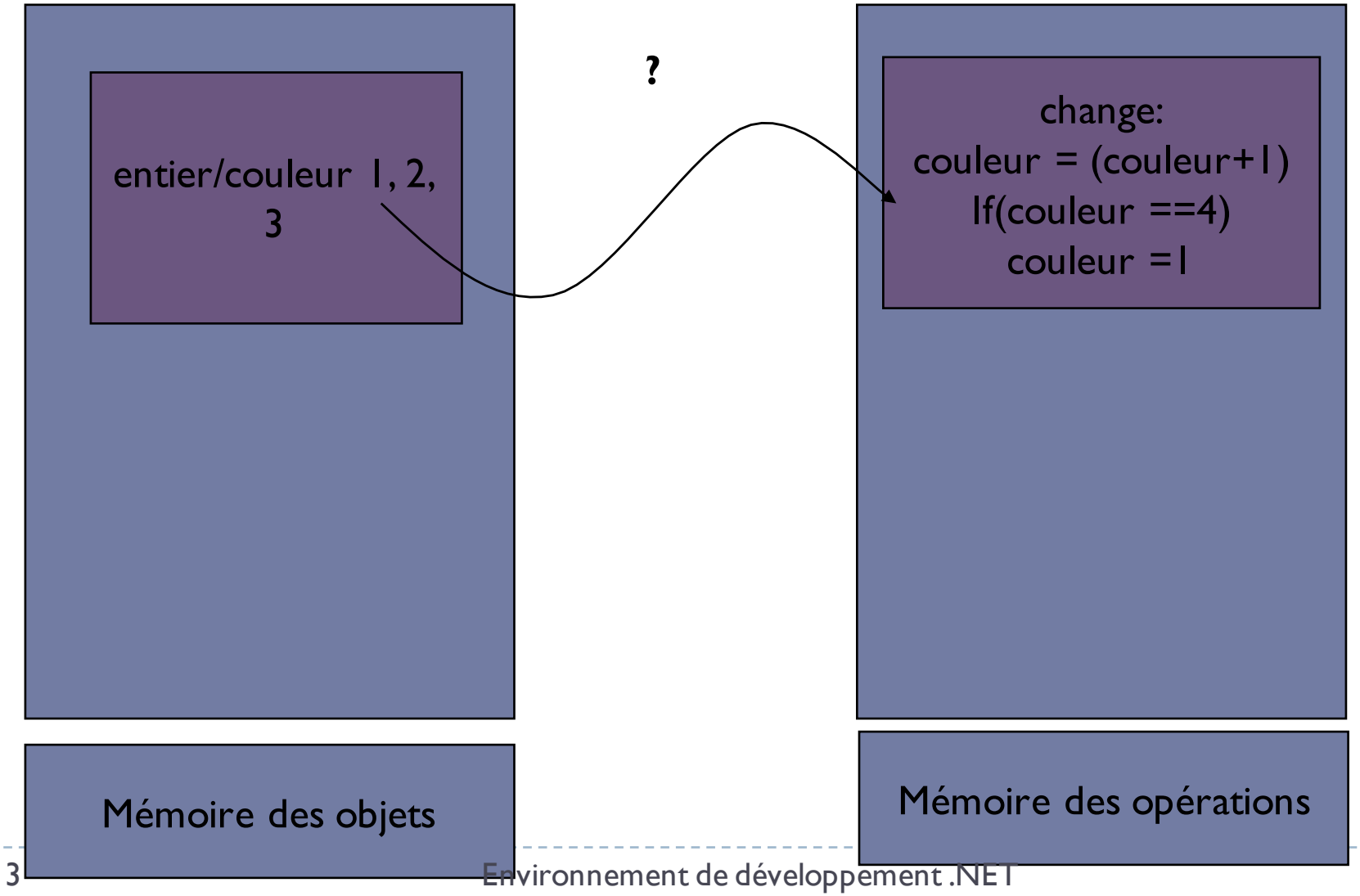
Changement d'états

- Le cycle de vie d'un objet, lors de l'exécution d'un programme orienté objet, se limite à une succession de changements d'états, jusqu'à sa disparition de la mémoire centrale.
- Le changement d'état est le changement de la valeur d'un attribut.

Qui est responsable du changement d'état

- Exemple: le feu de signalisation: celui-que-vous-avez-vu-dans –la-rue
- On suppose que la couleur est représentée par un entier: 1, 2, 3.
- L'opération de changement de couleur: change

L'opération « change »



Comment relier les opérations et les attributs?

- Comment les opérations de change savent elles qu'elle se porte sur le feu de signalisation?
- Bienvenue dans le monde de l'orienté objet !

Rappel de la notion de classe

- L'opération change: *méthode* doit être attachée qu'à des objets de type *feu de signalisation*
- Il faut donc unir l'objet et la méthode par le lien de la **classe**
- Les attributs de la classe sont des variables dont la portée d'action se limite à la seule classe.

Exemple

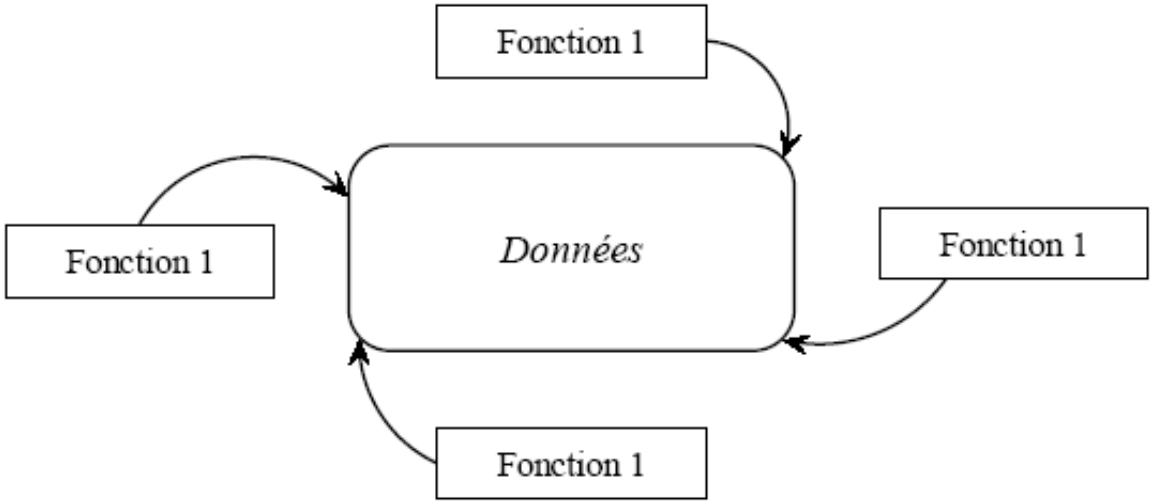
```
classe Feu-de-signalisation {
    int couleur;
    change() {
            couleur = couleur +1;
            if(couleur ==4)couleur = 1;
    }
}
```

Sur quel objet précis s'exécute la méthode

- On lie la méthode f(x) à l'objet « a », sur lequel elle doit s'appliquer, au moyen d'une instruction: **a.f(x)**.
- Appeler la méthode pour l'objet en question:
- **Feu-de-signalisation-en-question.change()**

Rappel POO

- Programmation Procédurale : La programmation procédurale (C, Pascal,…) est constituée d'une suite d'instructions (fonctions, procédures) qui agissent sur des données pour produire un effet quelconque.

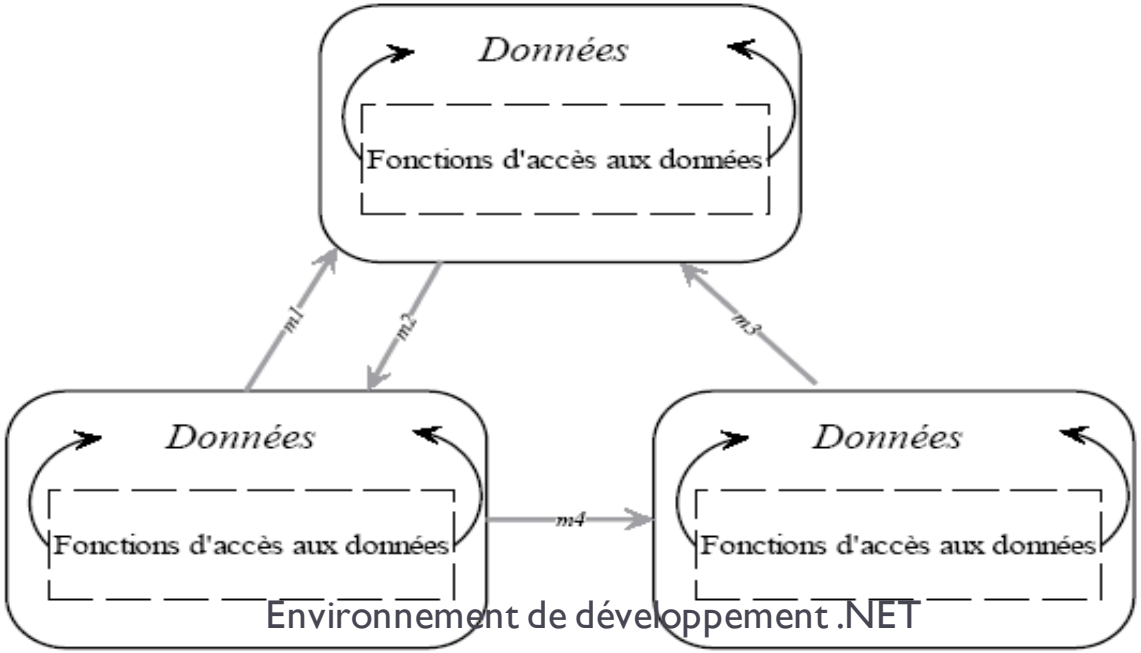


Il y a une **dissociation** entre les données et les fonctions : des difficultés lorsque l'on désire changer les structures de données.

Rappel POO

- L'analyse du cycle de vie des logiciels a démontré que les éléments les plus stables étaient les structures de données.

- De là découle l'idée de centrer la structuration des programmes autour des données en association avec leurs traitements spécifiques (Principe de **l'encapsulation**).



Objectifs de la POO (1)

L'objectif de la POO est donc d'accroître la productivité des programmeurs et d'améliorer la fiabilité des logiciels.

- Conformité fonctionnelle :Capacités d'un logiciel à exécuter les fonctionnalités définies dans les spécifications.
- Robustesse : Capacités d'un logiciel à fonctionner dans des conditions anormales d'utilisation.
- Extensibilité : Facilité avec laquelle un logiciel est capable de s'adapter à un changement dans les spécifications.

Objectifs de la POO (2)

- Réutilisabilité : Capacité du logiciel d'être réutilisé totalement ou partiellement dans de nouvelles applications.
- Portabilité : Facilité avec laquelle un logiciel peut être transporté sur différents systèmes
- Vérifiabilité : Faculté d'accepter des procédures de validation et des jeux de tests
- Intégrité : Capacité d'un logiciel à protéger ses propres composants contre des processus n'ayant pas le droit d'y accéder ou de les modifier

Objectifs de la POO (3)

- Convivialité : Facilité d'apprentissage et d'utilisation d'un logiciel
- Fiabilité : Sécurité contre les maladresses du développement

Fondements de la POO

- **Encapsulation :** Les concepts de Classe et d'Objet sont conséquents au principe de l'encapsulation de de données

- **Héritage :** définir de nouveaux objets à partir d'objets existants

- **Polymorphisme :** la possibilité aux objets d'avoir plusieurs formes de structure et de comportement

Taxonomie

- **Classe :** décrit des "choses" ayant les mêmes propriétés. *Une classe est un prototype qui définit des attributs et des méthodes communes à tous les objets d'une certaine nature classification*
- **Objet :** une instance de la classe. *Types de données complexes avec leurs opérations associées. Un objet est une collection d 'attributs munis de méthodes*

- **Attribut :** *donnée définie par un nom unique pour une classe et par une valeur pour chaque instance (membre, champ)*
- **Méthodes :** actions ou procédures qui s'appliquent à un objet

Taxonomie : Objet (1)

- Un *objet* est une entité cohérente rassemblant des données et les fonctionnalités qui leur sont associées
  - Il possède des attributs (données) et un modèle comportemental composé par des méthodes (traitements)
  - Types de données complexes avec leurs opérations associées. Un objet est une collection d 'attributs munis de méthodes
- Tout objet est une instance d'une classe

Taxonomie : Objet (2)

- Un objet est donc composé de:
  - comportement visible;
  - état interne caché;
  - identité.
- On conçoit un objet comme une représentation abstraite d'un composant du système vu par l'utilisateur.
- Un objet possède également une durée de vie : la durée de temps entre sa création et sa destruction.

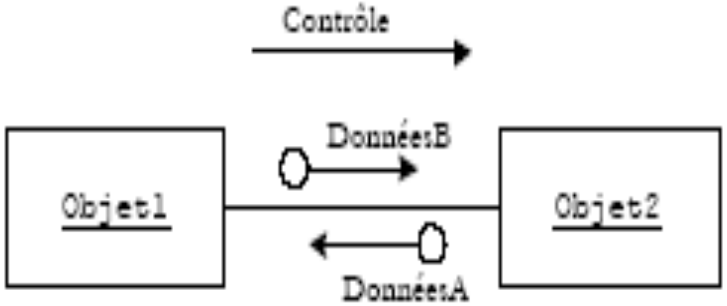
Taxonomie : Objet (3)

Objet = État + comportement + identité

- L'état d'un objet est l'ensemble des valeurs des attributs à un moment donné durant la vie de l'objet
- L'identité d'un objet permet de faire la distinction entre les objets de même type
- Le comportement d'un objet regroupe les capacités d'un objet
- Chaque composant du comportement d'un objet est appelé une opération

Interaction entre Objets (1)

- Le message est l'unité de base des interactions entre objets
- Les messages sont essentiels pour l'établissement d'une communication entre deux objets
- La réalisation concrète des messages peut prendre différentes formes :
  - un appel de fonction
  - une interruption
  - un événement



Interaction entre Objets (2)

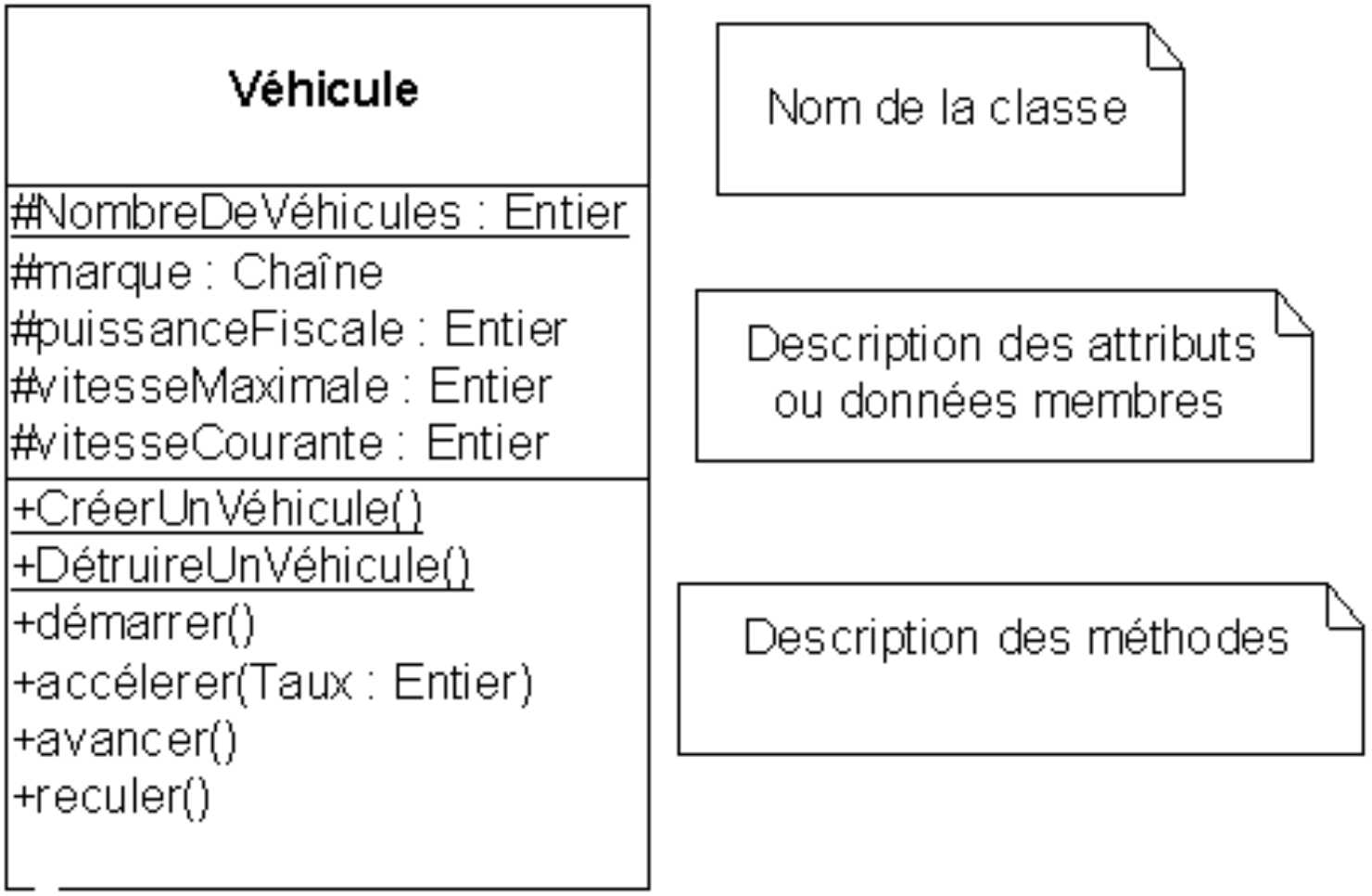
Quelques messages importants à un objet:

- message constructeur
  - Création d'un objet
- message destructeur
  - Destruction d'un objet
- message sélecteur
  - Obtenir des informations d'un objet
- message modificateur
  - Modifie l'état (valeurs des attributs) d'un objet
- message itérateur
  - Obtenir un objet parmi un ensemble d'objet

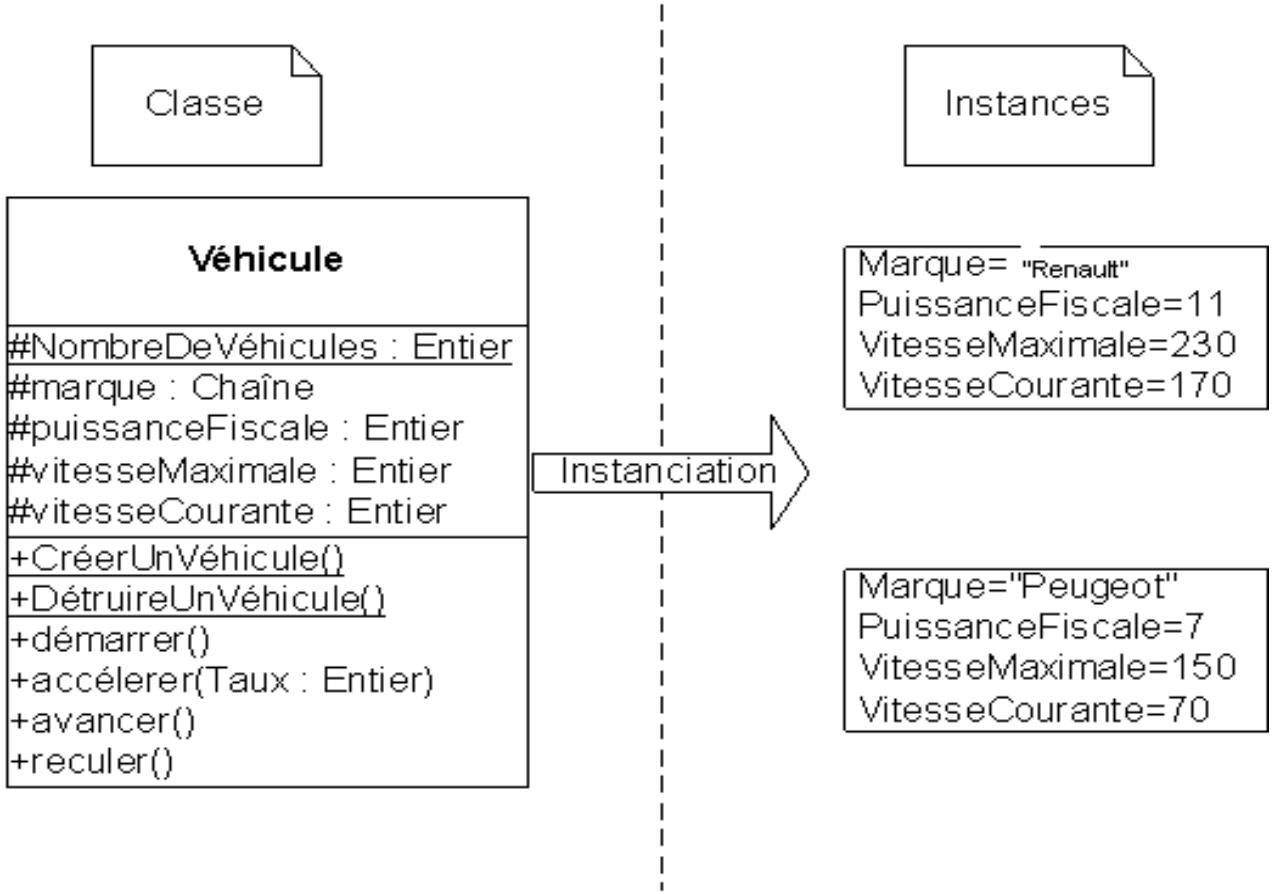
Taxonomie : Classe (1)

- Une classe décrit donc les caractéristiques générales d'un ensemble d'objets
  - Donc, chaque objet appartient à une classe.
- La création d'un objet à partir de sa classe est appelée une instanciation
  - On dit un objet est une *instance* d'une classe
- Une classe décrit la structure interne d'un objet : les données qu'il regroupe, les actions qu'il est capable d'assurer sur ses données. Un objet est un *état* de sa classe où les données prennent une valeur

Taxonomie : Classe (2)



Taxonomie : Classe (3)



Taxonomie : Classe (4)

Programmation Procédurale	VARIABLE	TYPE
Programmation Orientée-Objet	OBJET	CLASSE

Encapsulation (1)

- Par rapport à une approche « classique » que l'on peut observer dans la programmation procédurale, l'approche objet se caractérise par le regroupement dans une même classe de la description des attributs et des opérations (méthodes). Ce regroupement s'appelle l'*encapsulation*. On masque l'information au monde extérieur
- **Encapsulation :** améliore l'autonomie et l'indépendance de chaque classe et augmente le potentiel de réutilisation de chacune d'elles
- La visibilité des attributs et des opérations d'une classe vis à vis d'autres porte le nom d'*interface*.

Encapsulation (2)

- Une amélioration significative de la fiabilité des logiciels est la possibilité d'interdire ou de permettre l'accès de certains membres d'une classe aux autres éléments du programme. Ceci est une des propriétés de l'encapsulation des données.

L'extérieur ne connaît ni les procédés ni les détails.

- Des *mots-clés particuliers* permettront de déclarer les membres qui sont accessibles au monde extérieur.

Encapsulation (3)

- *Normalement,* la valeur des attributs d'un objet n'est pas accessible directement par un autre objet
  - On dit que les attributs sont masqués à l'intérieur d'un objet

- L'interaction entre les objets s'opère en activant différentes opérations déclarées dans l'interface

- Les opérations de l'interface d'une classe sont accessibles à d'autres classes

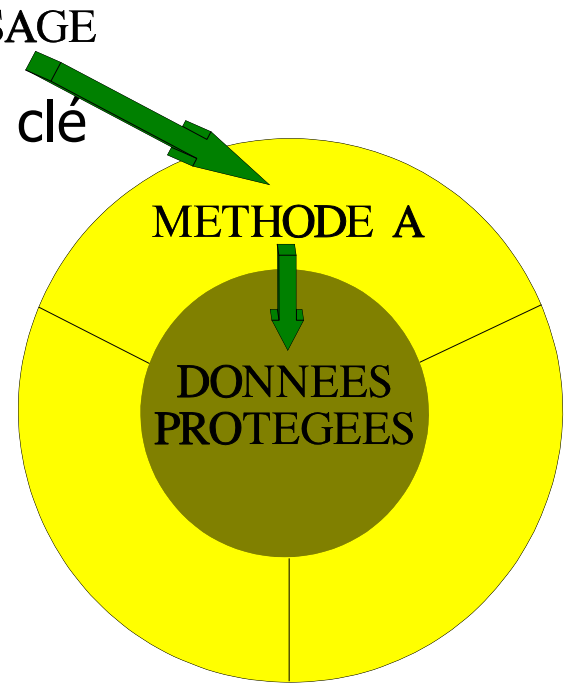
Encapsulation (4)

Exemple : Encapsulation des données

Méthode : Calcul de la clé d'un numéro de sécurité sociale

Objet : Individu (Assuré Social par exemple)

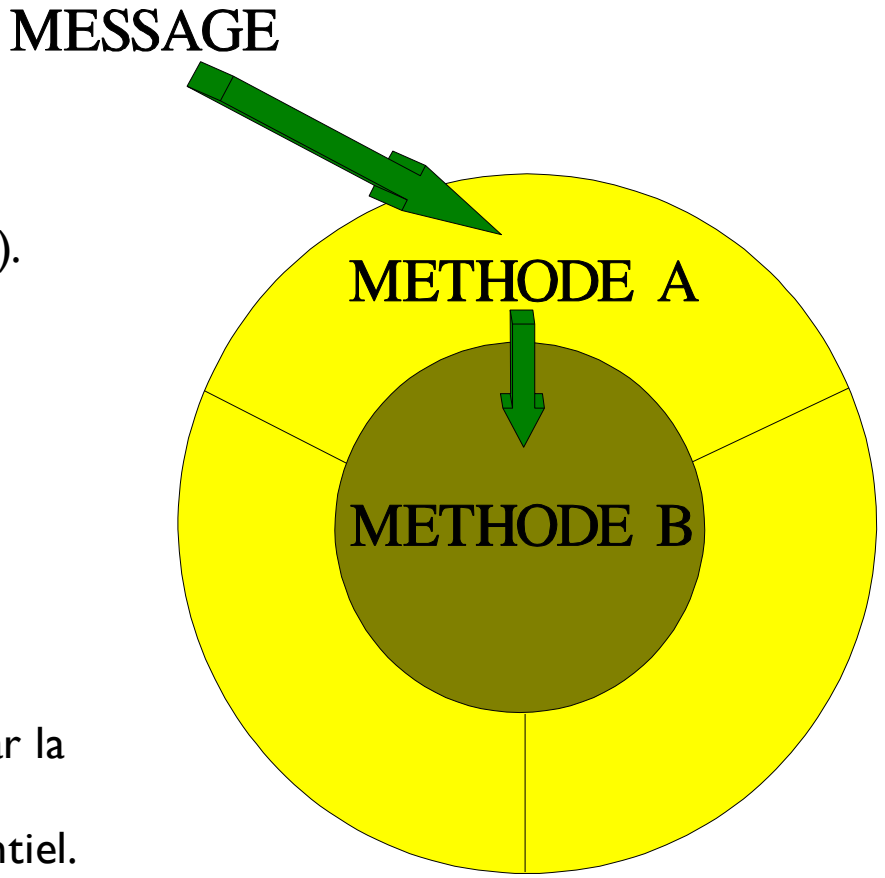
Donnée protégée : Valeur du modulo pour le calcul de clé



Encapsulation (4)

Cas	Positions	Signification	Valeurs possibles
Tous	1	sexe : 1 pour les hommes, 2 pour les femmes, 3 pour les personnes étrangères de sexe masculin en cours d'immatriculation en France [réf. nécessaire] [A], 4 pour les personnes étrangères de sexe féminin en cours d'immatriculation en France [A]	1 ou 2 ou 3 ou 4 ou 7 ou 8
	2 et 3	deux derniers chiffres de l'année de naissance (ce qui donne l'année à un siècle près)	de 00 à 99
	4 et 5	mois de naissance	de 01 (janvier) à 12 (décembre)
A	6 et 7	département de naissance métropolitain (2A ou 2B pour la Corse)[C]	de **01** à **95 ou 2A ou 2B**
	8, 9 et 10	numéro d'ordre de la commune de naissance dans le département[C,D]	de 001 à 989, ou 990 [2]
B	6, 7 et 8	département de naissance en outre-mer[C]	de **97**0 à **98**9
	9 et 10	numéro d'ordre de la commune de naissance dans le département[C,D]	de 01 à 89, ou 90 [2]
C	6 et 7	naissance hors de France[C]	**99**
	8, 9 et 10	identifiant du pays de naissance[C]	de 001 à 989, ou 990[B]
Tous	11, 12 et 13	numéro d'ordre de l'acte de naissance dans le mois et la commune (ou le pays)[D]	de 001 à 999
	14 et 15	clé de contrôle = complément à 97 du nombre formé par les 13 premiers chiffres du NIR modulo 97[E] (complément au NIR pour la Sécurité sociale)	de 01 à 97

Encapsulation (5)



Exemple : Encapsulation des méthodes

Message = Débiter(somme, compte, code_confidentiel).

Objet = compte bancaire.

Méthode A (Visible depuis l'interface) = Débiter_carte_de_crédit.

Méthode B (non visible par l'utilisateur mais appelée par la méthode A) = Algorithme de validation du code confidentiel.

Des méthodes encapsulées peuvent faire appel à des données qui sont elles-mêmes protégées

Taxonomie : Méthodes et attributs (1)

- Attributs d'instance : Chaque objet a ses propres données. on parle alors d'attribut *d'instance*. L'opération *d'instanciation* qui permet de créer un objet à partir d'une classe consiste précisément à fournir des valeurs particulières pour chacun des attributs d'instance.

- Attributs de classe : Chaque classe peut posséder ses propres attributs. Un attribut partagé par l'ensemble des objets

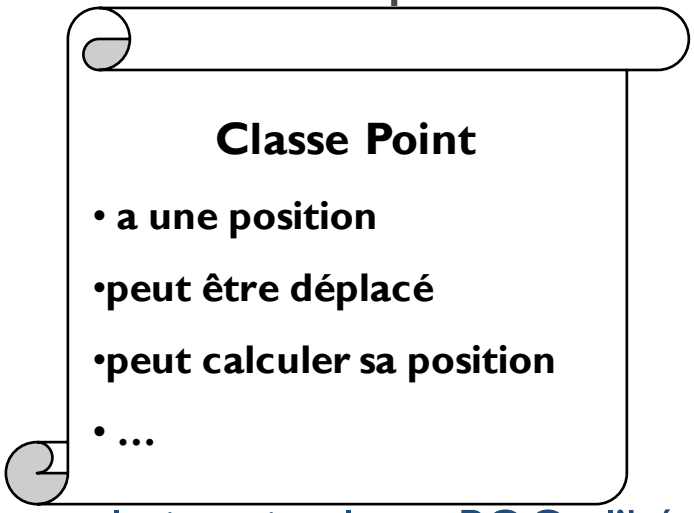
Taxonomie : Méthodes et attributs (2)

- Méthodes d'instance : C'est une méthode qui peut s'appliquer individuellement à chaque objet. En outre, cette méthode va clairement utiliser les attributs d'instance de l'objet auquel elle va s'appliquer

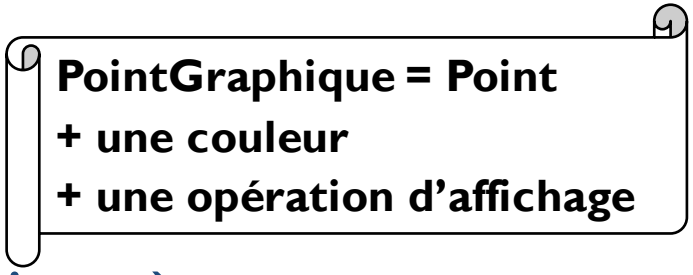
- Méthodes de classe : Chaque classe peut posséder ses propres méthodes.

Héritage (1)

- Extension d'une classe
- Intérêts
  - Une application a besoin de services dont une partie seulement est proposée par une classe déjà définie (on ne possède pas nécessairement le code source de cette classe)
  - Permet de ne pas réécrire tout le code



Une application a besoin de manipuler des points (comme le permet la classe Point), mais en plus de dessiner des points à l'écran



- Une solution simple en POO : l'héritage **(inheritance)**
  - Définir une nouvelle classe à partir de la classe déjà existante

Héritage (2)

Côté Technique

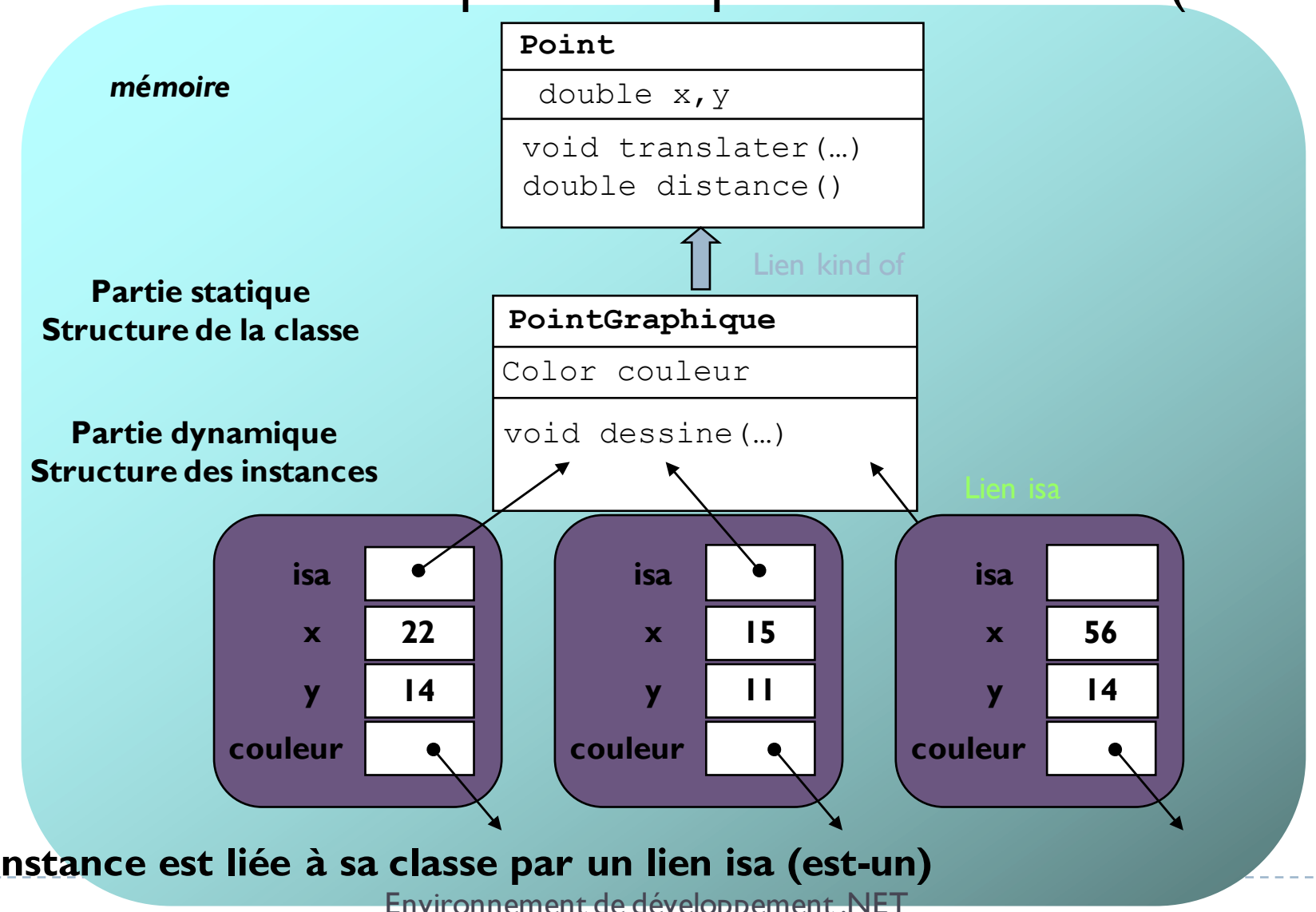
- Technique de programmation permettant de définir de nouvelles classes à partir de classes de base.
- Les nouvelles classes héritent le contexte des classes et rajoutent de nouvelles potentialités.

Côté Formel

- Technique de modélisation permettant d'organiser les classes d'un domaine selon des niveaux d'abstraction.
- L'approche objet permet de factoriser la connaissance grâce à cette arborescence

Héritage (3)

- Une classe est liée à sa super-classe par un lien kind-of (sorte-de)



- **Une instance est liée à sa classe par un lien isa (est-un)**

Héritage (4)

- Utilisation de la classe PointGraphique
  - Attributs et méthodes concernant les instances de la classe Point s'appliquent aussi aux instances de la classe PointGraphique

MAIS !!!

- L'héritage ne remet pas en cause le principe de L'encapsulation de données

Polymorphisme (1)

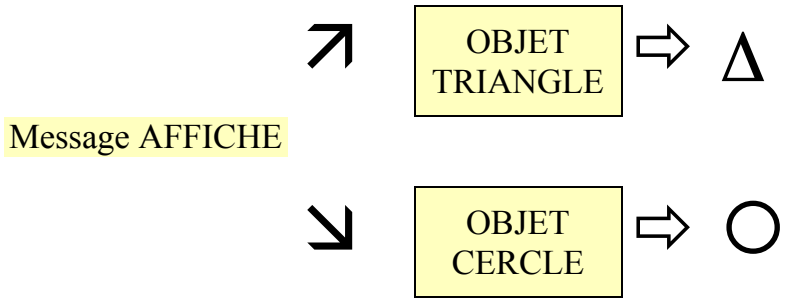
- **Langue :** Le terme polymorphisme signifie un élément qui peut prendre plusieurs formes

- **POO :** Le polymorphisme est un concept dans lequel un nom d'objet peut désigner des instances de différentes classes dans une même hiérarchie de classes.

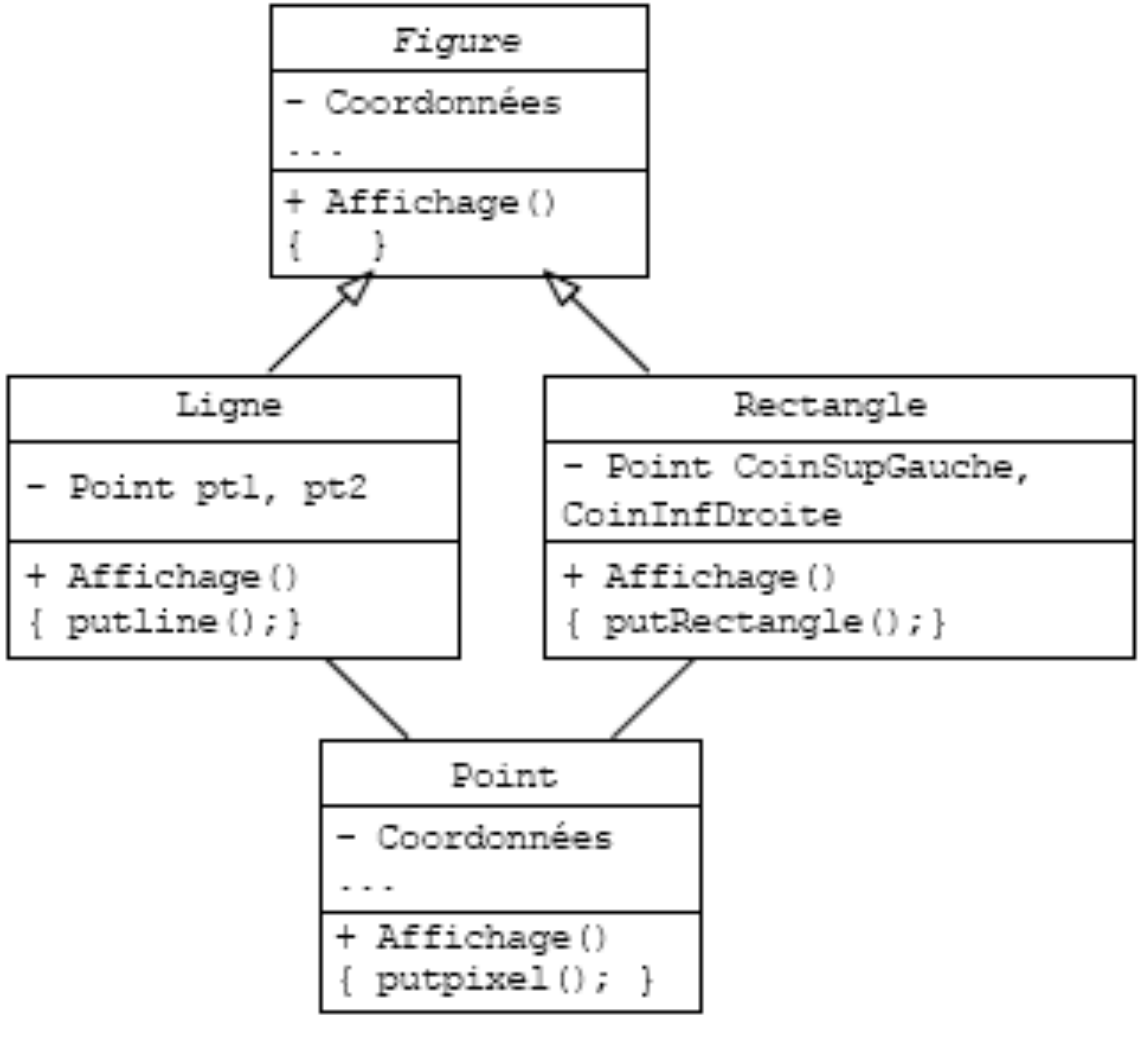
- **Pratique :** le polymorphisme des classes est appliqué à des opérations (méthodes)

Polymorphisme (2)

- Le **polymorphisme** permet de définir plusieurs formes pour une méthode commune à une hiérarchie d'objets. C'est à ***l'exécution*** que l'on détermine quelle forme appeler suivant la classe de l'objet courant.



Polymorphisme (3)



Polymorphisme (3)

- Adaptation au type des données
- Choix de la méthode à l'exécution
- En particulier, les méthodes décrites dans les interfaces sont, par définition, polymorphes puisqu'elles sont implémentées de façon indépendante dans chaque classe implémentant une même interface

MAIS !!!

Ne pas confondre polymorphisme et surcharge

Glossaire (1)

- **Abstraction** : permet de séparer l'étape de la conception de celle de la programmation et l'optimisation finales.
- **Classe** : le concept de base de la programmation orientée objets. Est une entité informatique qui regroupe des données d'un type précis sur lesquelles agissent un ensemble d'opérations.
- **Classe abstraite** : Une classe dont certains comportements ne sont pas implémentés car ils dépendent trop de la sous-classe concernée.
- **Classe de base** : On appelle généralement par ce nom une classe qui n'hérite d'aucune autre classe.

Glossaire (2)

- **Classe dérivée** : Une classe qui hérite d'une autre classe est appelée classe dérivée.
- **Classe mère** : La classe mère d'une classe est celle dont elle hérite. Les classes mères peuvent elles aussi avoir des classes mères. On parle également de parent, et de classe de base, bien que ce dernier terme soit généralement réservé aux classes mères qui n'ont pas de classe mère au-dessus d'elles.

- **Constructeur** : C'est une ***fonction membre*** particulière d'une classe. Elle est appelée à chaque ***création d'une instance de la classe*** . Son but est d'initialiser les champs de données qui doivent l'être et de réserver toute la mémoire nécessaire.

Glossaire (3)

- **Fonction membre** : Une fonction membre est une des méthodes déclarées dans une classe. Les fonctions membres accèdent librement aux champs de données de l'instance qui les appelle, du moins les champs accessibles (méthode).

- **Signature** : Liste de paramètres et type de retour d'une méthode, une procédure ou une fonction.

- **Instance** : L'instance d'une classe est un objet. La classe correspond à la définition, et l'instance, ou objet, à la déclaration. On retrouve la distinction entre l'abstrait et le concret, entre la conception et la programmation.

Glossaire (4)

- **Héritage** : c'est la méthode selon laquelle une classe dérive d'une autre. Par exemple, une classe dérivée hérite de tous les champs et de toutes les fonctions membres de ses classes mères (même les champs privés auxquels elle n'aura jamais accès directement). Selon les langages et les implémentations, l'héritage permet de redéfinir certaines caractéristiques de la classe héritée.

- **Racine** : La racine d'une hiérarchie de classe est la classe mère qui n'a pas de classe mère. On parle aussi de classe de base.
- **Surcharge** : Une fonction qui reçoit une certaine liste de paramètres peut être redéfinie autant de fois que souhaité, à condition que la liste des paramètres soit différente.

Glossaire (5)

- **Interface** :
  - Partie visible d'un objet ou d'une classe. C'est donc la liste des messages auquel un objet (une classe) est capable de répondre.
  - Sorte de classe sans attribut et ne déclarant que des méthodes abstraites (*i.e.* sans implémentation)

Glossaire (6)

- **Polymorphisme :** Mécanisme permettant à une classe d'avoir plusieurs comportements différents mais avec une seule signature.

- Dans la plupart des langages orientés objet, ce comportement n'est disponible que pour des classes appartenant au même graphe d'héritage.

Glossaire (7)

- **Flot :** Les flots sont des entités informatiques capables de recevoir ou fournir des données sous forme de suite de caractères. Ils permettent de représenter d'une manière suffisamment abstraite les périphériques d'entrée et de sortie comme le clavier, l'écran ou une ligne série. On peut aussi les connecter à des fichiers facilement (i.e canal)

- **Stream** : utilisé pour les flots de données.

Le Framework .NET

Historique Microsoft

- 1981 : Sortie du Système d'exploitation MS-DOS et du langage BASIC.
- 1990 : Première version graphique de Windows et sortie du langage Visual BASIC.
- 1995 : Microsoft met un pied dans le monde de l'Internet en sortant un navigateur Web « Internet Explorer », qui avait à l'époque un grand concurrent : « Netscape». Microsoft sort en parallèle son serveur Web IIS et propose aux développeurs une interface de développement appelée Visual Studio.
- 2000 : Début de l'ère .NET.

.NET?

- Contrairement aux idées reçues .NET n'est pas un langage ou un logiciel : c'est la nouvelle stratégie de Microsoft. .NET se présente donc comme une vision de la prochaine génération d'applications qui repose sur des standards tels que XML, HTTP, SOAP, WSDL…
- → Microsoft se repose sur quatre piliers :
  - le .NET Framework
  - les .NET Servers (futures versions Serveur de Microsoft).
  - Visual Studio .NET
  - .NET Enterprise servers (tous les logiciels serveurs comme Commerce Server, SQL Server, Content Management Server…)

.NET?

Le .NET Framework est un environnement qui est distribuable gratuitement sur toutes les versions de Windows depuis Windows 95 (et Microsoft Windows Mobile depuis la version 5). La version 3.0 est intégrée à Windows Vista et à la plupart des versions de Windows Server 2008. La dernière version connue est la 4.6.2 et est intégrée à Windows 10.

Pourquoi Framework .NET?

- En réalisant le Framework .NET, Microsoft voulait tout d'abord sortir de l'enfer des technologies COM : en effet toutes les versions COM devaient supporter les anciennes versions ce qui était assez lourd à gérer.

- De plus la communication pour accéder aux objets COM se faisait toujours sur le même port d'écoute

- Cela ne posait pas de problème pour l'intranet de l'entreprise, mais lorsqu'une entreprise voulait utiliser un objet COM d'une autre entreprise, cela posait souvent des problèmes car la communication ne pouvait pas s'effectuer pour des raisons de sécurité (Firewall de l'entreprise).

Pourquoi Framework .NET?

- Mais ce n'est pas pour autant que les technologies COM sont mortes : en effet il est tout à fait concevable de développer un objet COM puis ensuite de l'utiliser en .NET.

Pourquoi Framework .NET?

- Pour résoudre ce dernier problème de sécurité avec les objets COM, le Framework propose les Services Web plus couramment appelés les WebServices. L'avantage des WebServices est que la communication entre le client et le serveur (ici le WebService) se fait via le protocole http, c'est-à-dire via le port 80 : il n'y a donc plus de problèmes de communication entre les entreprises. De plus les informations émises par le Webservice sont sous format XML donc peuvent être traitées par la quasi totalité des langages tels que le C, C++, Java, Perl, Python, PHP, Cobol… et bien sûr les langages .NET.

Pourquoi Framework .NET?

- Ce qui fait donc la force de .NET est qu'il regroupe plusieurs technologies tels que COM, Applications Web (ASP.NET) , Applications Windows (Windows Form) et Applications Mobiles (Compact Framework)…

- Pour ce qui est des applications Web, le changement est assez net entre l'ASP et l'ASP.NET, car le langage n'est plus interprété mais compilé. En ASP, le développeur n'avait pas le choix du langage alors qu'en ASP.NET il a le choix.
- En effet la plateforme .NET est multi-langages ce qui est une grande nouveauté par rapport à ses concurrents directs.

Pourquoi Framework .NET?

- Actuellement le Framework .NET supporte une vingtaine de langages dont le C#, VB.NET, J#, COBOL, Eiffel#, PERL.NET…
- De plus avec le Framework, Microsoft a voulu mettre fin aux nombreux problèmes causés par des dll : il était impossible d'avoir plusieurs versions de dll en même temps et cela posait donc de gros problèmes de compatibilité pour les applications.
- En effet lorsque par exemple au sein d'une entreprise on changeait de version de dll, certaines applications ne marchaient tout simplement plus car elles n'étaient pas compatibles avec la nouvelle version de la dll.

Pourquoi Framework .NET?

- En .NET lorsque vous créez une application Windows ou Web, une assemblée couramment appelée **assembly** est automatiquement créée. Une **assembly** est en fait le conteneur physique de classes de votre projet et donc ces classes peuvent être utilisées par plusieurs applications. Et il est bien sûr possible d'avoir plusieurs versions de la même **assembly**.

Eléments du Framework .NET

- Contrairement aux API Windows, le Framework .NET est totalement objet : ce n'est plus en effet une simple liste de méthodes comme les API Windows.
- De plus les classes du Framework sont ordonnées hiérarchiquement. En effet, toutes les classes concernant la manipulation des fichiers XML se trouveront dans un namespace « **System.XML** » ;
- les classes permettant de faire de la manipulation de données seront dans le namespace « **System.Data** » …

Eléments du Framework .NET

- Le Framework intègre de base des classes pour la connexion aux bases de données via ADO.NET, OLEDB, ODBC, ODBC.NET… Ces classes permettent donc de se connecter à toutes les bases de données existantes sur le marché telles que SqlServer, Oracle, Access, Sybase… Il existe aussi actuellement des drivers optimisés pour SqlServer et Oracle.

Eléments du Framework .NET

- Il existe 3 versions téléchargeables du Framework qui sont totalement gratuites:
- « Framework Redistribuable» : Version qui contient les classes de bases ainsi que l'environnement d'exécution .NET.
- « Framework SDK » : Version identique à la version précédente à la différence près que la version SDK contient les compilateurs CSharp et VB.NET ainsi que des utilitaires. La version SDK va donc permettre de créer des applications .NET.
- Silverlight (version allégée du Framework) : Plutôt un plugin web, sorte d'alternative à Adobe Flash

Les bénéfices de .NET pour les entreprises

- Les avantages de .NET pour les entreprises sont multiples et variés : tout d'abord, la productivité, c'est à dire le développement des applications, est plus rapide. Il y a un gain de productivité en .NET car tout est objet en .NET. Lors du développement des composants, il est possible de les réutiliser dans plusieurs applications, ce qui évite de développer les mêmes méthodes à chaque fois.

Les bénéfices de .NET pour les entreprises

- Exemple un composant « Employé » qui sera utilisé par toutes les applications de l'entreprise : ce composant permettra donc d'avoir des informations sur les employés et de modifier ces informations. Le composant « Employé » fait donc le lien entre la couche de données de l'entreprise (Bases de données) et la couche applicative.

Les bénéfices de .NET pour les entreprises

- Pour faciliter la tâche du développeur, Microsoft propose un environnement de développement appelé Visual Studio .NET. Avec Visual Studio le développement .NET devient beaucoup plus rapide et simple : en effet Visual Studio dispose d'un éditeur WYSIWYG et il est entièrement RAD (Rapid Application Development): C'est à dire que pour utiliser un composant, il suffit de le faire glisser sur la page ou la form (Principe du Drag And Drop).

Les bénéfices de .NET pour les entreprises

- Visual .NET intègre aussi plusieurs outils et le développeur n'est plus obligé d'utiliser plusieurs logiciels pour développer ses applications. Par exemple, via le Solution Explorer de Visual .NET, le développeur peut accéder à sa base de données que ce soit un SqlServer, Oracle ou autre… Et, de plus, Visual .NET permet de développer des applications Web, Windows, WebService, applications mobiles.

Les bénéfices de .NET pour les entreprises

- Le FrameWork .NET est multi-langages, donc passer d'un langage à un autre ne pose pas de problème. Le plus dur lorsque un développeur change de langage n'est pas l'apprentissage de la syntaxe, mais l'utilisation des fonctionnalités propre au langage. Or en .NET, tous les langages utilisent les mêmes fonctionnalités, c'est à dire les mêmes classes de base donc le changement ne pose aucun problème au développeur.

Les bénéfices de .NET pour les entreprises

- Un autre point très important dans la stratégie .NET de Microsoft surtout pour les entreprises, c'est la sécurité. En effet, Microsoft notamment avec leurs serveurs a été beaucoup critiqué par les professionnels de la sécurité: Tout est désormais fermé par défaut sur les .NET Servers. Par exemple, pour faire fonctionner un Serveur Web, il faut donner les droits nécessaire à l'utilisateur lançant le processus du Serveur Web.

Les bénéfices de .NET pour les entreprises

- De plus, les nouvelles versions du Serveur Web IIS sont beaucoup plus fiables, plus robustes, plus sécurisées et surtout plus performantes, car IIS passe d'une architecture multi-threads à une architecture multi processus comme le serveur Web Apache qui est reconnu pour ses performances.

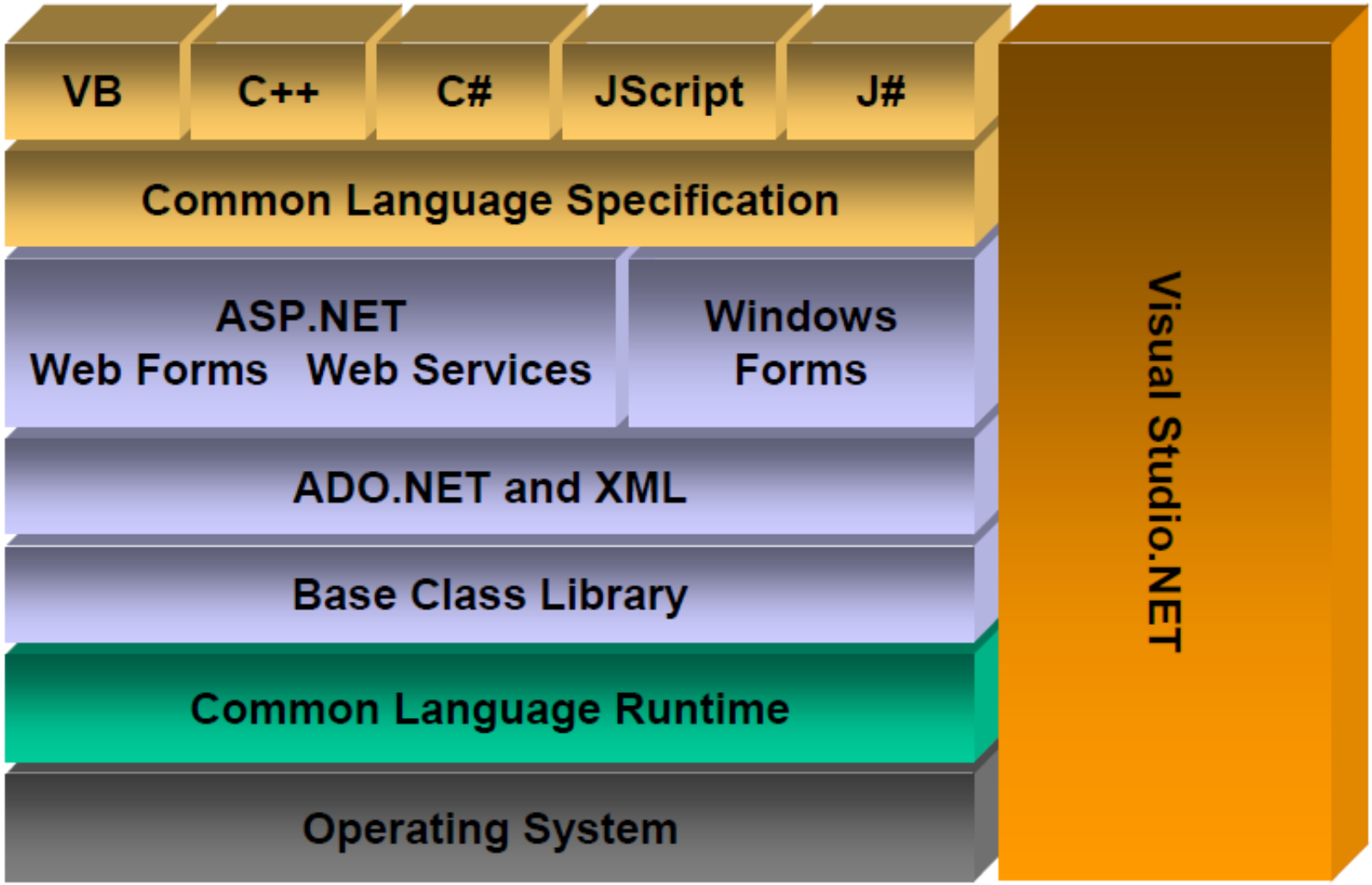
.NET et Java

- Le Framework est assez similaire à l'environnement de Java, le Java Runtime Environnement (JRE).
- En effet comme la JRE, le Framework dispose d'une machine virtuelle appelé la CLR (Common Language Runtime) ainsi que des classes de base mises à disposition du développeur.
- Alors .NET est il un clone de Java ?

.NET et Java

- Non!
- la Plateforme .NET est la seule pour l'instant à être multi langages. Cela a été rendu possible par la définition de format de type et de description standard. Cette définition s'appelle la CLS (Common Language Specification).
- La stratégie .NET est axée sur les Webservices et sur les mobiles à partir de 2002 avec le Compact Framework, alors que Java s'adresse directement aux applications mobiles, tout en offrant un bon support des WebServices.

Les couches du Framework



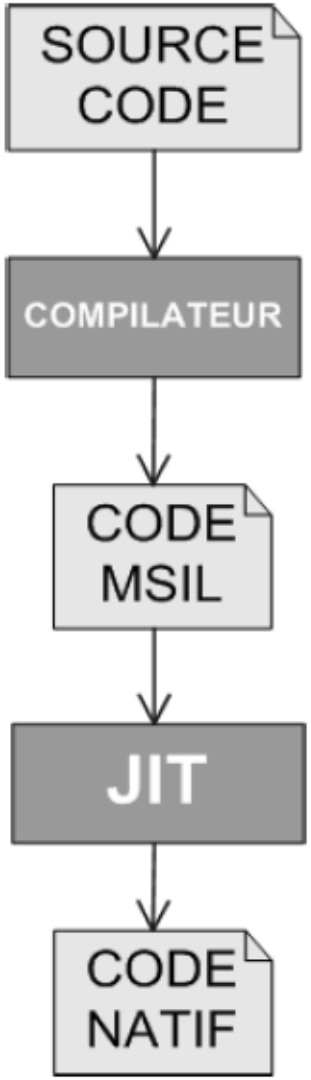
Common Language Runtime

- La CLR est un des piliers du Framework .NET
- Comme Sun avec Java, Microsoft avec .NET a choisi de se munir d'une machine virtuelle appelée la CLR (Common Language Runtime) et d'un code intermédiaire nommé MSIL (Microsoft Intermediate Language).
- La CLR se place juste au dessus du Système d'exploitation et c'est la CLR qui va exécuter les applications .NET puis gérer la gestion de mémoire et la sécurité de l'application.

Common Language Runtime

- Dans des langages traditionnels tels que le C ou le C++, le code était compilé dans un code natif, c'est-à-dire un code machine qui était propre au processeur de la machine.
- La différence entre ces langages et les langages .NET est qu'en .NET les langages ne sont pas compilés en langage machine, mais en code intermédiaire.

Common Language Runtime



Common Language Runtime

- Ce n'est qu'au moment de l'exécution de l'application que la CLR va interpréter le code intermédiaire (MSIL) en code machine via son compilateur JIT (Just In Time) : le code sera donc compilé à la volée.

Common Language Runtime

- Chaque langage possède son propre compilateur : pour C# le compilateur se nomme « csc », pour VB.NET c'est le « vbc »…
- Le compilateur ne va donc pas compiler le programme en code machine mais en code intermédiaire appelé le MSIL : Si nous compilons deux programmes, un en C# et le deuxième en VB.NET, nous obtiendrons le même code.
- C'est pour cette raison que tous les langages .NET ont les mêmes performances. Il n'y a donc pas un langage qui est plus performant qu'un autre.

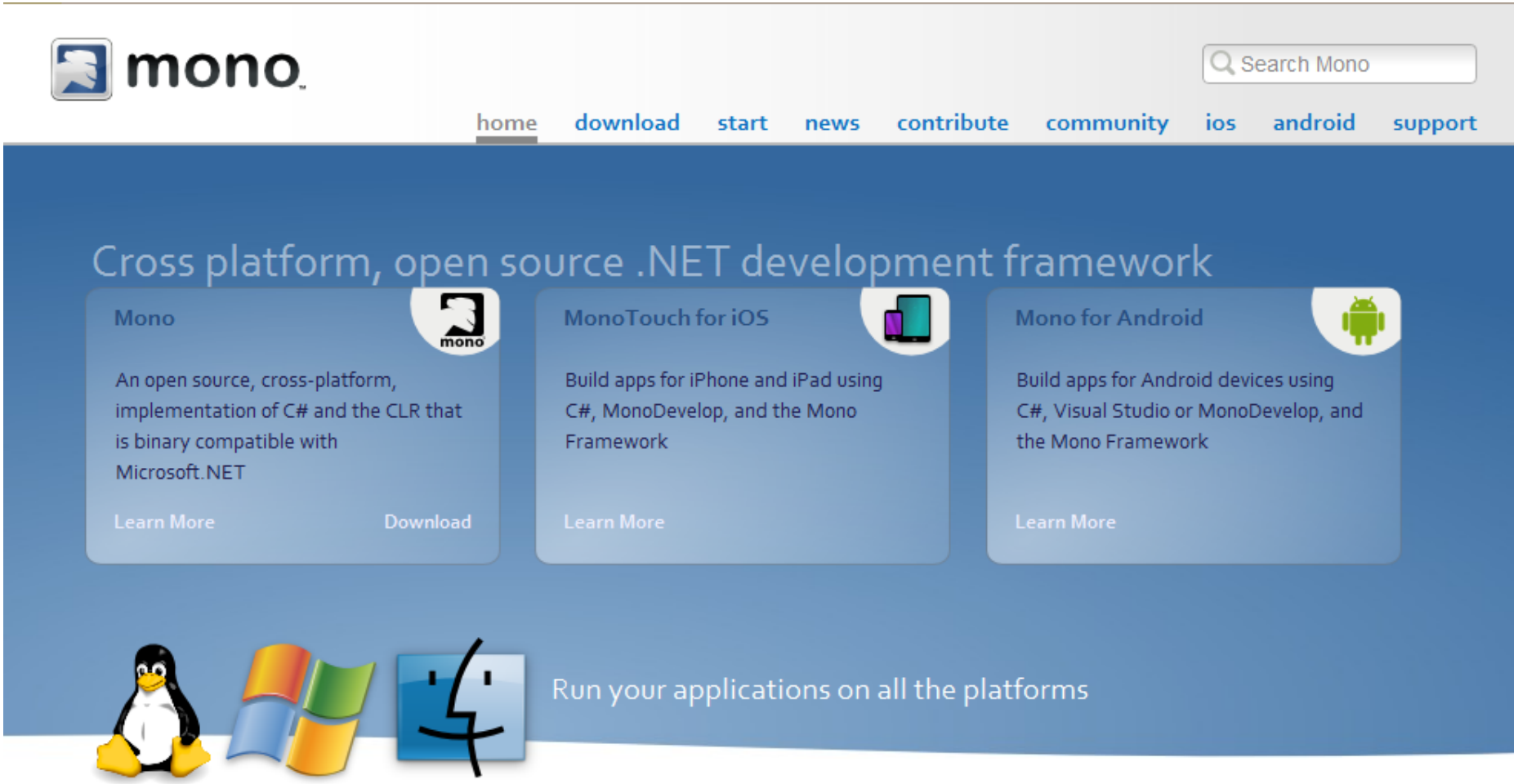
Common Language Runtime

- Ce code intermédiaire donne aux langages .NET une grande portabilité, car le code intermédiaire n'est pas propre à la plateforme ou au processeur et ce n'est que lors de l'exécution de l'application que le code intermédiaire est compilé à la volée en code machine.
- Microsoft nomme ce principe « Execute on Many Platforms » alors que SUN l'appelle « Write One, Run Anywhere » (WORA).

Common Language Runtime

- Dans l'absolu on pourrait faire fonctionner les applications .NET sur n'importe quelle plateforme mais le Framework n'est implémenté que sur les versions de Windows à partir de Windows 98.
- De plus il existe des projets d'implémentation du Framework sur d'autres plateformes telles que Linux, Mac, Android ou encore iOS avec le projet « Mono ».

Common Language Runtime



http://www.mono-project.com

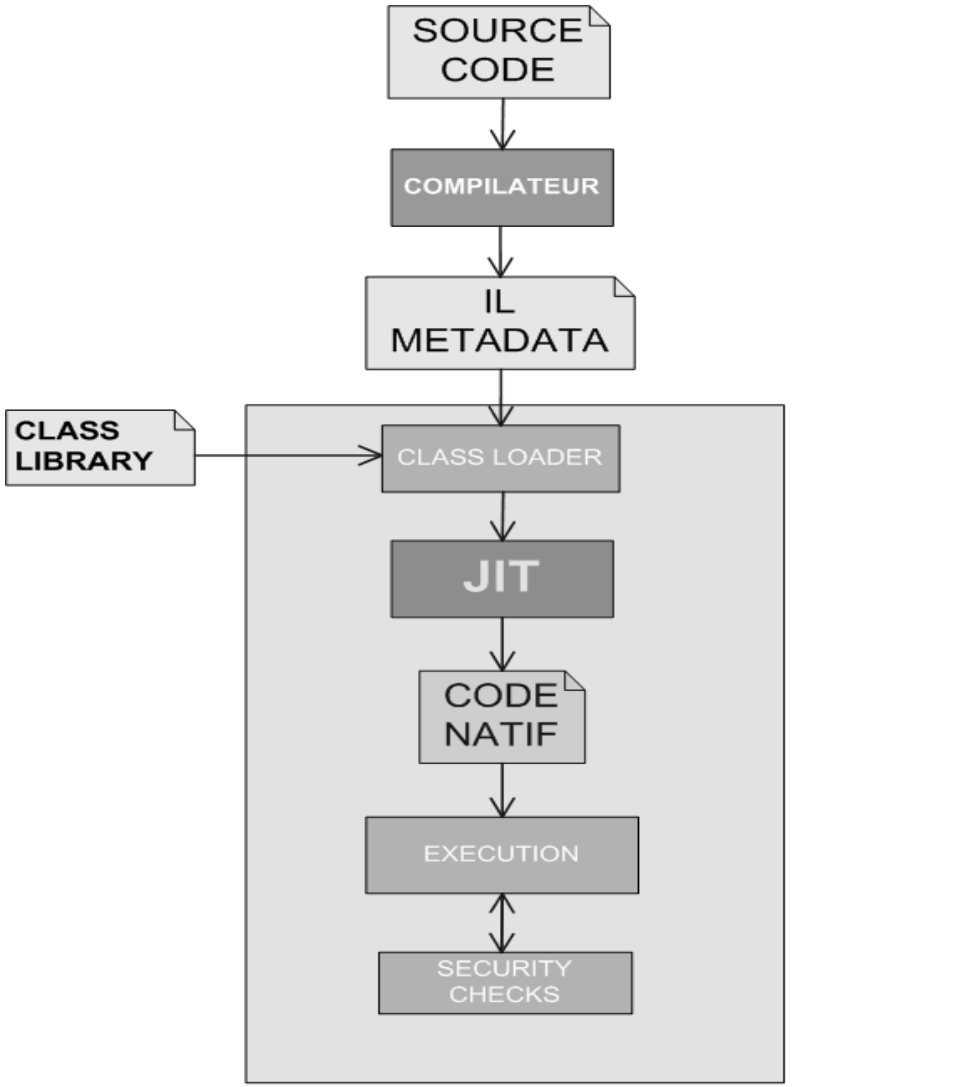
Common Language Runtime

- L'environnement .NET comprend aussi un mécanisme de « ramasse-miettes » et gère aussi la sécurité de l'application tout au long de son exécution.
- Le mécanisme de ramasse-miettes de la CLR permet de contrôler le cycle de vie des objets en libérant la mémoire occupée par un objet qui n'est plus référencé dans le programme.
- Ce mécanisme se nomme le « Garbage Collector » et permet donc de libérer la mémoire qui n'est plus utilisée par l'application.

Common Language Runtime

- La CLR gère aussi la sécurité des applications .NET et le développeur peut donc décider de donner des droits restreints à l'application. Par exemple on peut très bien interdire à l'application l'accès au disque dur de la machine.

Common Language Runtime



CLR : Common Language Runtime

Common Language Runtime

- Par rapport au schéma de la diapositive 81, les compilateurs du Framework qu'il s'agisse du compilateur C#, VB.NET… ne compilent pas l'application en un simple fichier contenant du code MSIL mais en une assembly.

- Une assembly contient du langage intermédiaire, des Metadata ainsi qu'un Manifest.

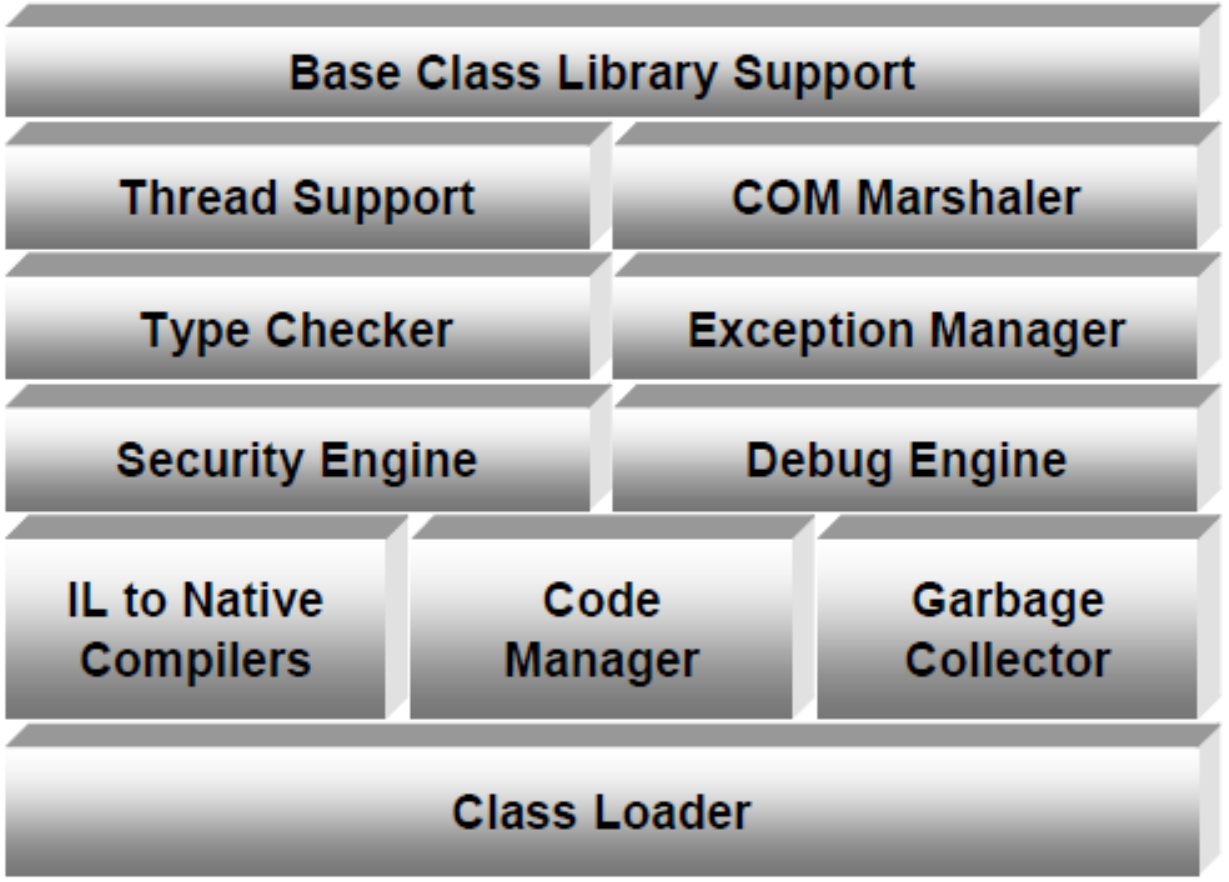
- Lors de l'exécution de l'application, la CLR prend le relais en chargeant le type et les classes nécessaires puis fait appel au compilateur Just In Time (JIT) pour compiler le code MSIL en code natif.

Common Language Runtime

- Durant l'exécution de l'application, la CLR gère la sécurité de l'application et décide si elle aura les droits nécessaires pour exécuter les routines voulues.

Common Language Runtime

- Les différents éléments qui sont appelés lors de l'exécution d'une application .NET



Schema: source Microsoft

Common Language Runtime

- Le code MSIL de l'application est tout d'abord compilé en code natif. Mais le travail de la CLR ne s'arrête pas là. En effet, le « Garbage Collector » permet de libérer l'espace mémoire des objets qui ne seront plus référencés, tout au long de l'exécution de l'application.
- Le « Garbage Collector » ne libère de l'espace mémoire que lorsque le processeur le permettra. Il attend que les ressources processeurs soient de niveau assez bas pour pouvoir libérer cet espace. Il est possible de forcer le Garbage Collector à libérer de l'espace mémoire, mais ce n'est pas conseillé car cela pourrait dégrader les performances de l'application au lieu de les améliorer.

MSIL

- MSIL est le langage intermédiaire de Microsoft : Microsoft Intermediate Language. Tous les compilateurs des langages .NET génèrent du code MSIL qui est ensuite interprété par la CLR.

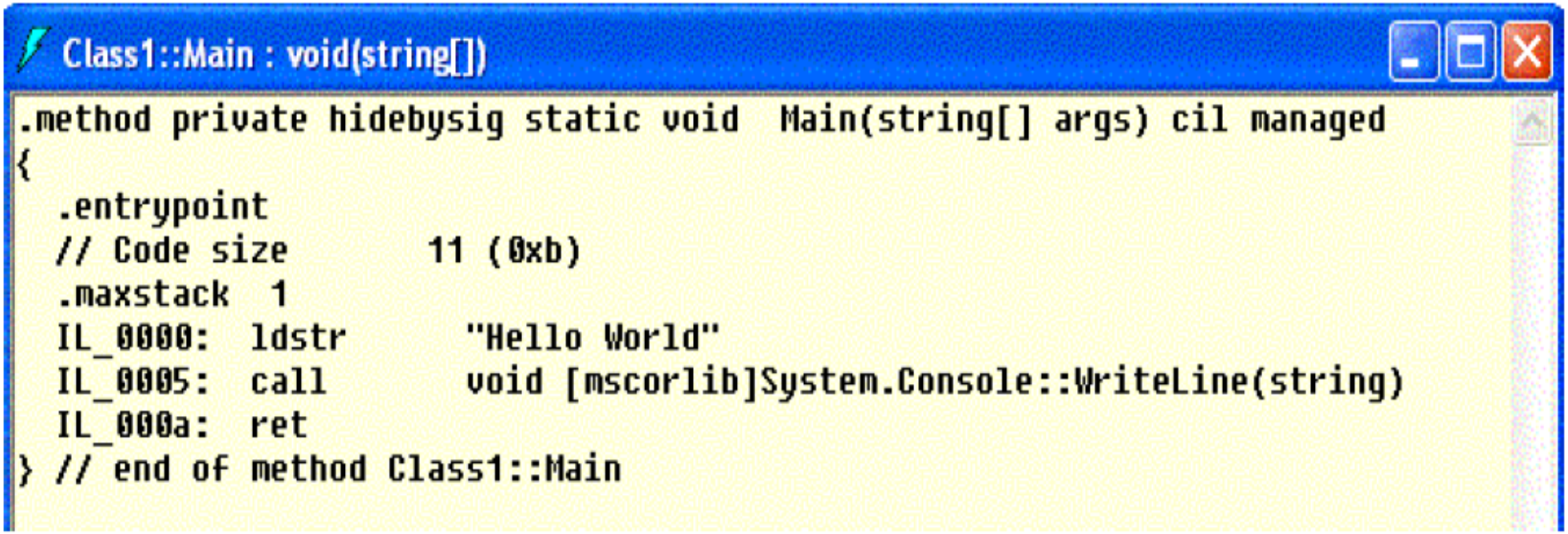
- Pour lire ou générer du code MSIL, il faut deux utilitaires : ildasm et ilasm.

MSIL

- Ildasm est en fait un désassembleur qui permet de lire du code MSIL à partir d'une assembly (fichier .dll). Ildasm se trouve dans le répertoire des utilitaires de Visual Studio .NET.
- Ilasm est donc l'assembleur et il se trouve dans le répertoire des utilitaires du Framework.

MSIL

- Exemple : Assembly d'une application console qui se contente d'afficher « Hello World ».



MSIL

- Pour protéger les applications d'un possible désassemblage (c'est à dire qu'une personne puisse retrouver le code source à partir du fichier MSIL), il est possible d'utiliser des Obfuscators qui permettent de modifier le code intermédiaire.

- Voir la liste ici :

- http://en.wikipedia.org/wiki/User:Scatophaga/Comparison_of_.NET_obfuscators

Assemblies

- Une Assembly est le conteneur physique des classes qui seront utilisées par la ou les applications. L'Assembly est automatiquement générée à chaque fois qu'il y a création d'une DLL ou d'un exécutable.

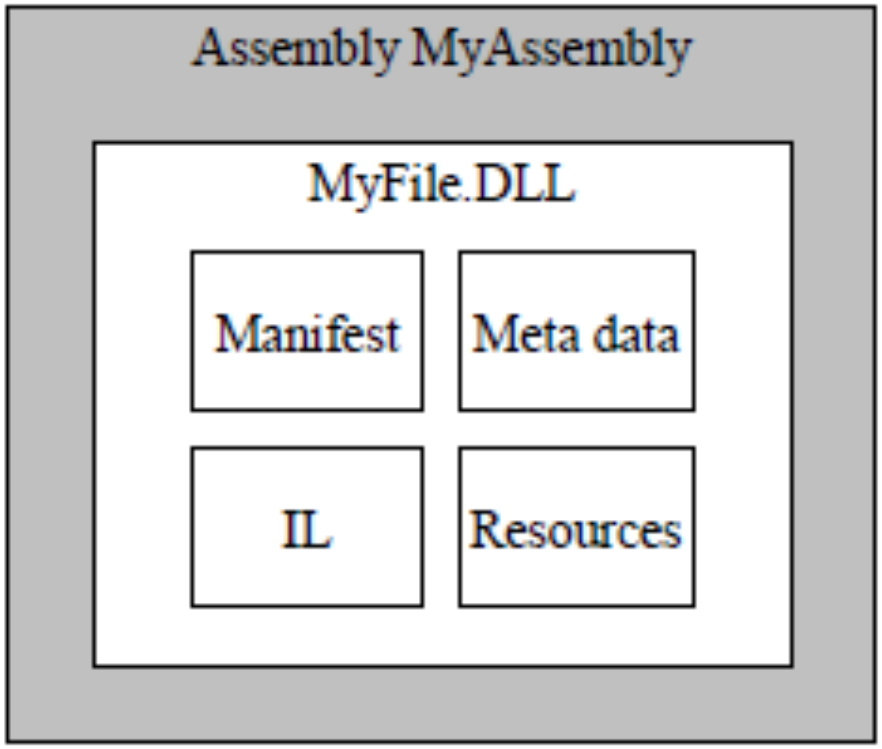
- Une Assembly est en fait une collection de plusieurs fichiers : Metadata, Manifest, IL et ressources

Assemblies

- Le Manifest définit toutes les exigences de contrôle de version, l'auteur de l'Assembly, les autorisations, et les dépendances avec les autres Assemblies (et pour chaque dépendance il y a le numéro de version de l'Assembly, car il peut avoir plusieurs versions de la même Assembly, contrairement aux DLL Windows).

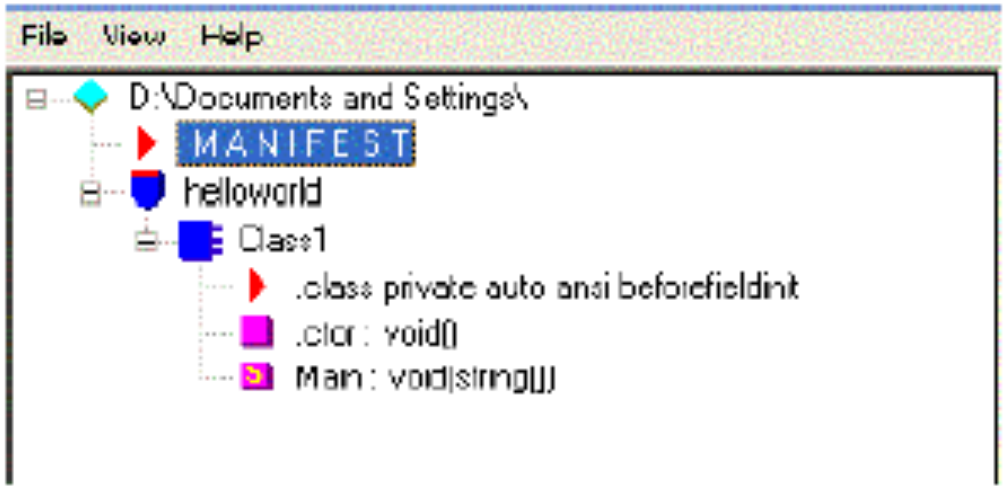
- Les métas données de l'Assembly permettent d'interroger l'Assembly sur ses types de données, méthodes… Une Assembly contient enfin le code MSIL de l'application qui sera ensuite compilé en code natif par la CLR lors de l'exécution de l'application.

Assemblies


Assembly MyAssembly
MyFile.DLL
Manifest
Meta data
IL
Resources

Assemblies

- Reprenons notre exemple d'application « HelloWorld » en utilisant le désassembleur **ildasm** pour pouvoir voir le Manifest.
- Lorsque nous ouvrons l'Assembly de l'application, la fenêtre suivante s'ouvre :

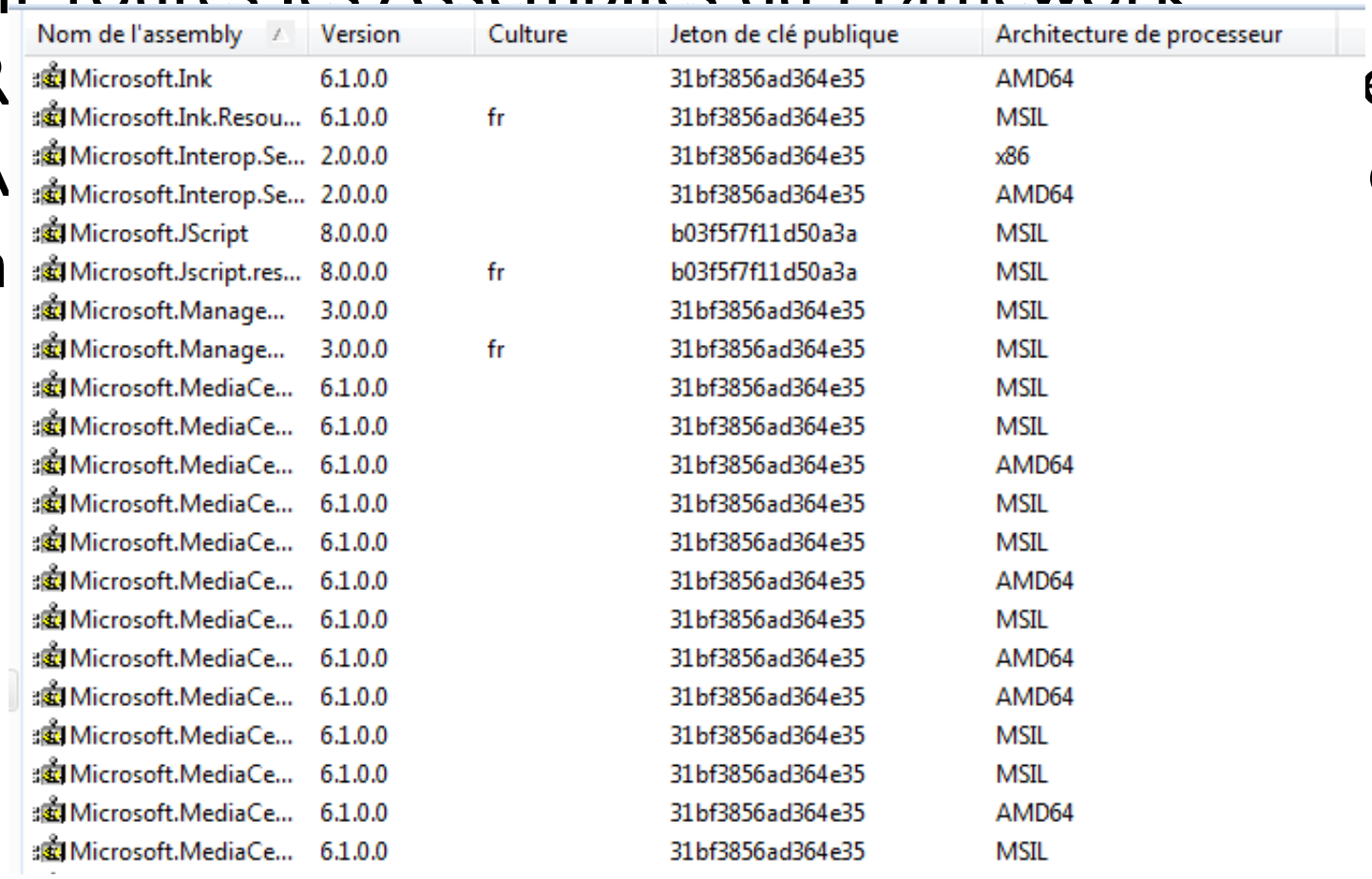


Assemblies

- Toute cette arborescence constitue l'Assembly. A chaque noeud, il y a une icône. Voici la légende de ces icônes :

GAC

- GAC veut dire Global Assembly Cache. La GAC va en fait contenir toutes les Assemblies du Framework
- La CLR début.
- Les A ertoire C:\Win



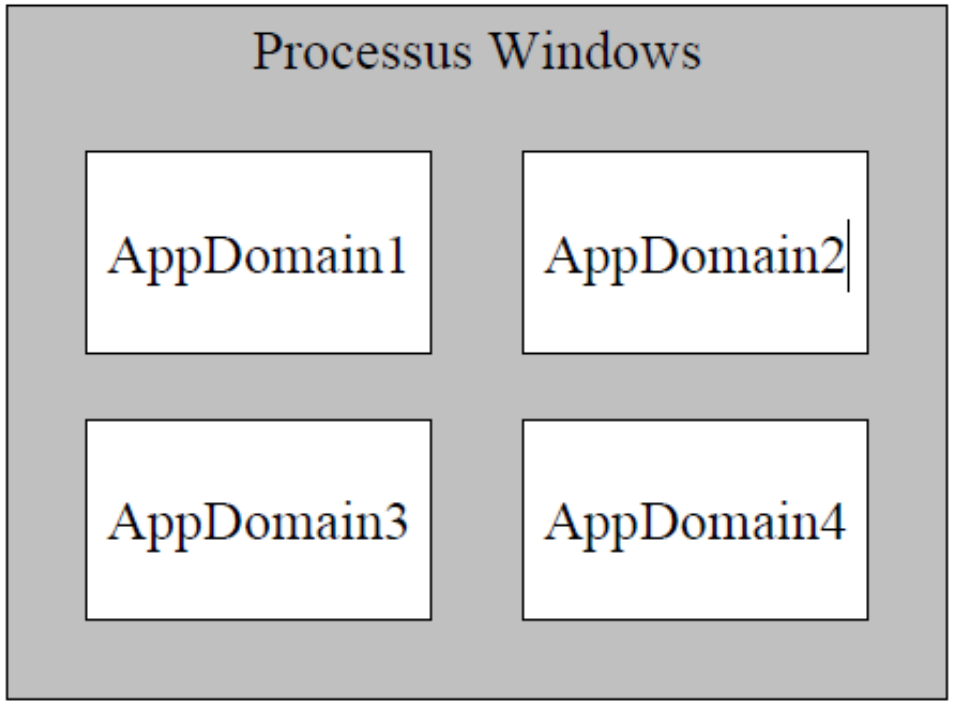
Nom de l'assembly	Version	Culture	Jeton de clé publique	Architecture de processeur
Microsoft.Ink	6.1.0.0		31bf3856ad364e35	AMD64
Microsoft.Ink.Resou...	6.1.0.0	fr	31bf3856ad364e35	MSIL
Microsoft.Interop.Se...	2.0.0.0		31bf3856ad364e35	x86
Microsoft.Interop.Se...	2.0.0.0		31bf3856ad364e35	AMD64
Microsoft.JScript	8.0.0.0		b03f5f7f11d50a3a	MSIL
Microsoft.Jscript.res...	8.0.0.0	fr	b03f5f7f11d50a3a	MSIL
Microsoft.Manage...	3.0.0.0		31bf3856ad364e35	MSIL
Microsoft.Manage...	3.0.0.0	fr	31bf3856ad364e35	MSIL
Microsoft.MediaCe...	6.1.0.0		31bf3856ad364e35	MSIL
Microsoft.MediaCe...	6.1.0.0		31bf3856ad364e35	MSIL
Microsoft.MediaCe...	6.1.0.0		31bf3856ad364e35	AMD64
Microsoft.MediaCe...	6.1.0.0		31bf3856ad364e35	MSIL
Microsoft.MediaCe...	6.1.0.0		31bf3856ad364e35	MSIL
Microsoft.MediaCe...	6.1.0.0		31bf3856ad364e35	AMD64
Microsoft.MediaCe...	6.1.0.0		31bf3856ad364e35	MSIL
Microsoft.MediaCe...	6.1.0.0		31bf3856ad364e35	AMD64
Microsoft.MediaCe...	6.1.0.0		31bf3856ad364e35	AMD64
Microsoft.MediaCe...	6.1.0.0		31bf3856ad364e35	MSIL
Microsoft.MediaCe...	6.1.0.0		31bf3856ad364e35	MSIL
Microsoft.MediaCe...	6.1.0.0		31bf3856ad364e35	AMD64
Microsoft.MediaCe...	6.1.0.0		31bf3856ad364e35	MSIL

Les domaines d'application

- Sous Windows, les applications fonctionnent dans un processus (c'est-à-dire des ressources et des threads). Cette architecture pose un problème car il faut beaucoup de ressources pour la création et gestion des processus.

- Sous .NET, il y a la notion de domaine d'application. Un domaine d'application est identique à un processus mais plusieurs domaines d'application peuvent s'exécuter dans un processus ce qui a pour conséquence un gain de performance et de ressources non négligeable.

Les domaines d'application



Classes de bases

- le Framework met à disposition des développeurs une boîte à outils qui est constituée de classes qui sont accessibles par tous les langages .NET.
- En effet, seule la syntaxe change entre les différents langages .NET, mais les classes restent les mêmes. Il est donc très facile de passer d'un langage .NET à un autre (par exemple du C# au VB.NET).
- Les classes du Framework sont organisées selon une structure hiérarchisée. En effet, les classes sont regroupées selon leur utilité dans des namespaces. Cela facilite grandement le travail du développeur.

Norme du Framework

- Les classes du Framework respectent la normes suivante :
- La première lettre d'une classe est en majuscule, les autres en minuscules exemple : la classe « Console » du namespace «System).
- Si le nom d'une classe comporte plusieurs mots, chaque mot doit commencer par une majuscule, le reste en minuscules (exemple : la classe « DateTime » du namespace « System »).
- Si le nom d'une classe ne contient que deux lettres, alors ces deux lettres pourront être en majuscule (exemple : « System.IO »).

Principaux Namespace

- **System**
  - Le namespace « System » regroupe tout d'abord tous les types de bases : String, Int, Bool… C'est donc pour cela qu'il faut toujours inclure dans les applications le namespace « System » à moins de vouloir taper le chemin complet des types (par exemple «System.String »).
  - Le namespace « System » contient aussi une classe qui est très importante pour les applications console : c'est la classe « Console ».A partir de cette classe, il y a toutes les méthodes qui permettent d'afficher du texte, de prendre une valeur entrée par un utilisateur..

Principaux Namespaces

- **System.IO**
  - Le namespace « System.IO » regroupe toutes les classes qui permettent de lire des fichiers, écrire et modifier un fichier…
  - Par exemple, la classe « TextReader » du namespace « System.IO » permet de lire un fichier texte et afficher son contenu.

Principaux Namespaces

- **System.Data**
  - Le namespace « System.Data » regroupe toutes les classes et types qui permettent d'accéder à une base de données.

Principaux Namespaces

- **System.Collections**
- Le namespace « System.Collections » regroupe toutes les classes qui permettent de manipuler des collections d'objet. Une collection d'objet permet de stocker dans un ensemble logique des objets.
- A l'instar des tableaux, les collections peuvent être redimensionnées. Il est possible de gérer des piles FIFO avec la classe « Queue » et des piles LIFO avec la classe « Stack ».

Principaux Namespaces

- **System.Net**
- Le namespace « System.Net » regroupe toutes les classes utiles à l'accès au réseau : socket, requête http…

Principaux Namespaces

- **System.Reflection**
- Le namespace « System.Reflection » regroupe toutes les classes utiles à l'interrogation des métas données. La réflexion permet de connaître toutes les méthodes d'une classe, ainsi que ses paramètres, constructeurs… L'utilitaire « Wincv » qui permet d'effectuer des recherches sur les classes du Framework et d'afficher les informations sur ces classes utilise la réflexion.

Principaux Namespaces

- **System.XML**
- Le namespace « System.Xml » regroupe toutes les classes qui permettent de manipuler des fichiers XML et de les modifier.
- En effet, le développeur aura à disposition plusieurs classes qui lui permettront de parcourir les différents noeuds d'un fichier XML, de lire et de modifier les éléments et ses attributs.

ASP.NET et Windows Form

- **ASP.NET**
- ASP.NET permet la création d'application Web au sein de .NET et non pas de pages Web comme avec ASP. En .NET tout est objet, donc en ASP.NET une page est un objet, un bouton est un objet… : ces objets s'appellent des contrôles Web.
- Lorsqu'un développeur voudra insérer un bouton sur son application Web, il devra insérer le contrôle button. C'est ensuite ASP.NET qui se chargera de créer le code HTML correspondant au contrôle.

ASP.NET et Windows Form

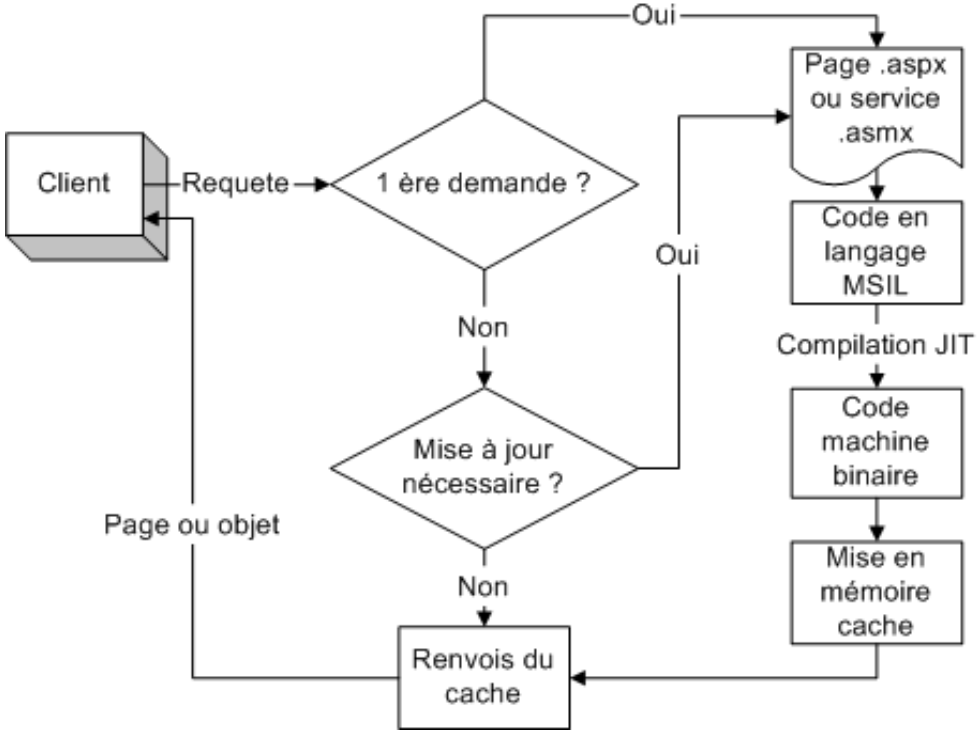
- **ASP.NET**
- En ASP, le code et l'interface (HTML) n'étaient pas séparés et se trouvaient donc sur le même fichier : cela pose beaucoup de problème car par exemple le designer d'un site ne connaît pas forcément le langage VB Script. En ASP.NET, l'interface et le code de l'application Web sont séparés ce qui facilite le travail du designer et du développeur.

ASP.NET et Windows Form

- **ASP.NET**
- L'une des grandes forces de ASP.NET est sa portabilité face aux différents clients : en effet, ASP.NET se chargeait de créer le code HTML correspondant aux différents contrôles (Bouton, Label…). Mais le code HTML généré va dépendre du navigateur clients et donc les applications Web sous .NET pourront être lus sous la plupart des navigateurs Web que ce soit un Internet Explorer, Firefox, Chrome, Safari, Opera…

ASP.NET et Windows Form

- **ASP.NET langage compilé**
- Contrairement à ASP qui était un langage interprété, ASP.NET est un langage compilé comme le montre le schéma suivant :



ASP.NET et Windows Form

- **ASP.NET langage compilé**
- Le code n'est compilé qu'une seule fois lors de la première demande. Dans le cas contraire, les applications Web ne seraient pas du tout performantes s'il fallait compiler la page à chaque demande du client !
- Le processus ASP.NET va vérifier si une mise à jour est nécessaire, c'est-à-dire si il y a eu une modification du code, et compilera donc la page si nécessaire.

ASP.NET et Windows Form

- **ASP.NET**
- Vidéo Microsoft Techdays 2014

- http://www.microsoft.com/france/mstechdays/programmes/2014/fiche-session.aspx?ID=5f7102fa-7fc1-4aed-b916-00a93034l5c3

- Présentation Powerpoint

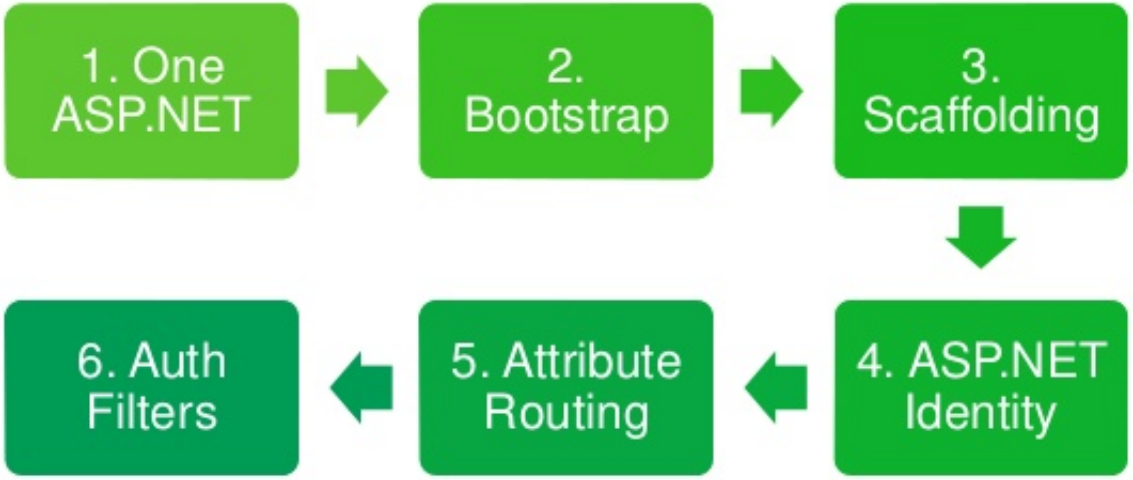
- http://www.louizi.com/MicrosoftTechdays2014.pptx

ASP.NET et Windows Form

- **ASP.NET**
- Termes en relations avec la Microsoft Techdays Keynote

ASP.NET Termes

- **MVC 5**
- ASP.NET MVC 5 est un framework pour construire des applications web évolutives, basées sur des normes en utilisant les modèles de conception bien établis et la puissance de ASP.NET et le .NET Framework.



ASP.NET Termes

- **MVC 5**
- Scaffolding
  - Scaffolding is a technique used by many MVC frameworks like ASP.NET MVC, Ruby on Rails, Cake PHP and Node.JS etc., to generate code for basic CRUD (create, read, update, and delete) operations against your database effectively. Further you can edit or customize this auto generated code according to your need.
  - Scaffolding consists of page templates, entity page templates, field page templates, and filter templates. These templates are called Scaffold templates and allow you to quickly build a functional data-driven Website.

ASP.NET Termes

- **MVC**
  - Bootstrap

ASP.NET Termes

- **Active Directory**
  - C'est la mise en œuvre par Microsoft des services d'annuaire LDAP pour les systèmes d'exploitation Windows. L'objectif principal d'Active Directory est de fournir des services centralisés d'identification et d'authentification à un réseau d'ordinateurs utilisant le système Windows. Il permet également l'attribution et l'application de stratégies, la distribution de logiciels, et l'installation de mises à jour critiques par les administrateurs.

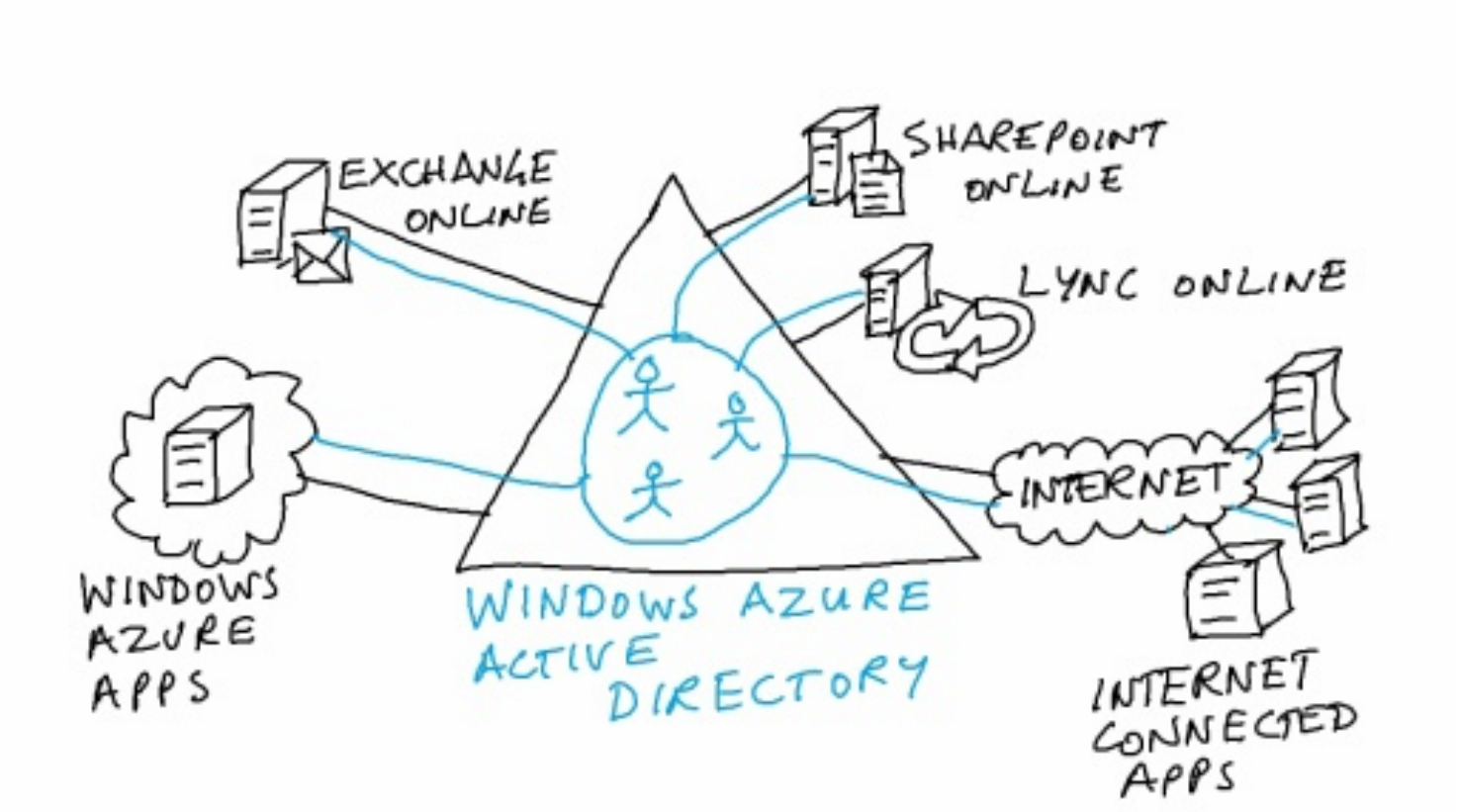
  - AD utilise **LDAP** (Lightweight Directory Access Protocol) versions 2 et 3, La version Microsoft de **Kerberos** et DNS.

ASP.NET Termes

- **Active Directory**
  - **LDAP** : Lightweight Directory Access Protocol (LDAP) est à l'origine un protocole permettant l'interrogation et la modification des services d'annuaire. Ce protocole repose sur TCP/IP. Il a cependant évolué pour représenter une norme pour les systèmes d'annuaires, incluant un modèle de données, un modèle de nommage, un modèle fonctionnel basé sur le protocole LDAP, un modèle de sécurité et un modèle de réplication.
  - **Kerberos** :Kerberos est un protocole d'authentification réseau qui repose sur un mécanisme de clés secrètes (chiffrement symétrique) et l'utilisation de tickets, et non de mots de passe en clair, évitant ainsi le risque d'interception frauduleuse des mots de passe des utilisateurs. Créé au Massachusetts Institute of Technology, il porte le nom grec de Cerbère, gardien des Enfers

ASP.NET Termes

- **Active Directory**
  - **Azure Active Directory**



ASP.NET Termes

- **JSON**
  - JSON (JavaScript Object Notation – Notation Objet issue de JavaScript) est un format léger d'échange de données. Il est facile à lire ou à écrire pour des humains. Il est aisément analysable ou généré par des machines. Il est basé sur un sous-ensemble du langage de programmation JavaScript
  - JSON est un format texte complètement indépendant de tout langage, mais les conventions qu'il utilise seront familières à tout programmeur habitué aux langages descendant du C, comme par exemple : C lui-même, C++, C#, Java, JavaScript, Perl, Python et bien d'autres. Ces propriétés font de JSON un langage d'échange de données idéal.

ASP.NET Termes

- **JSON**
  - Exemple

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

ASP.NET Termes

- **JSON**
  - Exemple (le même en XML)

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

ASP.NET Termes

- **Web Forms**
- Web Forms sont des pages que les utilisateurs demandent à l'aide de leur navigateur. Ces pages peuvent être écrites en utilisant une combinaison de HTML, script client, les contrôles serveur et code serveur. Lorsque les utilisateurs demandent une page, elle est compilée et exécutée sur le serveur par le framework, et puis le framework génère le balisage HTML que le navigateur peut rendre. Une page Web Forms ASP.NET présente des informations à l'utilisateur dans n'importe quel navigateur ou appareil client.

ASP.NET Termes

- **SignalR**
- SignalR ASP.NET est une bibliothèque pour les développeurs ASP.NET qui simplifie le processus d'ajout de fonctionnalités de web en temps réel aux applications.
- Avant les Websockets HTML5, plusieurs techniques Javascript pouvaient être utilisées côté client pour simuler l'effet push/temps réel du serveur : le client envoie continuellement des requêtes au serveur pour voir s'il a de nouveaux messages qui lui sont destinés (utilisation du *XMLHttpRequest* et de fonctions Javascript tels que *setInterval* ou*setTimeout*). On se retrouvait souvent à maintenir un certain nombre de lignes de code écrites en Javascript

ASP.NET Termes

- **SignalR**
- SignalR est une bibliothèque client/serveur intégrée fournissant toute la plomberie nécessaire pour ajouter des fonctionnalités temps-réel à une application Web ASP.NET.
- Cette bibliothèque se base sur les Websockets. Quand ces derniers ne sont pas gérés par le navigateur du client, la librairie offre une solution de fallback en utilisant d'autres techniques sans avoir à changer le code de l'application côté client et serveur. SignalR va, en effet, masquer toute la complexité liée à la gestion des appels Javascripts au serveur. Il va, également, permettre l'appel de fonctions Javascript clients à partir du serveur.

ASP.NET Termes

- **Handler**
- C'est une classe qui permet d'envoyer et de traiter des Messages et des objets exécutables associés à un Thread « MessageQueue ». Chaque instance de Handler est associée à un seul Thread et sa MessageQueue.

ASP.NET Termes

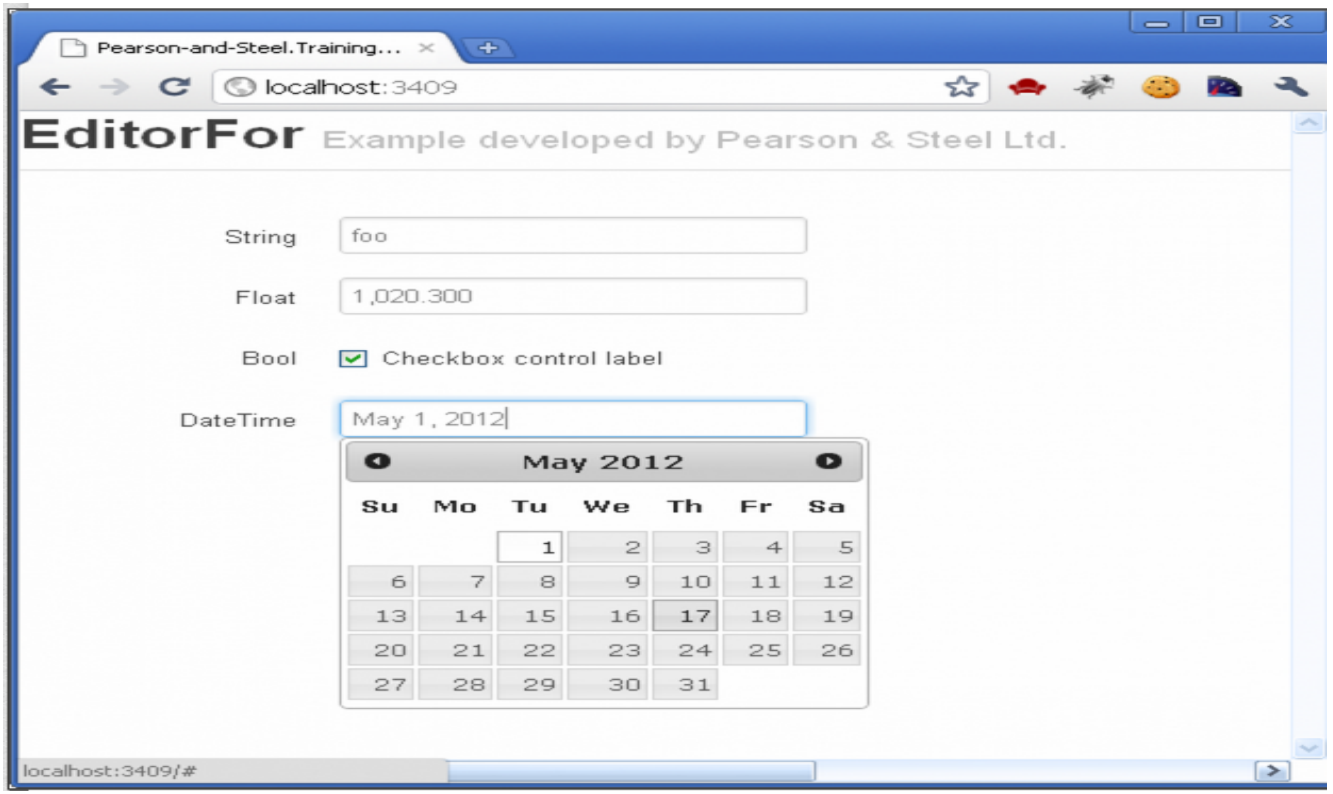
- **OWIN**
  - Il s'agit d'un ensemble de spécifications établies par Microsoft qui définissent comment une application web s'exécute sur un serveur web, et qui permettent notamment de s'abstraire de l'host qui exécutera l'application au final.
  - OWIN se base sur le développement de middleware qui sont chaînés entre eux et qui permettent d'ajouter des fonctionnalités à une application. On les utilise notamment pour mettre en place de l'authentification externe avec des fournisseurs d'identité tels que Facebook, Microsoft, Google ou encore Azure Active Directory. Ils peuvent aussi être de simples composants techniques qui permettent à une application d'effectuer un traitement particulier, par exemple logguer les requêtes http. Chaque middleware OWIN est responsable de sa fonctionnalité métier.

ASP.NET Termes

- **BSON**
  - BSON est un format d'échange de données informatiques utilisé principalement comme stockage de données et format de transfert de données par le réseau dans la base de données MongoDB. C'est un format binaire permettant de représenter des structures de données simples et des tableaux associatifs (appelées objets ou des documents dans MongoDB). Le nom "BSON" est basé sur le terme JSON et signifie « Binary JSON »
  - Il s'agit en quelques sortes de JSON compressé (75 à 80%)

ASP.NET Termes

- **EditorFor**
- Retourne un élément **input** HTML for chaque propriété de l'objet qui est représenté par une expression donnée.



ASP.NET et Windows Form

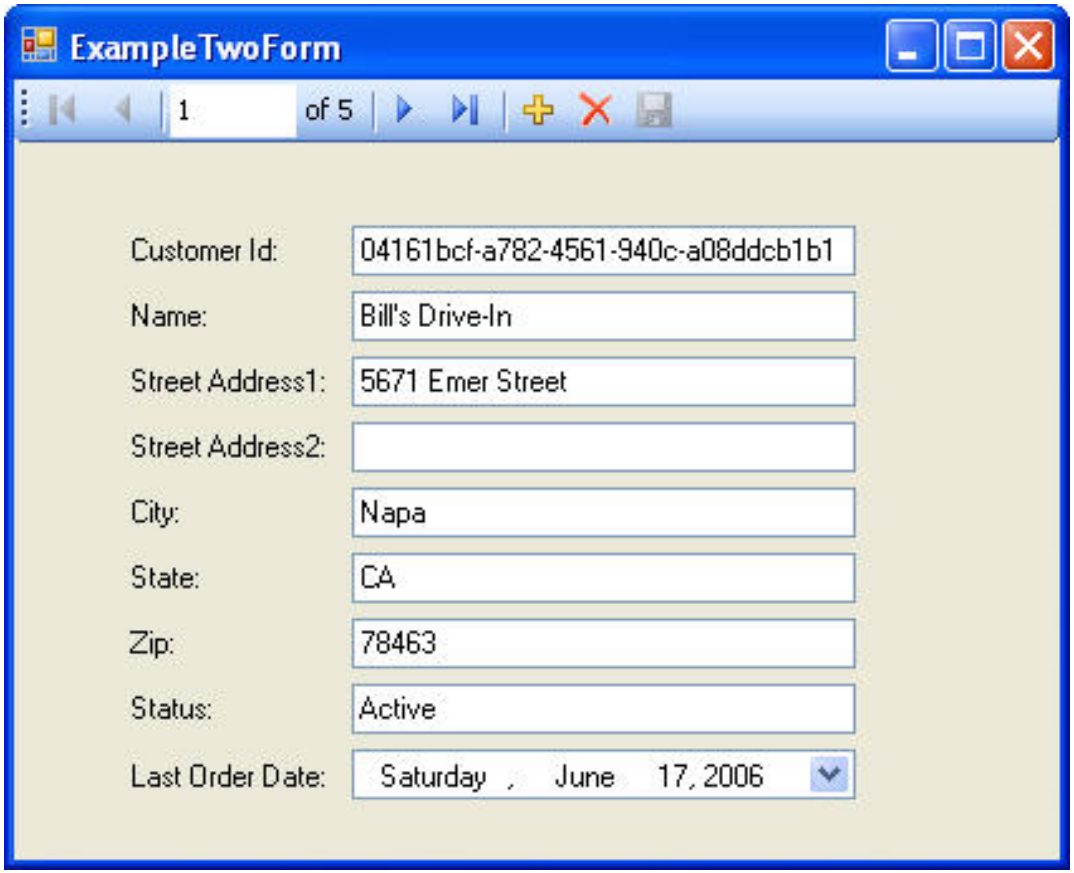
- **Windows Forms**
- Windows Forms est une infrastructure qui permet de construire des applications clientes Windows .NET, qui utilise la CLR. Les applications Windows .NET ne peuvent donc être déployées que sous un environnement où le Framework a été installé. Les applications Windows Form peuvent être créées dans n'importe quel langage .NET.

ASP.NET et Windows Form

- **Windows Form**
- Les Windows Forms présentent une multitude d'avantages
  - Windows Forms associe la simplicité de programmation (drag and drop de composant) et la puissance de la CLR.
  - Windows Forms tire profit des fonctionnalités de sécurité de la CLR pour avoir au final une application Windows sécurisée.
  - Windows Form prend complètement en charge la connexion rapide et aisée à des services Web XML.
  - Windows Form prend totalement en charge les contrôles ActiveX.

ASP.NET et Windows Form

- **Windows Form**



Portabilité de .NET

- **Standardisation**
- Microsoft a déposé les spécifications de la plateforme .NET à l'ECMA qui est un organisme de standardisation. Une partie des spécifications de la CLR et des classes de base du Framework ainsi que le langage C# sont donc disponibles sur le site de la ECMA ou sur le site de Microsoft.
- Ce n'est pas Microsoft qui va porter le Framework sur d'autres systèmes d'exploitation tels que Linux mais des organismes indépendants voire une communauté de développeurs (ex: le projet Mono).

Portabilité de .NET

- **Standardisation**
- Les spécifications soumises à la ECMA vont donc permettre aux développeurs de développer leur propre Framework voire leur propre compilateur C# sur leur architecture préférée.

Portabilité de .NET

- **CLS**
- CLS signifie Common Language Specification. La CLS va être en fait la spécification commune à tous les langages .NET qui va permettre la compatibilité entre tous les langages .NET.
- La totalité des langages .NET respectent cette spécification : par exemple le C++ Managé, c'est à dire le C++ .NET qui utilise les classes du Framework et qui est managé par la CLR, a dû être épuré de certains concepts tel que l'héritage multiple car les spécifications de la CLS ne supportent pas l'héritage multiple.

Portabilité de .NET

- **CLS**
- La CLS est donc le plus petit dénominateur commun entre tous les langages .NET et de par ses spécifications, la CLS exclu tous langages procéduraux, c'est-à-dire non objet, tel que le C.

.NET Server

- Ce qui change radicalement avec .NET Server comparé aux anciennes versions des serveurs Windows c'est la stratégie de sécurité.
- En effet, Microsoft s'est fait beaucoup critiquer pour la sécurité de ses serveurs, notamment à cause de ses failles de sécurité.
- Microsoft a répondu à ce défaut en fermant tout les services sur .NET Server : par défaut dorénavant, tout sera protégé. Avant, c'était exactement l'inverse : par défaut, tout était ouvert, et il fallait que les développeurs et administrateurs réseaux puissent protéger tous les accès.

.NET Server

- Maintenant, comme tout sera protégé par défaut, les applications seront beaucoup plus sécurisées : il faut autoriser les applications à avoir accès à certaines ressources.

.NET Server

- **Evolution**
- Windows 2000 Server → Windows 2003 Server (.NET) → Windows Server 2008 → Windows Server R2 → Windows Server 2012 (Windows Server 8) → Windows Server 2016

- Versions de Windows Server 2016 :
  - Essentials
  - Standard
  - Datacenter

.NET Server

Windows Server 2016 edition	Ideal for	Licensing model	CAL requirements*	Pricing Open NL ERP (US$)
Datacenter**	Highly virtualized and software-defined datacenter environments	Core-based	Windows Server CAL	$6,155
Standard**	Low density or non-virtualized environments	Core-based	Windows Server CAL	$882
Essentials	Small businesses with up to 25 users and 50 devices	Processor-based	No CAL required	$501

.NET Server

- **Windows Server 2016**
- Interface Graphique : Metro (comme Windows 8)
- Version Internet Information Service (IIS) : 10
- Version SQL Server : 2014

Langages de développement

Visual Studio

Introduction

- Les langages supportés par .Net sont nombreux. Au départ, Visual Studio.Net est livré avec les langages suivants : C#, VB.Net, Managed C++(ou C++.Net ou encore C++ Managé) ainsi que JScript.Net. Le Managed C++ est une variante du C++ qui garde la même syntaxe mais qui utilise les classes et qui est exécuté par la CLR. On peut toutefois programmer des applications Win32 standards avec Visual C++ (on parle alors de C++ unmanaged).

Introduction

- Ses langages respectent tous la CLS. Aujourd'hui Microsoft fournit déjà deux nouveaux langages, le J# et le F#, et d'autres sociétés ont implémenté des langages alternatifs (Perl, Python, Cobol, Eiffel et même Delphi).
- Chaque langage possède son propre compilateur. Par exemple, pour compiler un programme en C#, il faut utiliser csc (C Sharp Compiler) :

```
csc hello.cs
```

- Pour VB.Net : vbc (**V**isual **B**asic **C**ompiler) et ainsi de suite.

Les outils de développement

- **Ligne de commande**
  - Placer dans le PATH les répertoires suivants
    - Microsoft.NET\Framework (contient les principaux exécutables du Framework)
    - Microsoft Visual Studio .NET\FrameworkSDK\Bin (contient le SDK du Framework)
  - Il est également possible d'utiliser le fichier de commande fourni avec Visual Studio .NET (External Tools)

Les outils de développement

- **Ligne de commande**

```
// HelloMaster.cs

public class HelloMaster
{
      public static void Main()
      {
            System.Console.WriteLine("Hello Master !");
      }
}
```

  - Exécuter « csc hellomaster.cs »
  - Ajouter « System.Console.Readline() ; » pour attendre la saisie d'une chaîne de caractères avant de fermer la fenêtre.

Les outils de développement

- **Ligne de commande**

```
'Hellomaster.vb

Module HelloMaster

    Sub Main()
        System.Console.WriteLine("Hello Master")
        System.Console.ReadLine()
    End Sub

End Module
```

  - Equivalent en VB.NET
  - Exécuter « vbc hellomaster.vb »

Les outils de développement

- **Principales options de compilation**

Option	Equivalent VB.Net	Explications
/out:*filename*	/out:*filename*	Indique le nom du fichier de sortie. Si ce paramètre n'est pas spécifié, le nom par défaut sera celui de la classe contenant le Main ou celui du premier fichier.
/target:*type* /t:*type*	/target:*type* /t:*type*	Spécifie le type de fichier de sortie : • "exe" pour une application console • "winexe" pour une application Windows • "library" pour une bibliothèque de classe • "module" pour un module
/recurse:*wildcard*	/recurse:*wildcard*	Compile tous les fichiers situés dans le répertoire courant et dans les sous répertoires dont le nom correspond à *wildcard* (*.cs, *.vb, etc…). Très utile pour batcher une compilation (cf. les exemples de compilation pour un exemple concret).
/reference:*file_list* /r:*file_list*	/reference:*file_list* /r:*file_list*	Indique les fichiers des assemblées à référencer. Cela peut être une assemblée du Framework (ex : "/r:System.Xml.dll") ou une assemblée tierce (ex : "/r:MyLibrary.dll").
/doc:*file*	Pas d'équivalence pour le moment, sans doute cette option sera livrée lors d'un service pack.	Précise le fichier de documentation au format XML à générer.
/optimize:[+/-]	/optimize:[+/-]	Active (+) ou désactive (-) les optimisations. Remarque, "/optimize" seul est l'équivalent de "/optimize:+"
/debug:[+/-]	/debug:[+/-]	Active (+) ou désactive (-) l'emission d'information de débogage. Remarque, "/debug" seul est l'équivalent de "/debug:+"
@*file*	@*file*	Spécifie un fichier de réponse. Ce fichier peut contenir diverses options de compilation.

Les outils de développement

- **Principales options de compilation**
  - Pour voir les autres options, taper (suivant que l'on soit en VB. NET ou C#)

```
csc /?
vbc /?
```

Microsoft Visual Studio

- Visual Studio 2015 Free Offering Lineup

- Visual Studio Community is a full-featured IDE that is free for students, open source contributors, and small teams.
- All free offerings can be downloaded from http://www.visualstudio.com/

atidan

Microsoft Visual Studio

- **Visual Studio 2015**

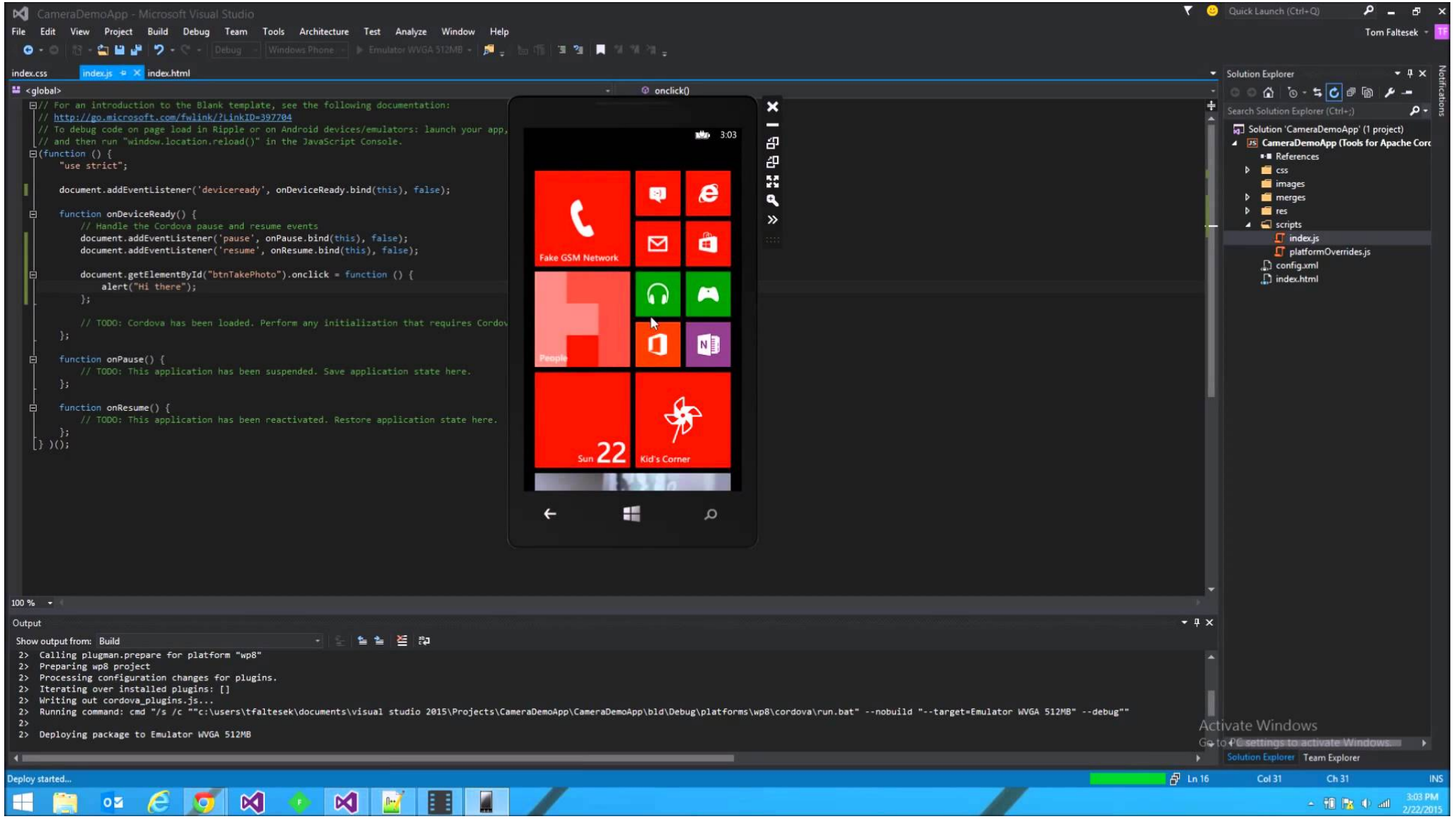
IF YOU HAVE	YOU WILL GET	NEW CUSTOMER PRICE
Visual Studio Community	Visual Studio Community 2015	**FREE**
Visual Studio Professional with MSDN	Visual Studio Professional 2015 Your Visual Studio Professional with MSDN subscription will continue without change	**$1,199** [3]
Visual Studio Premium with MSDN	**Visual Studio Enterprise 2015** Your subscription will be upgraded to Visual Studio Enterprise with MSDN [1]	**$5,999** [2]
Visual Studio Ultimate with MSDN		

Microsoft Visual Studio

MSDN subscriptions Benefit Matrix

BENEFIT	MSDN SUBSCRIPTION LEVEL			
	Enterprise	MSDN Platforms	Test Professional	Professional
Software and services for production use				
Team Foundation Server access level	Advanced	Advanced	Advanced	Basic
Visual Studio Online access level	Advanced	Advanced	Advanced	Basic
Microsoft Office Professional Plus 2013	Yes	No	No	No
Software and services for development and testing				
Windows, Windows Server, SQL Server	Yes	Yes	Yes	Yes
All other servers (Exchange, SharePoint, Dynamics, etc.)	Yes	Yes	No	No
Office	Yes	No	No	No
Azure	$150/mo.	$100/mo.	$50/mo.	$50/mo.
Additional benefits				
Pluralsight training	30 courses for 12 months	30 courses for 12 months (*upgraded*)	10 courses for 3 months	10 courses for 3 months
Technical support incidents	4	2	2	2
Office 365 Developer subscription	Yes	No	No	No
Universal Store developer account	Yes	No	Yes	Yes
Priority support in MSDN Forums	Yes	Yes	Yes	Yes
Microsoft e-learning course collections	2	2	1	1
MSDN Magazine	Yes	Yes	Yes	Yes
MSDN Online Chat	Yes	Yes	Yes	Yes
Special Offers from Visual Studio Partners	Yes	Yes	Yes	Yes

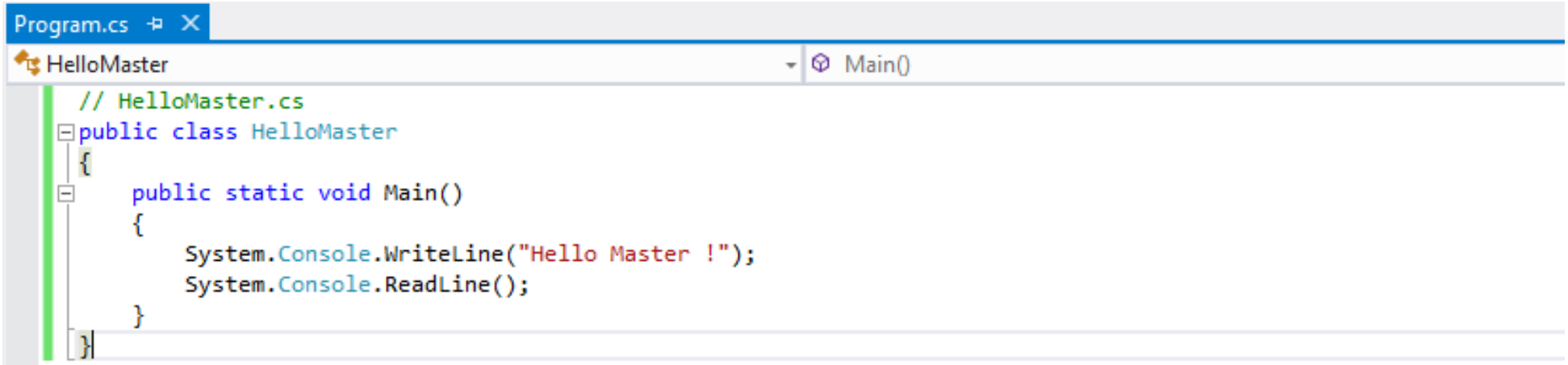
Microsoft Visual Studio



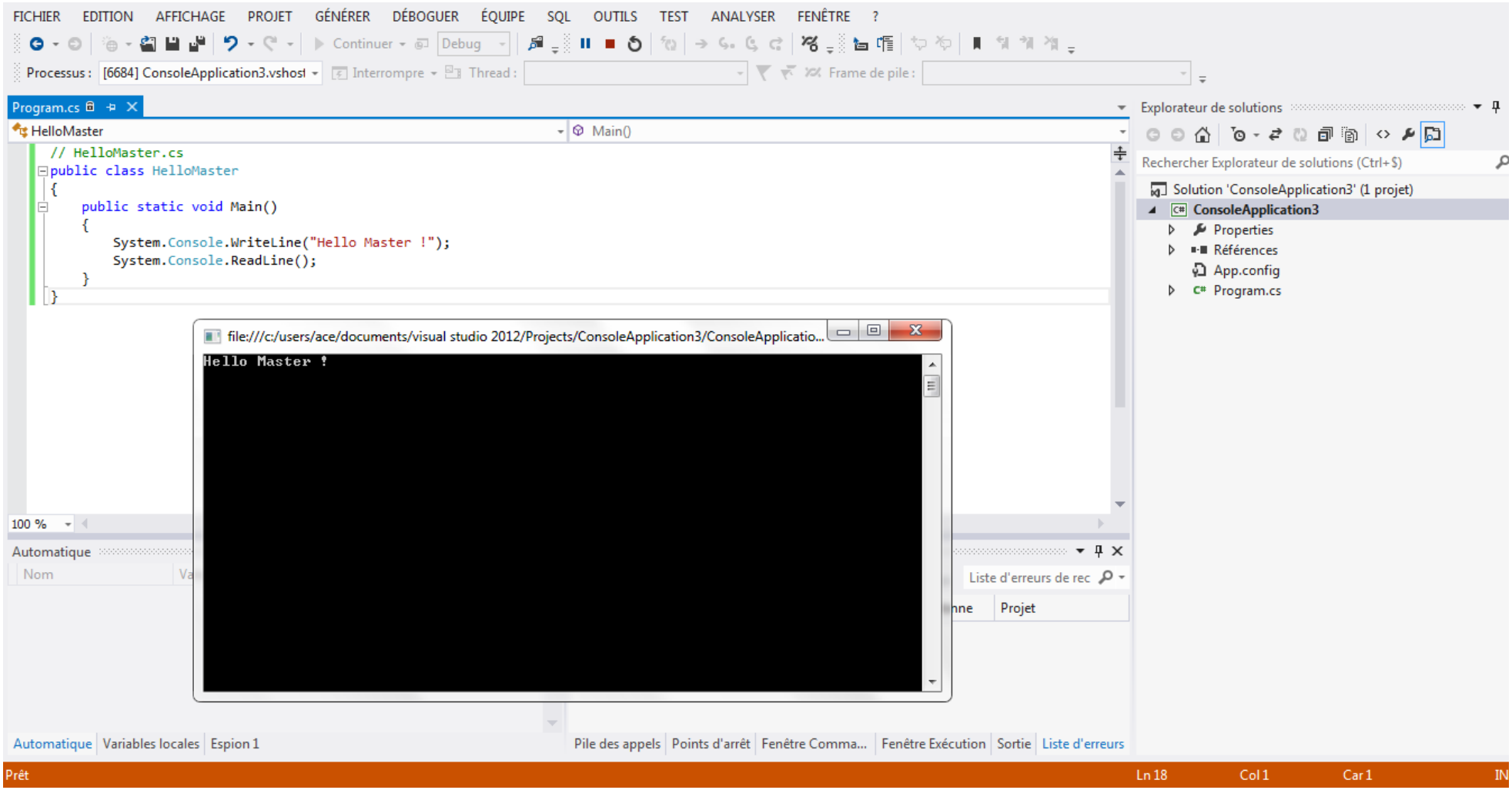
Microsoft Visual Studio

- **Visual Studio 2015**
  - Nouveau Projet → Nouvelle solution
  - Quelle est la différence entre projet et solution?

Microsoft Visual Studio



Microsoft Visual Studio



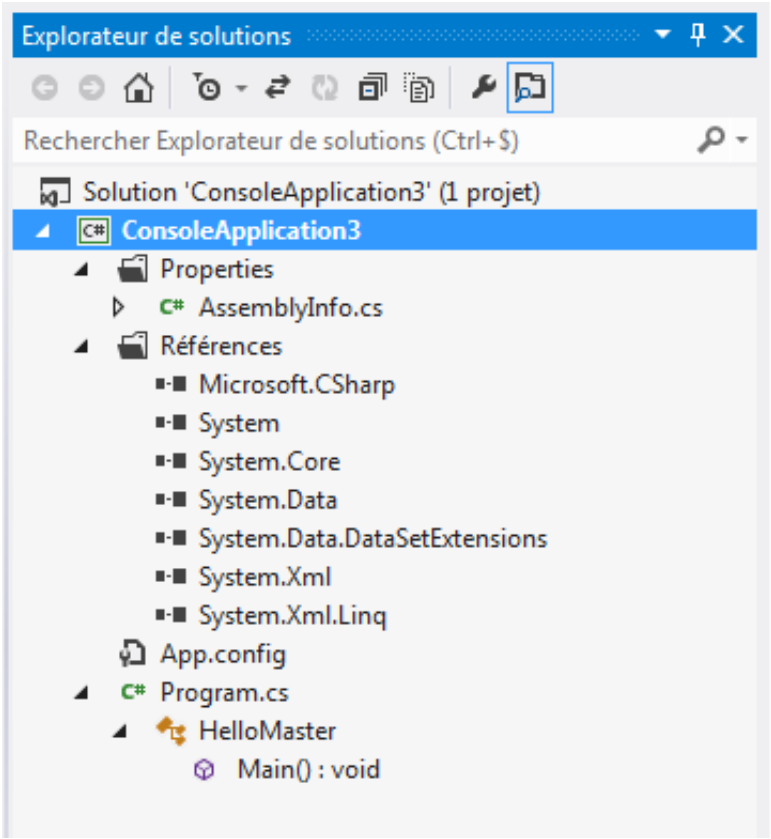
Microsoft Visual Studio

- La fenêtre se décompose en plusieurs parties :
  - La fenêtre principale : pour l'édition du code
  - L'explorateur de solution : liste des fichiers par projet pour la solution. Il faut remarquer le dossier 'References' qui ne contient pas des fichiers mais des références vers d'autres assemblies. Ainsi, si on utilise une assembly 'MyAssembly' contenue dans le fichier 'myassembly.dll', c'est ici qu'il faudra le référencer (bouton droit puis clic sur 'Add reference')

Microsoft Visual Studio

- La grille de propriété : affiche les propriétés de l'élément sélectionné (un fichier, un contrôle, n'importe quoi de sélectionnable), dans notre cas on a les propriétés du fichier class1.cs.
- Remarquez les onglets 'Class View' (pour l'explorateur de classe) et 'Dynamic Help' (pour l'aide dynamique) qui peuvent à tout moment être mis en avant par un clic ou déplacé par un cliqué déplacer

Microsoft Visual Studio



Microsoft Visual Studio

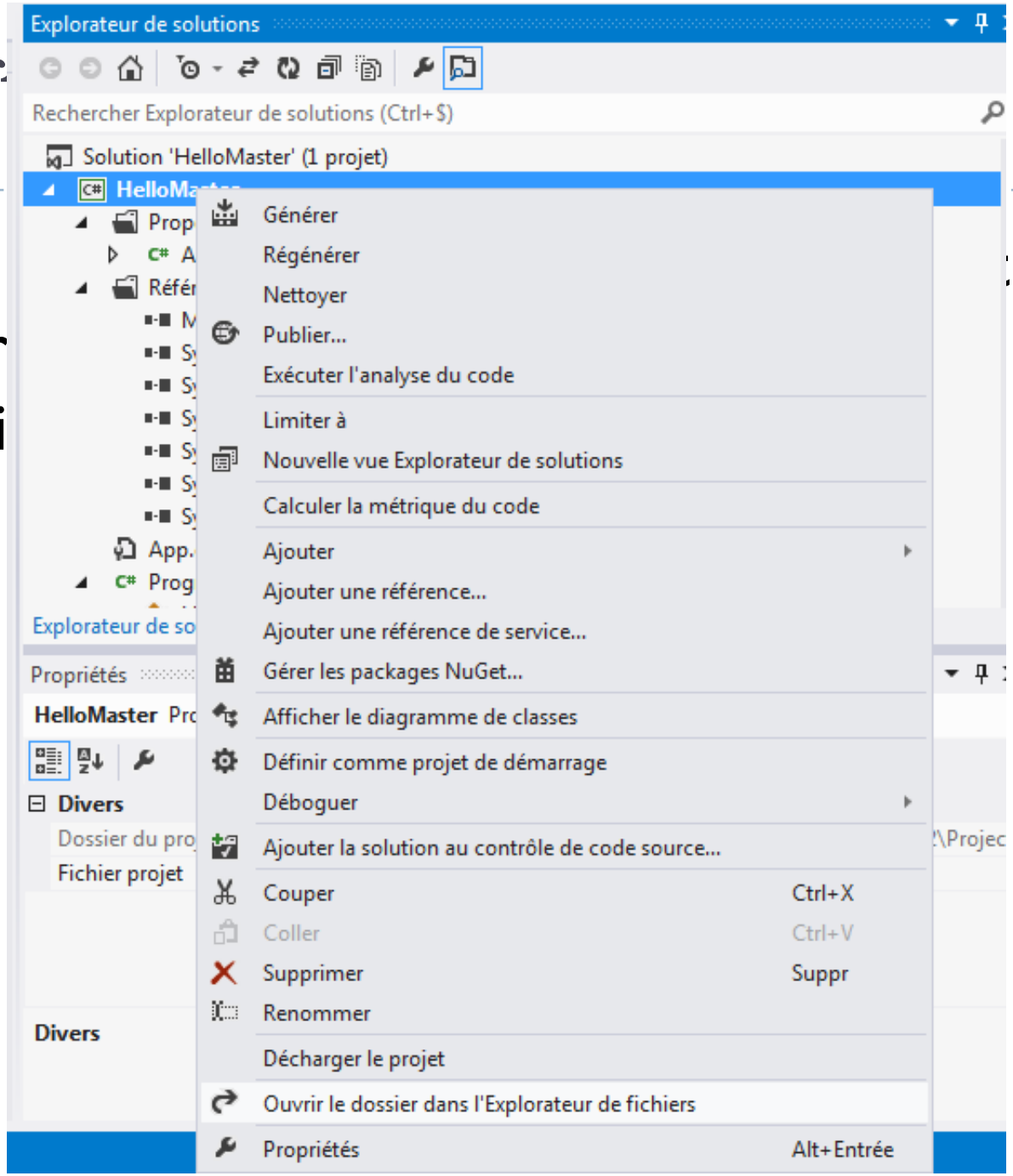
- Quels sont les fichiers qui ont été créés?
  - Program.cs : ce fichier contient la définition de la classe HelloMaster et qui est la classe contenant le point d'entrée de notre programme).
  - AssemblyInfo.cs: ce fichier ne contient que des déclarations relatives à la description de l'assembly. Il est composé d'un ensemble d'attributs définissant par exemple, l'auteur de la classe (ou société), la version, la description, etc…

Microsoft Visual Studio

- Quels sont les fichiers qui ont été créés?
  - App.config: Configuration de l'application (ex. Framework à utiliser…)
  - Un projet (HelloMaster.csproj): contient la définition de notre projet, les options de compilation, les fichiers associés aux projet, etc…
  - Une solution (HelloMaster.sln): décrit la solution (liste des projets et des fichiers)

Microsof

- Ce que voir via l'explor 'Ouvrir le dossi

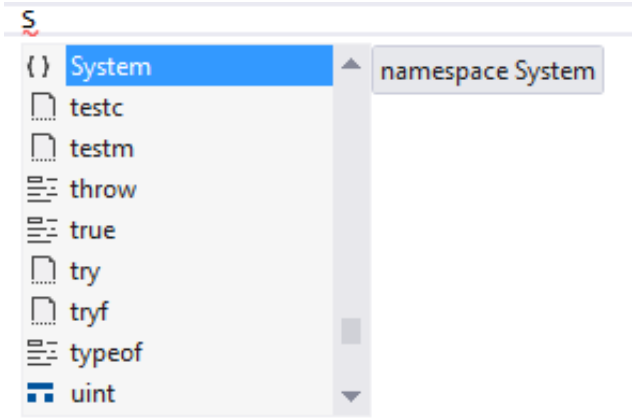


Microsoft Visual Studio

- un répertoire bin: contient les exécutables générés et les fichiers d'information de debug (.pdb)
- un répertoire obj: contient les éléments intermédiaires de compilation
- un fichier 'HelloMaster.csproj.user' et 'HelloMaster.suo' qui sont des sous fichiers de respectivement 'HelloMaster.csproj' et 'HelloMaster.sln'. Ces fichiers sont générés par Visual Studio et il vaut mieux ne pas y toucher à moins de savoir ce que l'on fait.

Microsoft Visual Studio

- Taper « s » et le menu de l'Intellisense fait apparaître



Microsoft Visual Studio

- Il est possible de remarquer qu'à chaque élément du menu est associé une icône.

Icône	Signification
	Espace de nom
	Structure publique
	Structure protégée
	Structure privée
	Classe publique
	Classe protégée
	Classe privée
	Interface
 (rose)	Méthode ou fonction publique

Microsoft Visual Studio

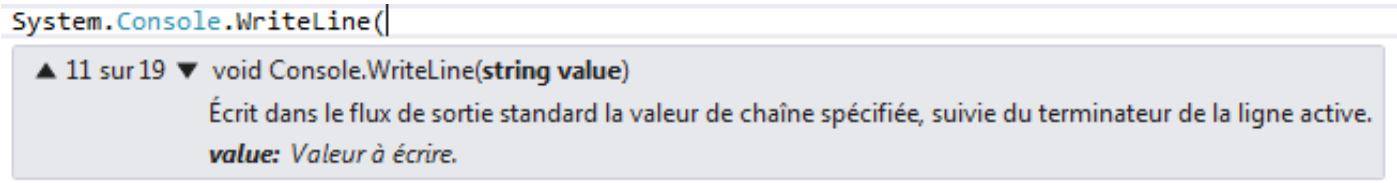
 (rose)	Méthode protégée
 (rose)	Méthode privée
 (cyan)	Variable publique
 (cyan)	Variable protégée
 (cyan)	Variable privée
	Propriété publique
	Propriété protégée
	Propriété privée
	Evénement public
	Evénement protégé
	Evénement privé
	Délégation publique
	Délégation protégée

Microsoft Visual Studio

	Délégation privée
	Enumération publique
	Enumération protégée
	Enumération privée
	Constante publique
	Constante protégée
	Constante privée
	Assemblée

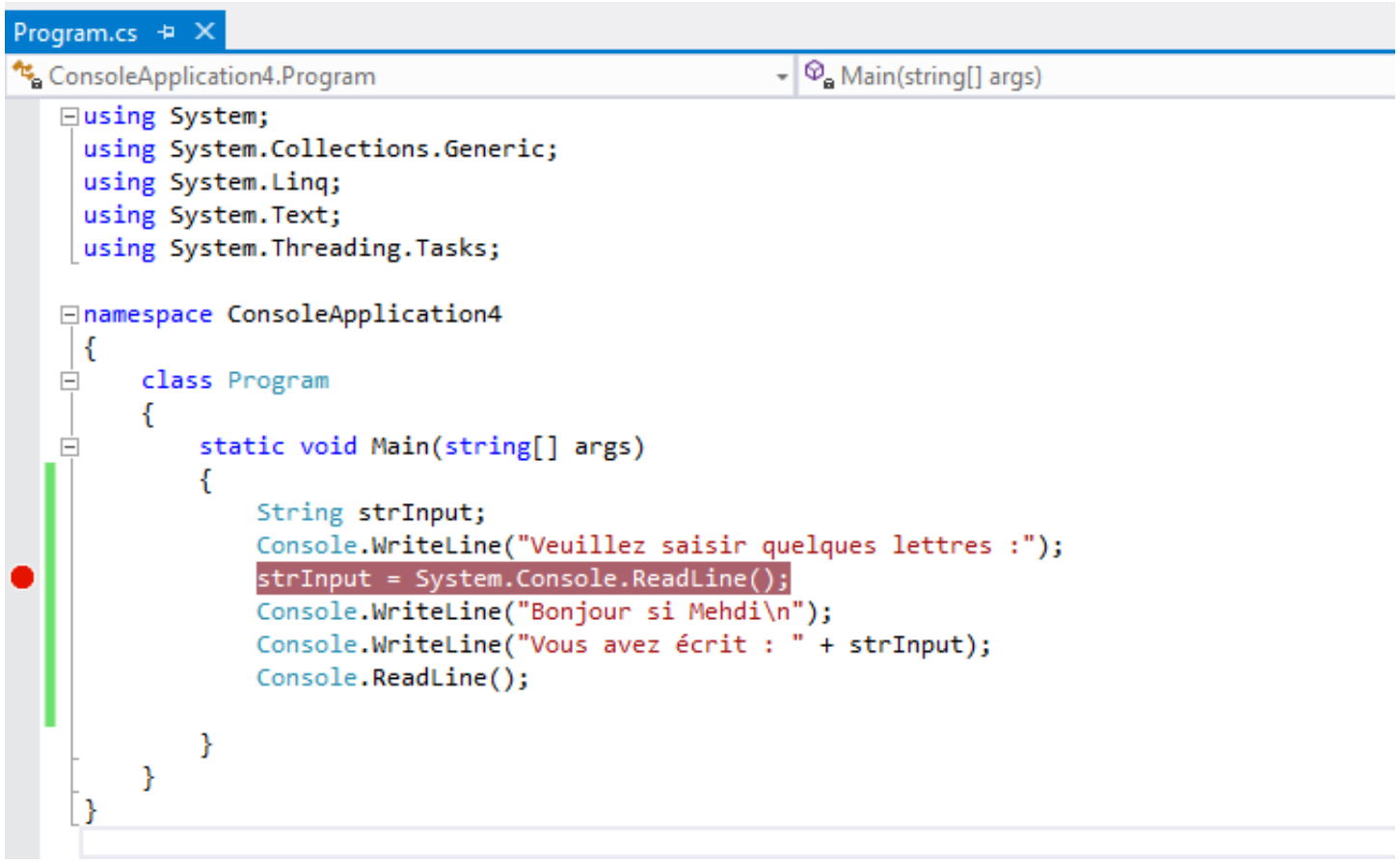
Microsoft Visual Studio

- Surcharge



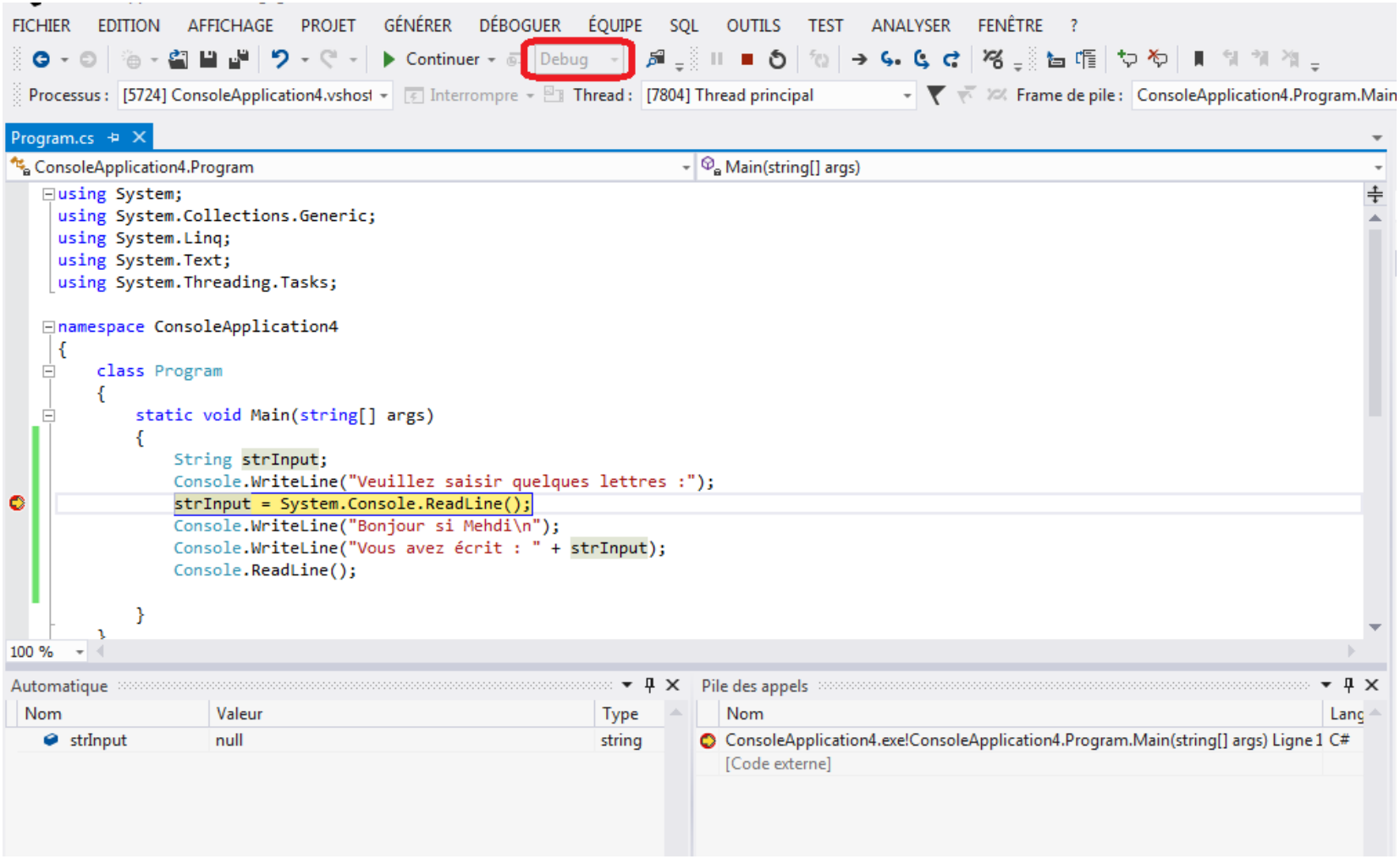
Microsoft Visual Studio

- Débugage



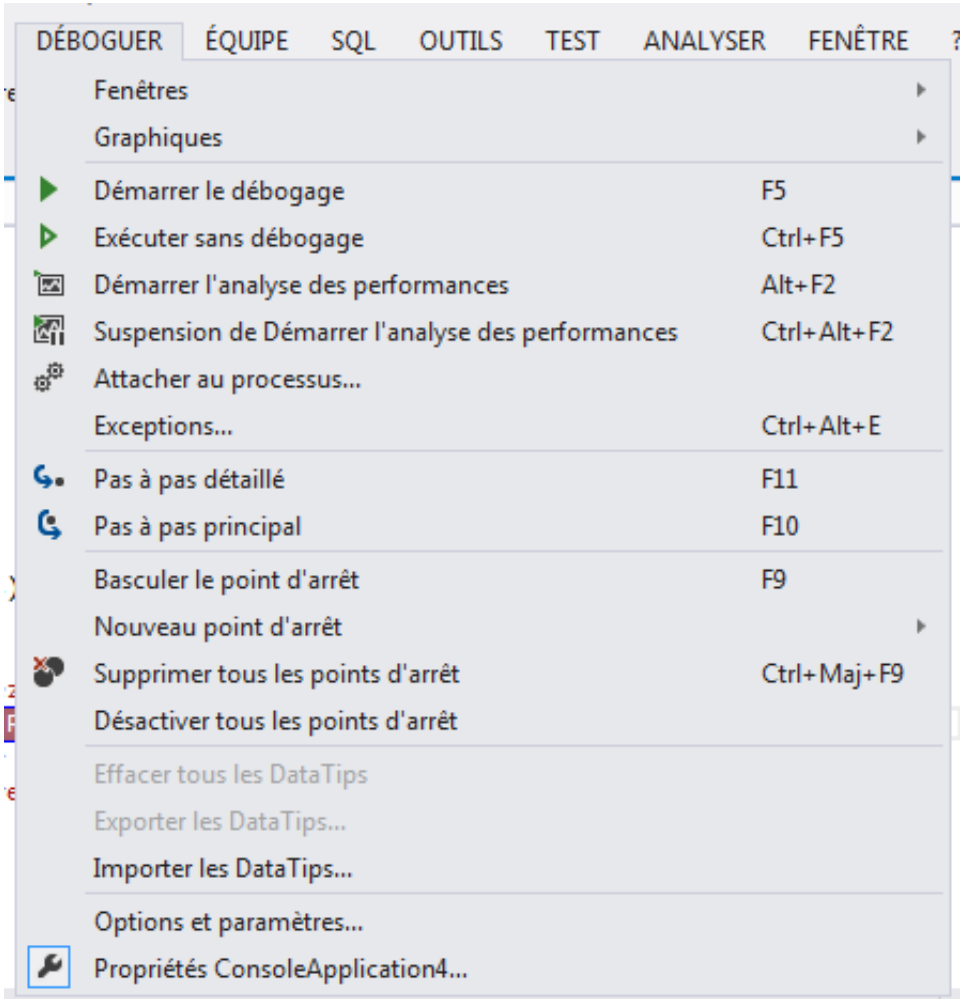
Microsoft Visual Studio

- Débugage



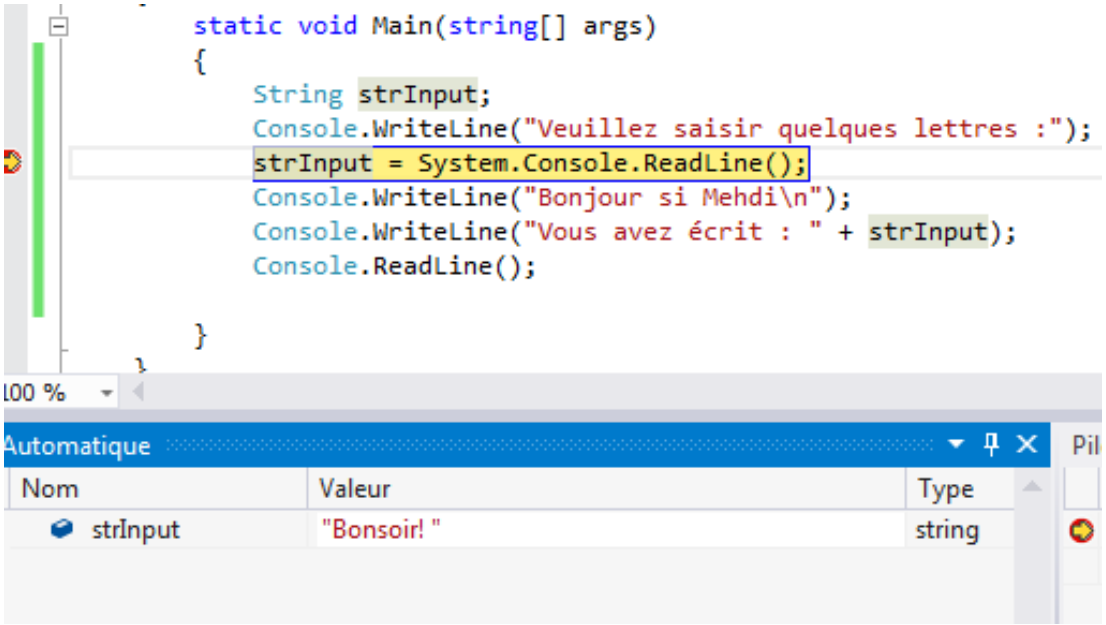
Microsoft Visual Studio

- Débugage



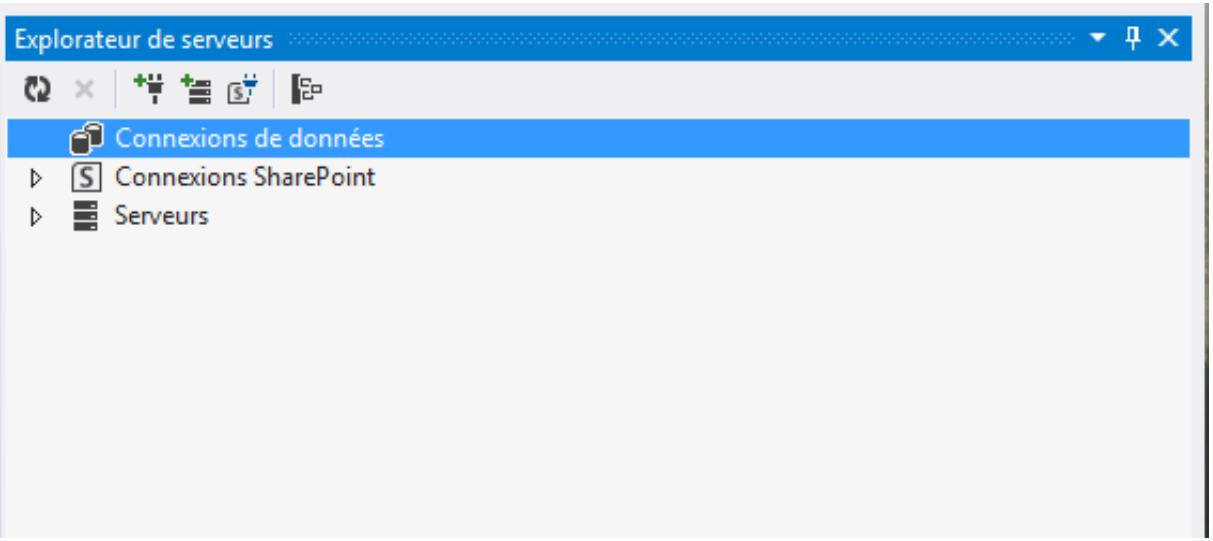
Microsoft Visual Studio

- Débugage



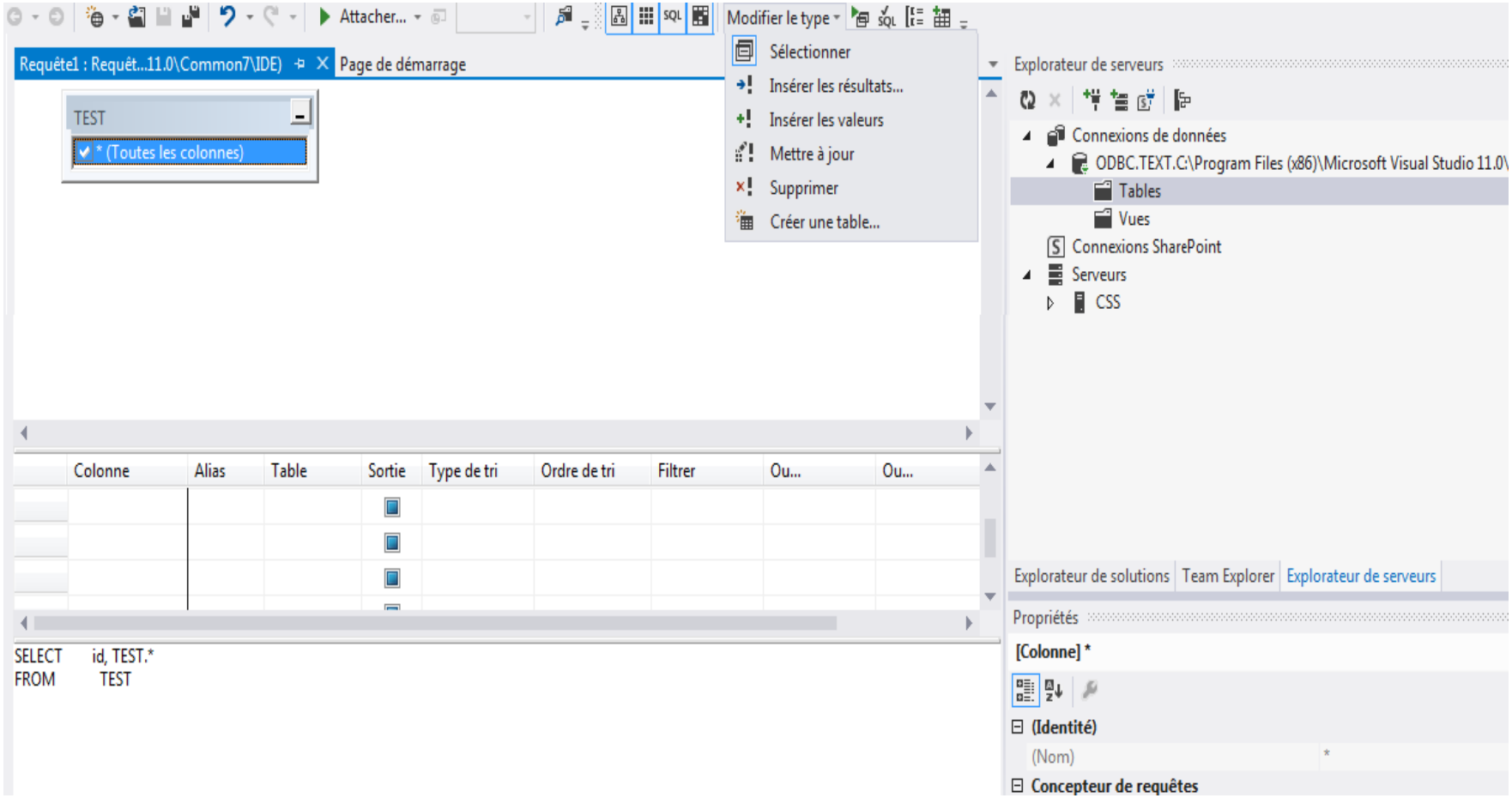
Microsoft Visual Studio

- Fonctionnalités avancées
  - Le design des bases de données



Microsoft Visual Studio

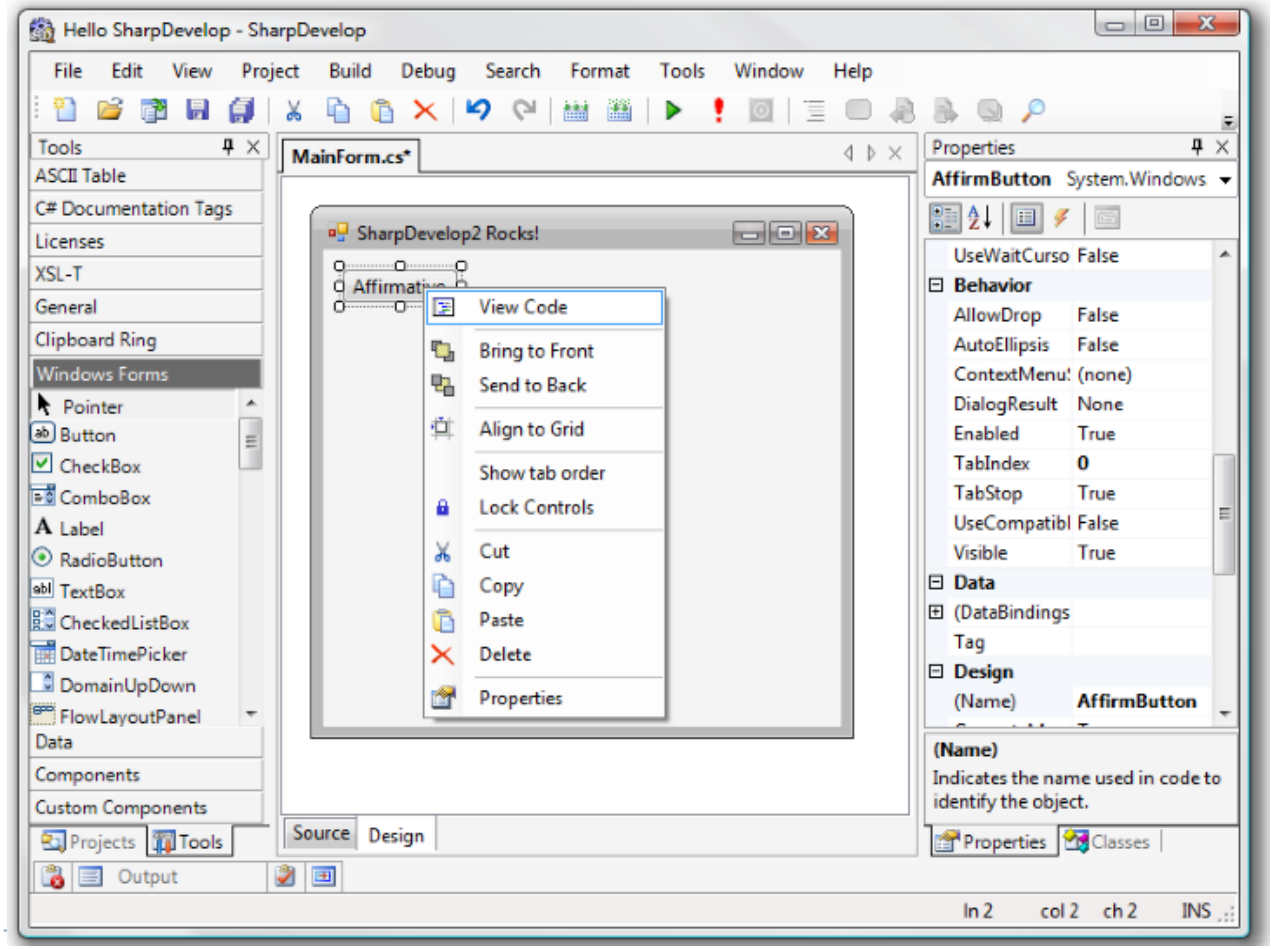
- Fonctionnalités avancées
  - Le design des bases de données



Microsoft Visual Studio

- Alternatives

http://www.icsharpcode.net/OpenSource/SD/Default.aspx



Le langage C#

Le langage C#

- **HelloMaster**
  - Ecriture de 3 manières différentes du programme HelloMaster
  - Attention, Un programme en .Net exécutable (donc incluant les applications consoles et les applications Windows) nécessite un point d'entrée pour être exécuté. La CLR cherche alors parmi toutes les classes de l'assembly une fonction Main qui soit statique. Il faut donc n'avoir qu'une et une seule fonction Main statique dans le programme.

Le langage C#

- **HelloMaster – Exemple 1**

```
// Hello1.cs

public class Hello1
{
      public static void Main()
      {
            System.Console.WriteLine("Hello, Master !");
      }
}
```

- **Explications :**
- La méthode Main doit être incluse dans une classe. En effet, tout programme en .Net est une classe. Le Main étant le point d'entrée pour les exécutables.
- La méthode Main accepte **UN SEUL PARAMETRE : STRING []**

Le langage C#

- **HelloMaster – Exemple 1**

```
// Hello1.cs

public class Hello1
{
      public static void Main()
      {
            System.Console.WriteLine("Hello, Master !");
      }
}
```

- **Explications :**
- Ici, on fait appel à la méthode WriteLine(string value) de la classe System.Console (classe Console du Namespace System). Cette méthode étant statique (static), il n'y pas besoin d'instancier un objet de la classe Console.

Le langage C#

- **HelloMaster – Exemple 2**

```
// Hello2.cs

using System;

public class Hello2
{
      public static void Main()
      {
            string msg="Hello, Master!";
            Console.WriteLine(msg);
      }
}
```

- **Même sortie qu'Exemple 1. Différences?**

Le langage C#

- **HelloMaster – Exemple 2 – Explications**
  - Le mot clef **using**, suivi d'un namespace, permet d'indiquer au compilateur où chercher les classes. Dans cet exemple, comme la classe Console est dan le namespace System, avoir importé System permet d'éviter d'avoir à saisir le chemin complet de la classe.
  - Attention toutefois à ne pas utiliser des namespaces ayant des noms de classes communs.
  - Il faut utiliser ses propres namespaces qu'il est également possible d'importer de cette manière.

Le langage C#

- **HelloMaster – Exemple 2 – Explications**
  - Le texte affiché n'est plus directement mis dans les arguments de la méthode WriteLine, mais est affecté à une variable de type string msg que nous déclarons grâce à type var = value.
  - Il faut obligatoirement déclarer les variables en C# et on peut directement affecter une valeur lors de la déclaration. On passe alors la variable msg qui contient le texte à la méthode WriteLine.

Le langage C#

- **HelloMaster – Exemple 2 – Explications**
  - Remarque : en VB.Net, la déclaration des variables n'est pas obligatoire. Une déclaration permet de définir si oui ou non elle le devienne : Option Explicit On/Off.
  - Par défaut elle est à On, et il n'est pas conseillé de la fixer à Off car cela apporte une sécurité supplémentaire dans la qualité du codage.

Le langage C#

- **HelloMaster – Exemple 3**

```
// Hello3.cs
// arguments: A B C D

using System;

public class Hello3
{
   public static void Main(string[] args)
   {
      Console.WriteLine("Hello, Master!");
      Console.WriteLine(
            "You entered the following {0} command line arguments:",
            args.Length );
      for (int i=0; i < args.Length; i++)
      {
         Console.WriteLine("Arg[{0}]={1}", i, args[i]);
      }
   }
}
```

Le langage C#

- **HelloMaster – Exemple 3 – Explications**
- Compiler, puis exécuter le programme en lui donnant des arguments de ligne de commande :

```
Hello3.exe A B C D
```

Sortie :

```
Hello, Master!
You entered the following 4 command line arguments:
Arg[0] = A
Arg[1] = B
Arg[2] = C
Arg[3] = D
```

- **HelloMaster – Exemple 3 – Explications**
  - La méthode Main possède un argument : string[] args. Cet argument est affecté par le système d'exploitation. Il est de type string[], ce qui veut dire qu'il s'agit là d'un tableau de string. On accède à son contenu par : string[index] où index est l'index de l'élément (commence à 0).

- **HelloMaster – Exemple 3 – Explications**
  - La deuxième ligne de Main est un appel à une autre surcharge de la méthode WriteLine, elle aussi déclarée en statique (static). Cette surcharge demande en entrée une chaîne ("You entered the following {0} command line arguments :") pour le format et un objet pour l'argument de formatage.
  - En effet, le programme va parcourir la chaîne de format pour y trouver tous les {n} pour y insérer le $(n-1)^{ème}$ argument. Ici, un seul argument est fourni : args.Length qui va remplacer le {0} de la chaîne de format.

- **HelloMaster – Exemple 3 – Explications**
  - Length est une propriété de args. Une propriété stocke un objet, mais à la différence d'une variable de classe publique, on contrôle son comportement en définissant une méthode get et/ou une méthode set pour chaque propriété.

- **HelloMaster – Exemple 3 – Explications**
  - On récupère le nombre d'élément du tableau args grâce à sa propriété Length. Bien que l'on soit dans le cas d'un tableau, on peut appeler des méthodes, des propriétés, des champs, etc. d'un tableau ou tout autre type de base, grâce à la technique du Boxing et du UnBoxing, qui consiste à mettre tout type de base dans un objet.
  - Ainsi, le type System.Array associé à ce tableau possède la propriété Length qui, justement, retourne le nombre d'éléments de ce dernier.

- **HelloMaster – Exemple 3 – Explications**
  - Le dernier bloc de programme est une boucle for. La boucle for prend trois expressions en entrée :
    - int i=0 : initialisation. On initialise la boucle for ici. Ces instructions ne seront exécutés qu'une seule fois.
    - i < args.Length : condition d'arrêt. A chaque itération, on évalue cette expression. Si le résultat est vrai (true), on exécute l'itération. Si il est faux (false), on quitte la boucle for.
    - i++ : instruction d'itération. Après chaque itération, on exécute cette expression. Ici, on incrémente i.

- **HelloMaster – Exemple 3 – Explications**
  - L'algorithme de ce for est donc :
    - Création et initialisation d'une variable de compteur à 0
    - Si ce compteur est inférieur au nombre d'élément du tableau args, on exécute une itération de la boucle.
    - L'itération consiste à afficher le contenu du tableau à l'index spécifié par le compteur.
    - On incrémente le compteur à la fin de l'itération
    - On recommence, tant que la condition d'arrêt est vraie.

Le langage C#

- **HelloMaster – Exemple 4**

```
// Hello4.cs
// arguments: La vie est un long fleuve tranquille

using System;

public class CommandLine2
{
   public static void Main(string[] args)
   {
       Console.WriteLine("Hello, Master!");
       Console.WriteLine("Number of command line parameters = {0}"
                         ,args.Length);
       foreach(string s in args)
       {
          Console.WriteLine(s);
       }
   }
}
```

- Compiler puis exécuter

```
hello4.exe La vie est un long fleuve tranquille
```

Le langage C#

- **HelloMaster – Exemple 4**
- Sortie

```
Hello, Master !
Number of command line parameters =7
La
vie
est
un
long
fleuve
tranquille
```

- **HelloMaster – Exemple 4 – Explications**
  - La différence avec le programme précédent est l'emploi du foreach à la place du for.
  - La syntaxe du foreach est la suivante : foreach (type var in container). Ainsi, on laisse le soin au framework de gérer les index à notre place. Plutôt que de parcourir manuellement les index du conteneur, on lui indique simplement un variable dans laquelle mettre tous les éléments du conteneur pour le traiter.
  - Ainsi dans cet exemple : Pour toutes les chaînes de caractère contenues dans args, la copier dans une chaîne s et traiter s (l'afficher).

Le langage C#

- **Structures conditionnelles**

Le langage C#

```
// Condition.cs

using System;

public class Condition
{
	public static void Main()
	{
		int i = 5;
		if (i==5)
		{
			Console.WriteLine("{0}=5",i);
		}
		else
		{
			Console.WriteLine("{0}!=5",i);
		}
		i=42;
		if (i==5)
		{
			Console.WriteLine("{0}=5",i);
		}
		else
		{
			Console.WriteLine("{0}!=5",i);
		}

		string str = "Koala";
		if (str.StartsWith("K"))
		{
			Console.WriteLine(str + " starts with \"K\"");
		}
		else if (str.Length>3)
		{
			Console.WriteLine(str +
	" does not starts with \"K\" but its length is greater than 3"
			);
		}
		str = "Elephant";
		if (str.StartsWith("K"))
		{
			Console.WriteLine(str +" starts with \"K\"");
		}
		else if (str.Length>3)
		{
			Console.WriteLine(str +
	" does not starts with \"K\" but its length is greater than 3"
			);
		}
```

Le langage C#

```
            string msg;

            str = "Cat";
            msg=((str=="Cat")? "Cat" : "not Cat");
            Console.WriteLine("str is : \"{0}\" " , msg );

            str = "Dog";
            msg=((str=="Cat")? "Cat" : "not Cat");
            Console.WriteLine("str is : \"{0}\" " , msg );

            str = "Turtle";
            switch (str)
            {
                  case "Rabbit":
                        Console.WriteLine(
                            "str is a Rabbit and run fast !");
                        break;
                  case "Turtle":
                        Console.WriteLine(
                            "str is a Turtle and walk very slow !");
                        break;
                  case "Fly":
                        Console.WriteLine(
                            "str is a Fly and fly in the air");
                        break;
                  default:
                        Console.WriteLine("str is... unknown !!");
                        break;
            }

            Console.ReadLine();
      }

}
```

Le langage C#

- **Structures conditionnelles**
  - La sortie console

```
5=5
42!=5
Koala starts with "K"
Elephant does not starts with "K" but its length is greater than 3
str is : "Cat"
str is : "not Cat"
str is a Turtle and walk very slow !
```

Le langage C#

- **Structures conditionnelles – Explications**
  - if (expression) { instructions if true } [else { instructions if false }]. Si expression est évaluée true, c'est le premier bloc d'instruction qui est exécuté, sinon le deuxième.
  - if (expression) { instructions if true } [else if(condition) { instructions if true }[else if (condition) { instructions if true } [else …]]]. On peut enchaîner les tests autant que l'on souhaite. Si la succession de tests consiste à tester une valeur particulière, il vaut mieux passer par un while.

Le langage C#

- **Structures conditionnelles – Explications**
  - ((condition) ? expression if true: expression if false) : cette construction évalue la condition et retourne la première expression si la condition est vraie, sinon la seconde expression. Utile pour diminuer le nombre de lignes de code.
  - switch (expression) : cette structure est intéressant pour éviter les if à répétition. Elle évalue l'expression puis parcourt tous les « case value : » et exécute le code associé. S'il ne trouve rien, il exécute le contenu du « default : ».

Le langage C#

- **Boucles**

Le la

```
// iterations.cs

using System;
using System.Collections;

public class Test
{
	public static void Main()
	{
		// 1ère boucle: for
		int i;
		for (i=0 ; i<10 ; i++)
		{
			Console.WriteLine("i={0}",i);
		}

		// 2ème boucle: foreach pour un tableau
		int[] primes= {1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
		foreach (int prime in primes)
		{
			Console.WriteLine("prime={0}",prime);
		}

		// 3ème boucle: foreach pour une collection
		ArrayList myArrayList =new ArrayList();
		myArrayList.Add("Une chaîne de caractère");
		myArrayList.Add(5);
		myArrayList.Add(System.Math.PI);
		myArrayList.Add(0x11110010);
		foreach (object obj in myArrayList)
		{
			Console.WriteLine(obj);
		}

		// 4ème boucle: while
		int j=10;
		while(j>0)
		{
			Console.WriteLine("j={0}",j);
			j--;
		}

		// 5ème boucle: do while
		int k=2;
		do
		{
			Console.WriteLine("k={0}",k);
			k *= 2;
		} while (k != 8);

		Console.ReadLine();
	}
}
```

Le langage C#

- **Boucles**
- Sortie écran

```
i=0
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9
prime=1
prime=2
prime=3
prime=5
prime=7
prime=11
prime=13
prime=17
prime=19
prime=23
prime=29
Une chaîne de caractère
5
3,14159265358979
286326800
j=10
j=9
j=8
j=7
j=6
j=5
j=4
j=3
j=2
j=1
k=2
k=4
```

- **Boucles – Explications**
  - 1ère boucle : for
    - le for en c# se construit de la même façon que le for en C++.
    - En effet, il se divise en trois parties (chacune étant optionnelle) :
      - i=0 : initialisation. C'est une instruction qui sera exécutée une seule fois avant la première itération. Rien n'empêche d'appeler une fonction pour un grand nombre d'instructions, ou de déclarer une variable pour faire un compteur qui ne sera alors accessible que dans la boucle for.
      - i<10 : Condition d'arrêt. Cette expression est évaluée à chaque itération. Si l'expression est évaluée vrai (true) l'itération est exécutée. Si l'expression est évaluée fausse (false) le programme sort de la boucle for.

- i++ : fin d'itération. Cette instruction est exécutée APRES chaque itération, souvent utilisée pour incrémenter un compteur. Rien n'empêche d'appeler une fonction quelconque.

- Cet exemple a donc créé une variable i, puis l'a affichée et incrémentée une dizaine de fois.

- **Boucles – Explications**
  - 2ème boucle : foreach pour un tableau
    - foreach sert à parcourir tous les éléments d'un tableau. Ainsi, comme dans cet exemple, il faut spécifier une variable qui va successivement contenir toutes les valeurs du tableau. Ici, nous avons un tableau d'entier primes. Nous utilisons donc un entier prime qui va, tel un curseur, prendre les valeurs successives du tableau pour être utilisé dans la boucle.

- **Boucles – Explications**
  - 3ème boucle : foreach pour une collection
    - Avant de pouvoir expliquer le foreach sur une collection, il faut présenter ce qu'est une collection. Pour résumer, une collection est un conteneur à objet qui peut contenir soit tout type d'objet, soit un type d'objet.
    - Ainsi, dans notre exemple, nous avons créé une collection de type ArrayList (ArrayList est une classe, donc pour créer un objet nous avons utilisé le mot clé new) que nous avons rempli grâce à sa méthode Add de plusieurs objets, et de types divers pour l'exemple.
    - Ainsi, grâce au foreach, nous parcourons tous les éléments de la collection, en utilisant la variable obj de type object (tout en .Net est hérité de object. La variable obj prend alors successivement les valeurs de tous les éléments de la collection myArrayList pour être traité dans la boucle.

- **Boucles – Explications**
  - 4ème boucle : while
  - While exécute les instructions de la boucle tant que la condition spécifiée entre parenthèses est vrai. Ainsi, avant chaque itération, l'expression est évaluée. Si elle est vraie (true), la boucle est exécutée, sinon, le programme quitte la boucle while.
  - Dans notre cas, nous affichons un compteur puis nous le décrémentons à chaque itération. La boucle s'arrêtera dès que le compteur atteindra 0 (j>0 renvoie alors false)
  - Il faut veiller à ne pas provoquer de boucle infinie due à une évaluation fausse. Dans le cas contraire, le seul moyen de quitter la boucle est de générer une erreur ou de tuer le processus du programme.
  - Il est possible d'arrêter l'exécution de la boucle grâce à continue ou à break.: **continue** quitte l'itération en cours et passe à la prochaine (si bien sûr la condition est vraie) et **break** quitte l'itération en cours ET le while.

- **Boucles – Explications**
  - 5ème boucle : do while
  - Le principe du do while est identique à celui du while à une différence près : l'évaluation de l'expression d'arrêt est évaluée à la fin de chaque itération. Ainsi, on est sûr que le contenu de la boucle est exécuté au moins une fois dans le programme. On pourrait ainsi traduire le do while par : "Faire instructions puis répéter si condition".
  - Les mots clef continue et break s'appliquent également.

Le langage C#

- **Boucles**
- On remarque que chaque boucle pourrait servir pour la plupart des situations. Toutefois, certains cas se traitent avec plus de facilité avec un type de boucle qu'avec un autre. Par exemple, le parcours de collection est aisé avec foreach, alors qu'avec un while, il aura fallu gérer les index manuellement.

Le langage C#

- **Tableaux**

Le la

- **Tab**

```
// arrays.cs

using System;
class ArraySample
{
      public static void Main()
      {
            // Tableau à une dimension
            int[] numbers = new int[4] { 5, 10, 15, 20 };
            for(int i=0; i<numbers.Length ; i++)
            {
                  Console.WriteLine("Number {0} is {1}",i,numbers[i]);
            }

            // Tableau multidimensionnel (ici 2)
            string[,] names = new string[2,3] {
                  {"Jean", "Stephane", "Bernard"},
                  {"Sophie", "Sylvie", "Jeanne"}};
            for(int i=0; i< names.GetLength(0); i++)
            {
                  // On affiche sous forme de colonnes la ligne
                  for(int j=0; j< names.GetLength(1); j++)
                  {
                        Console.Write("{0}\t",names[i,j]);
                  }

                  // Après chaque ligne on saute une ligne
                  Console.Write("\n");
            }

            // Tableau de tableau
            byte[][] scores = new byte[5][];
            for (int i = 0; i < scores.Length; i++)
            {
                  scores[i] = new byte[i+3];
            }

            // Affiche la longueur de chaque ligne
            for (int i = 0; i < scores.Length; i++)
            {
                  Console.WriteLine("Length of row {0} is {1}",
                                    i, scores[i].Length);
            }
      }
}
```

Le langage C#

- **Tableaux**
  - Sortie écran

```
Number 0 is 5
Number 1 is 10
Number 2 is 15
Number 3 is 20
Jean       Stephane   Bernard
Sophie     Sylvie     Jeanne
Length of row 0 is 3
Length of row 1 is 4
Length of row 2 is 5
Length of row 3 is 6
Length of row 4 is 7
```

- **Tableaux – Explications**
  - Tableau à une dimension : *numbers*
  - On a ici déclaré un tableau de type int. C'est un tableau à cause des []. On affecte directement des valeurs grâce au constructeur new. Dans ce même constructeur, on indique la taille du tableau (4 éléments donc avec des index allant de 0 à 3) et on le remplit grâce aux accolades contenant les 4 éléments.
  - Puis on l'affiche en parcourant ces éléments grâce aux index.
  - Remarque : Nous n'avons pas besoin de libérer la mémoire. Le Garbage Collector s'occupe de supprimer les éléments anciens.

- **Tableaux – Explications**
  - Tableau à 2 dimensions : *names*
  - La déclaration de fait sur le même principe qu'un tableau à une dimension. Dans notre exemple, nous déclarons un tableau à deux dimensions avec [,], pour un tableau à trois dimensions il aurait fallu utiliser [,,] et ainsi de suite. Seule la quantité de mémoire est un facteur limitant.
  - Nous aurions pu déclarer tout de suite la taille du tableau avec string[2,3] mais l'utilisation de new aurait permis de paramétrer cette création.
  - L'initialisation des valeurs est quelque peu différente. Nous indiquons les valeurs non pas les unes à la suite des autres, mais en les regroupant par ligne (observer les niveaux d'imbrication des accolades).

- **Tableaux – Explications**
  - Tableau à 2 dimensions : *names*
  - Enfin, pour l'affichage nous utilisons autant d'index que de nombre de dimensions. La taille sur l'une des dimensions se récupère grâce à *names.GetLength(dimension)*. Ainsi, la première dimension se récupère avec *names.GetLength(0)*, (ne pas oublier que les index de tableaux)commencent par 0, les dimensions étant elles mêmes en interne stockées dans un tableau.

- **Tableaux – Explications**
  - Tableau de tableau : *scores*
  - On peut stocker n'importe quel type de donnée dans un tableau. Alors pourquoi ne pas stocker des tableaux dans un tableau ? Cela peut être pratique pour les tableaux ayant plusieurs dimensions, mais de longueurs différentes.
  - Ainsi on initialise d'abord le tableau conteneur, puis on initialise chaque tableau contenu dans ce dernier. Il faut remarquer la longueur différente de chaque sous tableau.
  - Pour l'affichage, on se contente ici de simplement afficher la longueur de chaque sous tableau.

Le langage C#

- **Collections**

Le lan

- **Colle**

```
// Collections.cs

using System;
using System.Collections;

public class CollectionsApp
{
      public static void Main()
      {
            ArrayList myArray=new ArrayList();
            myArray.Add("Paris");
            myArray.Add("Nice");
            myArray.Add("Bordeaux");

            Console.WriteLine("There is {0} element in myArray",
                              myArray.Count);
            foreach (string city in myArray)
                  Console.WriteLine("City={0}",city);

            myArray.Clear();
            Console.WriteLine("There is {0} element in myArray",
                              myArray.Count);

            myArray.Add("Paris");
            myArray.Add("Strasbourg");
            myArray.Add("Nice");
            myArray.Add("Marseille");
            myArray.Add("Bordeaux");

            Console.WriteLine("There is {0} element in myArray",
                              myArray.Count);
            foreach (string city in myArray)
                  Console.WriteLine("City={0}",city);

            myArray.Remove("Nice");

            Console.WriteLine("There is {0} element in myArray",
                              myArray.Count);
            foreach (string city in myArray)
                  Console.WriteLine("City={0}",city);

            myArray.RemoveAt(3);

            Console.WriteLine("There is {0} element in myArray",
                              myArray.Count);
            foreach (string city in myArray)
                  Console.WriteLine("City={0}",city);

            Console.ReadLine();
      }
}
```

- **Collections**
  - Sortie écran

```
There are 3 elements in myArray
City=Paris
City=Nice
City=Bordeaux
There are 0 elements in myArray
There are 5 elements in myArray
City=Paris
City=Strasbourg
City=Nice
City=Marseille
City=Bordeaux
There are 4 elements in myArray
City=Paris
City=Strasbourg
City=Marseille
City=Bordeaux
There are 3 elements in myArray
City=Paris
City=Strasbourg
City=Marseille
```

- **Collections – Explications**
  - Une collection est une puissante classe proposée par le framework. Son but est d'aider le développeur dans la gestion des tableaux.
  - Ici, nous avons déclaré puis utilisé un ArrayList dont le but est simplement de stocker des objets. Il ne faut pas oublier le *using System.Collection*, les collections étant dans ce namespace.
  - Grâce à la méthode Add, nous avons ajouté plusieurs chaînes de caractères (nous aurions très bien pu y mettre n'importe quel objet de n'importe quel type et même en même temps).
  - La méthode Clear() efface tous les éléments, la méthode Remove() enleve un objet particulier et RemoveAt() un objet à l'index spécifié.

- **Collections – Explications**
  - Il existe plusieurs autres collections dont le fonctionnement est assez similaire.
    - HashTable : permet d'associer une clé unique de type string à chaque élément.
    - Queue et Stack : respectivement une file et une pile.
    - SortedList : une liste triée
    - Il existe encore d'autres collections. Toutes sont dans le namespace System.Collections et namespace fils.

Le langage C#

- **Structure**

Le langage C#

- **Structure**
  - Exempl

```
// Structure1.cs

using System;

public struct Point
{
      public float x;
      public float y;
}

public class Structure1
{
      public static void Main()
      {
            Point rectUpperLeft;
            Point rectBottomRight;

            rectUpperLeft.x=10;
            rectUpperLeft.y=20;

            rectBottomRight.x=20;
            rectBottomRight.y=40;

            Console.WriteLine("The rect has a area of {0}
m²",GetArea(rectUpperLeft,rectBottomRight));

            Console.ReadLine();
      }

      public static float GetArea(Point UpperLeft,Point BottomRight)
      {
            float width=UpperLeft.x-BottomRight.x;
            width=(width > 0)? width : -width;
            float height=UpperLeft.x-BottomRight.x;
            height=(height > 0)? height : -height;

            return width*height;
      }
}
```

Le langage C#

- **Structure**
  - Résultat

```
The rect has a area of 100 m²
```

- **Structure – Exemple – Explications**
  - Nous avons tout d'abord défini une structure Point qui est composée de deux membres publics x et y.
  - Puis nous avons créé deux variables de ce nouveau type : Point. Nous pouvons alors accéder aux variables de la structure via *NomDeLaStructure.Membre*.
  - C'est ainsi que nous affectons puis nous utilisons les membres de la structure
  - L'avantage d'une structure est de pouvoir regrouper plusieurs valeurs. Cet exemple regroupe deux float, mais on aurait très bien pu imaginer une structure comportant plus de champs, et de type variés, comme par exemple une structure Country ayant comme membres : int NumberOfHabitants, int AreaSize, string Name, string BiggestCity, etc…
  - D'une part cela clarifie le code, et d'autre part, on peut passer des structures à des fonctions ou des méthode, ce qui évite d'avoir à passer autant d'arguments que de variables dans la structure.

Le langage C#

- **Création d'une classe**

Le langage C#

- **Cré**

```
// PersonApp.cs

using System;

// notre classe exemple
public class Personne
{
      // Membres publics
      public int Age;
      public string Name;

      // Membre privé
      private string _thinkings;

      // Constructeurs
      public Personne(int Age,string Name)
      {
            this.Age=Age;
            this.Name=Name;
            EvaluateThinking();
      }
      public Personne():this(0,"Unknown")
      {
      }

      // méthode privée
      private void EvaluateThinking()
      {
            this._thinkings=string.Format(
                                    "My name is {0} and I'm {1} old",
                                    Name,Age);
      }

      // méthode publique
      public void SayMyThinkings()
      {
```

Le langage C#

- **Créat**

```
            EvaluateThinking();
            Console.WriteLine(_thinkings);
      }
}

// la classe contenant le point d'entrée Main
public class PersonApp
{
      public static void Main()
      {
            // Création de quelques personnes :
            Personne Charles=new Personne(25,"Charles");
            Personne Edward=new Personne(60,"Edward");
            Personne Inconnue=new Personne();

            // Faisons parler nos chères personnes.
            Charles.SayMyThinkings();
            Edward.SayMyThinkings();
            Inconnue.SayMyThinkings();

            // On peux bien sûr modifier les champs publics
            // de la classe
            Charles.Age++;
            Inconnue.Name="Robert";
            Inconnue.Age=10;

            // On leur redemande de dire leur pensée :
            Charles.SayMyThinkings();
            Edward.SayMyThinkings();
            Inconnue.SayMyThinkings();
      }
}
```

- **Création d'une classe – Explications**
  - ***Définition de la classe***
  - le mot clé class permet d'indiquer la déclaration d'une classe. On la précède ici du mot clé public afin de pouvoir utiliser cette classe dans tout le programme. La définition de la classe est alors incluse entre une { et une }.
  - On définit ses membres. L'ordre n'a pas d'importance pour la structure du programme, mais il est recommandé de classer les membres, soit par type, soit par fonctionnalité.
  - Ici, toutes les variables membres sont déclarées au début de la classe. Remarquez l'utilisation des mot clés de portée public et private. En effet, la variable Name et la variable Age seront accessibles depuis n'importe où dans le programme, tandis que la variable _thinkings ne sera accessible que depuis la classe en elle-même. C'est pourquoi nous sommes obligés de passer par une méthode intermédiaire SayMyThinkings.

- **Création d'une classe – Explications**
  - ***Définition de la classe***
  - Ensuite viennent les constructeurs. Qu'est ce qu'un constructeur ? Un constructeur est une méthode appelée automatiquement lors de la création de l'instance de la classe en mémoire, c'est-à-dire au moment où le développeur fait un new nomdelaclasse(args).
  - Un constructeur sert généralement à initialiser des variables, ou tout autre traitement 'pré-utilisation'. Nous avons défini deux constructeurs. Le premier permet de spécifier directement la variable Name et la variable Age. Remarquez l'emploi de this qui est une référence vers l'objet même. Cela permet de distinguer la variable Age de la classe et le paramètre Age du constructeur, de même pour Name. La deuxième version du constructeur ne possède pas d'arguments, mais fait appel à l'autre constructeur en lui spécifiant des arguments. On peut en effet depuis un constructeur appeler un autre constructeur, soit de la même classe, soit de la classe de base dans le cas d'un héritage :

- **Création d'une classe – Explications**
  - ***Définition de la classe***
  - public Personne():this(0,"Unknown") // fait appel à un constructeur de la même classe.
  - public Homme():base(0,"Unknown") // fait appel à un constructeur de la classe de base (si on a hérité Homme de Personne)
  - Viennent enfin deux méthodes de classe
  - private EvaluateThinkings : Cette méthode est déclarée private ce qui signifie que seule la classe elle-même pourra y accéder. On crée généralement des méthodes privées pour du fonctionnement interne. Ici, on évalue les pensées de la personne.
  - public SayMyThinkings : Comme le membre _thinkings est déclaré en privé, nous passons par une méthode de la classe pour accéder alors à ce membre. Cette méthode faisant partie de la classe, elle pourra lire dans les membres privés. Bien sûr, il faut que cette méthode soit publique si on veut pouvoir y accéder depuis notre programme.

- **Création d'une classe – Explications**
  - ***Déroulement du programme***
  - Tout d'abord on déclare et on crée les objets. On aurait très bien pu les déclarer (avec Personne Charles;) puis les initialiser plus tard, mais il est souvent plus pratique de le faire en une seule ligne.
  - Voici nos trois objets créés. Remarquez que le compilateur sait à quel constructeur faire appel puisqu'il n'y a pas ambiguïté.
  - Nous appelons une méthode publique. Nous avons ainsi eu accès indirectement au membre _thinkings grâce à cette méthode.
  - Comme les membres Age et Name sont public, on peut les modifier directement, ce que nous faisons.
  - On vérifie alors que les objets ont bien changé leurs valeurs.

- **Création d'une classe – Explications**
  - ***Pourquoi utiliser une classe plutôt qu'une structure ?***
  - Une structure est un type par valeur alors qu'une classe est un type par référence. Cela signifie qu'une structure contient directement la valeur en mémoire de chaque membre, alors qu'un objet d'une classe est un pointeur vers ses données. L'avantage est une économie de mémoire et une simplification du codage. Par exemple, si on appelle une méthode prenant en argument une structure, la structure passée va être intégralement dupliquée pour être traitée dans la méthode. Si on reprend la même méthode, mais ayant une classe en argument, c'est juste le pointeur qui va être dupliqué, les données de la classe étant toujours les mêmes. On gagne donc en clarté (plus besoin de passer par des pointeurs comme en c++) et en occupation mémoire.

Le langage C#

- **Création d'une propriété**

- **Création d'une propriété**
- ***Définition***
  - Les propriétés s'utilisent comme une variable. La différence se situe au niveau de l'implémentation. En effet, plutôt que d'attribuer un espace mémoire et de le remplir avec une valeur, on va soi-même définir son comportement en lecture et/ou en écriture. On peut ainsi, pour chaque propriété définir une méthode get et une méthode set pour respectivement récupérer une valeur ou l'affecter. Pour implémenter une propriété en lecture seule, on n'implémente que la méthode get, et de même pour implémenter une propriété en écriture seule, on n'implémente que la propriété get.

Le langage C#

- **Créatic**
- Exemple

```
// PropertySimple.cs

using System;

public class TestProp
{
      private int  updateCount;
      private int _accessCount;
      private int  value;

      public int Value
      {
            get
            {
                  accessCount++;
                return  value;
            }
            set
            {
                  _updateCount++;
                  _value=value;
            }
      }
      public override string ToString()
      {
            return string.Format(
    "My value is {0}, i've been accessed {1} times and updated {2} times.",
                  value,
                  accessCount,
                  updateCount
                 );
      }
      public TestProp(int Value)
      {
             accessCount=0;
            _updateCount=0;
            this.Value=Value;
      }
}
public class PropertySimple
{
      static void Main(string[] args)
```

Le langage C#

- **Création d'une propriété**

```
        {
            TestProp tp=new TestProp(3);

            Console.WriteLine(tp.Value);
            tp.Value=4;
            Console.WriteLine(tp.Value);
            tp.Value=6;
            Console.WriteLine(tp.Value);
            tp.Value=-12;
            Console.WriteLine(tp.Value);
            tp.Value=806;
            tp.Value=32;
            tp.Value=1;
            Console.WriteLine(tp.Value);


            Console.WriteLine(tp.ToString());

            Console.ReadLine();
        }
}
```

Le langage C#

- **Création d'une propriété**
- Sortie écran

```
3
4
6
-12
1
My value is 1, i've been accessed 5 times and updated 7 times.
```

- **Création d'une propriété – Explications**
  - *Faire Console.WriteLine(tp.Value); revient à appeler la méthode get de la propriété Value. Dans cette propriété on a simplement retourné une valeur (qui doit d'ailleurs être du même type que la propriété).*
  - *A l'opposé, faire tp.Value=32; revient à appeler la méthode set de la propriété Value en lui donnant comme argument 32 (que l'on récupère alors via le mot clé value).*

Le lang

- **Créati**
- Exempl

```
// PersonApp2.cs
using System;

// notre classe exemple
public class Personne
{
      // L'âge
      private int  age;
      public int Age
      {
            get
            {
                  return _age;
            }
            set
            {
                   age=value;
                  EvaluateThinking();
            }
      }
      // Le nom
      private string _name;
      public string Name
      {
            get
            {
                  return  name;
            }
            set
            {
                  _name=value;
                  EvaluateThinking();
            }
      }

      // Membre privé
      private string _thinkings;

      // Constructeurs
      public Personne(int Age,string Name)
      {
            this.Age=Age;
            this.Name=Name;
      }
      public Personne():this(0,"Unknown")
      {
      }

      // méthode privée
      private void EvaluateThinking()
```

Le la

- **Cré**
- Exer

```
        {
            this. thinkings=string.Format(
                                "My name is {0} and I'm {1} old",
                                Name,Age);
        }

        // méthode publique
        public void SayMyThinkings()
        {
            Console.WriteLine(_thinkings);
        }
}

// la classe contenant le point d'entrée Main
public class PersonApp
{
        public static void Main()
        {
            // Création de quelques personnes :
            Personne Charles=new Personne(25,"Charles");
            Personne Edward=new Personne(60,"Edward");
            Personne Inconnue=new Personne();

            // Faisons parler nos chères personnes.
            Charles.SayMyThinkings();
            Edward.SayMyThinkings();
            Inconnue.SayMyThinkings();

            // on fait plus appel ici aux membres publics
            // mais aux propriétés publiques
            Charles.Age++;
            Inconnue.Name="Robert";
            Inconnue.Age=10;

            // On leur redemande de dire leur pensée :
            Charles.SayMyThinkings();
            Edward.SayMyThinkings();
            Inconnue.SayMyThinkings();

            Console.ReadLine();
        }
}
```

- **Création d'une propriété**
- Le résultat est identique à la version précédente, pourtant il y a une différence. Explications :
  - Dans cet exemple, le but est de gérer deux membres privés de la classe Personne _age et _name.
  - La différence se situe lors de l'appel à la fonction EvaluateThinkings. Elle ne se fait plus manuellement dès que l'on a besoin de _thinkings, mais automatiquement dès que l'on met à jour le nom ou l'age. Comment se déroule le programme ? La ligne Inconnue.Name="Robert"; fait appel à la fonction set de la propriété Name. Cette dernière a alors deux instructions, d'une part, l'appel à la fonction EvaluateThinkings qui est nécessaire pour la mise à jour de _thinkings, et d'autre part l'affectation de _name.

  - Remarquez le mot clé value qui indique la valeur appelée par le programmeur.

Le langage C#

- **Exceptions**
- Une exception est une alerte donnée par la CLR lors de l'exécution d'un programme en cas d'erreur (division par zéro, index du tableau hors de la limite du tableau, etc.). Cette exception peut alors être interceptée par le développeur pour qu'il traite l'erreur. Les exceptions fournissent ainsi une puissante gestion d'erreur, d'autant plus qu'il est possible de créer ses propres erreurs.

Le langage C#

- **Exceptions**
- Exemple 1

```
using System;

public class exceptions1
{
   public static void Main()
   {
      int a=5;
      int b=0;
      int c=2;

      try
      {
            Console.WriteLine("a/c={0}",a/c);
            Console.WriteLine("b/a={0}",b/a);
            Console.WriteLine("a/b={0}",a/b);
            Console.WriteLine("c/a={0}",c/a);
      }
      catch
      {
            Console.WriteLine("Error as occured");
      }
      Console.ReadLine();
   }
}
```

Le langage C#

- **Exceptions**
- Exemple 1
  - Sortie écran

```
a/c=2
b/a=0
Error as occured
```

- **Exceptions – Exemple 1 – Explications**
  - *La première chose à faire est d'encadrer les blocs de codes susceptibles de générer une erreur par un try{ }*
  - *Il faut ensuite créer le code exécuté en cas d'erreur en mettant un bloc catch {}. Le code inclut dans ce bloc sera alors exécuté dès qu'une erreur sera détectée. On dit alors qu'une exception est levée.*
  - *Le code contenu dans le try{} est alors immédiatement interrompu, c'est pourquoi seules deux lignes de notre code sont affichées à l'écran.*

Le

-
- son

```
using System;

public class exceptions2
{
   public static void Main()
   {
     int a=5;
     int b=0;
     int c=2;

     try
     {
       Console.WriteLine("a/c={0}",a/c);
       Console.WriteLine("b/a={0}",b/a);
       Console.WriteLine("a/b={0}",a/b);
       Console.WriteLine("c/a={0}",c/a);
     }
     catch (DivideByZeroException exc)
     {
       Console.WriteLine("It is forbidden to divide by zero !");
     }
     catch(IndexOutOfRangeException exc)
     {
       Console.WriteLine("The specified index is out of range");
     }
     catch (Exception exc)
     {
       Console.WriteLine("An error has occured : {0}",exc.Message);
     }

     Console.ReadLine();
   }
}
```

Le langage C#

- **Exceptions**
- Exemple 2 – Sortie écran

```
a/c=2
b/a=0
It is forbidden to divide by zero !
```

Le langage C#

- **Exceptions**
- Exemple 2 – Explications
- On précise ici le type de l'exception qu'il faut intercepter. On précise autant de catch que l'on souhaite, afin de mieux traiter les erreurs.
- Comme toutes les exceptions héritent de la classe System.Exception, on termine le traitement d'erreur par un catch de Exception. Ainsi, toutes les erreurs qui n'auront pas été spécifiquement traitées seront traitées par ce bloc.
- Remarquez également que l'objet exception capturé contient des propriétés donnant des indications sur l'erreur, comme le message associé.

Le lan

- **Exce**
- Exem sera exécu bloc finally

```
using System;

public class Exception3
{
   public static void Main()
   {
      int a=5;
      int b=0;
      int c=2;

      try
      {
         Console.WriteLine("a/c={0}",a/c);
         Console.WriteLine("b/a={0}",b/a);
         Console.WriteLine("a/b={0}",a/b);
         Console.WriteLine("c/a={0}",c/a);
      }
      catch (DivideByZeroException exc)
      {
         Console.WriteLine("It is forbidden to divide by zero !");
      }
      catch(IndexOutOfRangeException exc)
      {
         Console.WriteLine("The index spécified is out of range");
      }
      catch (Exception exc)
      {
         Console.WriteLine("Error as occured : {0}", exc.Message);
      }
      finally
      {
         Console.WriteLine("This line will always be printed !");
      }

      Console.ReadLine();
   }
}
```

- **Exceptions**
- Exemple 3 – On peut, enfin, ajouter un bloc qui sera exécuté systématiquement en le plaçant dans un bloc finally.

Le langage C#

- **Exceptions**
- Exemple 3 – Explications
- Le code intégré dans le bloc finally sera alors toujours exécuté, qu'une exception soit levée ou non.

Le langage C#

- **Enumération**

Le l

- **En**

```
// enum.cs

using System;

public class EnumApp
{
      public enum City
      {
            Paris,
            Strasbourg,
            Marseille=42,
            Brest
      }
      public static void Main()
      {
            City Departure;
            City Arrival;

            Departure=City.Paris;
            Console.WriteLine("I am in {0}",Departure.ToString());

        Arrival=City.Marseille;

            int hoursToTravel;

            switch(Arrival)
            {
                  case City.Strasbourg:
                        hoursToTravel=4;
                        break;
                  case City.Marseille:
                        hoursToTravel=3;
                        break;
                  case City.Brest:
                        hoursToTravel=2;
                        break;
                  default:
                        hoursToTravel=0;
                        break;
            }

            Console.WriteLine(
                  "The travel to {0} takes {1} hours with train",
                  Arrival.ToString(),hoursToTravel);

            Console.ReadLine();
      }
}
```

Le langage C#

- **Enumération**
- Sortie écran

```
The travel to Marseille takes 3 hours with train
```

Le langage C#

- **Enumération**
- Explications
- La déclaration d'une énumération se fait très simplement grâce à enum. On place entre les accolades les différentes valeurs énumérées. En interne, chaque énumération est de type entier et se numérote de façon incrémentale de 0 à n, où n est le nombre d'énumération moins un.
- Il est toutefois possible de préciser les valeurs que les énumérations vont prendre en mettant par exemple : Marseille = 13 plutôt que de juste mettre Marseille.

Le langage C#

- **Enumération**
- Explications
- .Net propose un mécanisme très intéressant qui consiste à mettre .ToString() derrière une variable de type énumérée. Cela vous permet d'afficher le nom interne de la valeur énumérée.
- Utilisez un maximum les types énumérés. En effet, cela facilite l'écriture du code (il est en effet plus facile de relire City.Marseille plutôt que 13. De plus, les valeurs énumérées étant un ensemble fini et connu, il est moins probable d'avoir des bug que d'avoir utilisé un simple int stockant l'état. Par exemple, si on a : if(Arrival==12), qui peut savoir si 12 est bien une valeur possible ?

Le langage C#

- **Héritage et Polymorphisme**

Le la

- **Hé**

```
// inheritence.cs

using System;

class A
{
	protected string AProtectedMethod()
	{
		return "I'm in A.AProtectedMethod()";
	}
	public void TestScope()
	{
		Console.WriteLine(AProtectedMethod());
	}
	public void MyMethodFromAClass()
	{
		Console.WriteLine("I'm in A.MyMethodFromAClass()");
	}
	public void AnotherMethod()
	{
		Console.WriteLine("I'm in A.AnotherMethod()");
	}
	public virtual void MyVirtualMethod()
	{
		Console.WriteLine("I'm in A.MyVirtualMethod()");
	}
}
class B: A
{
	public void AccessingProtectedFromParent()
	{
		Console.WriteLine(AProtectedMethod());
	}
	public void MyMethodFromBClass()
	{
		Console.WriteLine("I'm in A.MyMethodFromACLass()");
	}
	new public void AnotherMethod()
```

Le lan

- **Hérit**

```
        {
            Console.WriteLine("I'm in B.AnotherMethod()");
        }
        public override void MyVirtualMethod()
        {
            Console.WriteLine("I'm in B.MyVirtualMethod()");
        }
}
class Inheritence
{
        static void Main()
        {
            A a=new A();

            Console.WriteLine("\na.MyMethodFromAClass()");
            a.MyMethodFromAClass();
            Console.WriteLine("\na.AnotherMethod()");
            a.AnotherMethod();
            Console.WriteLine("\na.MyVirtualMethod()");
            a.MyVirtualMethod();
            Console.WriteLine("\na.TestScope()");
            a.TestScope();

            B b=new B();
            Console.WriteLine("\nb.MyMethodFromBClass()");
            b.MyMethodFromBClass();
            Console.WriteLine("\nb.MyMethodFromAClass()");
            b.MyMethodFromAClass();
            Console.WriteLine("\nb.AnotherMethod()");
            b.AnotherMethod();
            Console.WriteLine("\nb.AccessingProtectedFromParent()");
            b.AccessingProtectedFromParent();

            A ha=new B();
            Console.WriteLine("\nha.AnotherMethod()");
            ha.AnotherMethod();
            Console.WriteLine("\nha.MyVirtualMethod()");
            ha.MyVirtualMethod();

            Console.ReadLine();
        }
}
```

Le langage C#

- **Héritage et Polymorphisme**
- Sortie écran

```
a.MyMethodFromAClass()
I'm in A.MyMethodFromAClass()

a.AnotherMethod()
I'm in A.AnotherMethod()

a.MyVirtualMethod()
I'm in A.MyVirtualMethod()

a.TestScope()
I'm in A.AProtectedMethod()

b.MyMethodFromBClass()
I'm in A.MyMethodFromACLass()
```

```
b.MyMethodFromAClass()
I'm in A.MyMethodFromAClass()

b.AnotherMethod()
I'm in B.AnotherMethod()

b.AccessingProtectedFromParent()
I'm in A.AProtectedMethod()

ha.AnotherMethod()
I'm in A.AnotherMethod()

ha.MyVirtualMethod()
I'm in B.MyVirtualMethod()
```

- **Héritage et Polymorphisme**
- Explications
- Observer la ligne : class B: A. On crée de cette façon une classe B qui hérite de A. La classe B possède ainsi tous les membres protégés ou publics de A. Observer la sortie du programme pour voir quelles sont les méthodes définies pour A accessibles par B.
- a.MyMethodFromAClass et a.AnotherMethod: ici, rien de spécial à signaler, on fait appel à deux méthodes de la classe A.

- **Héritage et Polymorphisme**
- Explications
- a.MyVirtualMethod : le mot clé virtual ne change rien pour le déroulement de la méthode pour la classe A.
- a.TestScope : on accède à une méthode private ou protected d'un objet que via l'objet lui-même ou par une méthode publique. C'est donc le seul moyen d'appeler un membre privé ou protégé d'une classe depuis le programme.
- b.MyMethodFromBClass : rien de spécial. C'est une méthode la classe B.

- **Héritage et Polymorphisme**
- Explications
- b.MyMethodFromAClass : on voit ici les avantage de l'héritage. La classe B ayant hérité de la classe A, on peut appeler les méthodes de la classe A qui sont publiques depuis un objet de la classe B.
- b.AnotherMethod : ici, bien que le nom de la méthode soit le même que l'une des méthodes de la classe A et que la classe B hérite de A, c'est une méthode de B. Le mot clé new permet en effet d'indiquer au compilateur que la fonction a le même nom, mais est totalement différente dans le corps.

- **Héritage et Polymorphisme**
- Explications
- b.AccessingProtectedFromParent : On accède depuis cette méthode à l'une des méthodes protégées de la classe de base.
- ha.AnotherMethod : bien que ha soit initialisé avec new B(), ha est déclarée en tant que A et donc c'est la méthode A.AnotherMethod qui est appelée, et non B.AnotherMethod.

- **Héritage et Polymorphisme**
- Explications
- ha.MyVirtualMethod : contrairement à la ligne précédente, ici c'est bien B.MyVirtualMethod qui est appelée. Ceci est dû au mot clé virtual qui permet d'appeler non pas la méthode de la classe précisée à la déclaration, mais à la méthode de la classe instanciée. Il a fallu également employer le mot clé override pour préciser que c'est une redéfinition de la méthode virtuelle de la classe de base.

Le langage C#

- **Interfaces**

Le langage C#

- **Interfaces**
- Définition
- Une interface est en quelque sorte un cahier des charges qu'une classe doit respecter. Imaginons que nous ayons une application utilisant des formes géométriques. Chaque forme possède au minimum une hauteur et une largeur. Ainsi, nous déclarons une interface comprenant un membre longueur et un membre largeur. Chaque forme devant alors implémenter cette interface. Implémenter signifie que le développeur est obligé de prévoir tous les membres de l'interface.

Le langage C#

- **Interfaces**
- Définition
- On pourrait alors se demander pourquoi ne pas utiliser une classe de base. La différence se situe dans le fait qu'avec une classe de base il aura déjà fallu implémenter le membre alors qu'avec une interface, le développeur devra le faire lui-même. On peut ainsi créer des fonctions génériques demandant en argument une objet implémentant l'interface, puisque cet objet possèdera forcement les membres de l'interface.

Le la

- **Inte**
- Exer

```
// interface.cs

using System;

interface IDimensions
{
      float Length();
      float Width();
}

class Box : IDimensions
{
      float lengthInches;
      float widthInches;

      public Box(float length, float width)
      {
            lengthInches = length;
            widthInches = width;
      }
      float IDimensions.Length()
      {
            return lengthInches;
      }
      float IDimensions.Width()
      {
            return widthInches;
      }

      public static void Main()
      {
            Box myBox = new Box(30.0f, 20.0f);
            IDimensions myDimensions = (IDimensions) myBox;
            System.Console.WriteLine("Length: {0}", myDimensions.Length());
            System.Console.WriteLine("Width: {0}", myDimensions.Width());

            Console.ReadLine();
      }
}
```

Le langage C#

- **Interfaces**
- Sortie console

```
Length: 30
Width: 20
```

Le langage C#

- **Interfaces**
- Explications
- Nous déclarons tout d'abord l'interface en précisant quels seront les membres et méthode à implémenter. En effet, utiliser les interfaces permet de forcer le développeur à développer un certain nombre de méthode. Dans notre exemple, nous déclarons deux méthodes (float Length() et float Width()).

Le langage C#

- **Interfaces**
- Explications
- Notre classe Box implémentant cette interface, le développeur doit alors développer ces méthodes. Box implémente ces interface en mettant en nom de méthode IDimensions.Length() par exemple, ce qui indique que c'est l'implémentation de Length de l'interface IDimensions.
- On peut alors appeler la méthode d'une interface depuis un objet, et ce quel que soit son type, tant que l'objet implémente l'interface.

Le langage C#

- **Interfaces**
- Explications
- Si besoin, le développeur peut implémenter autant d'interface qu'il le souhaite
- Il faut commencer obligatoirement le nom de l'interface par un 'I'.
- Le framework .Net comprend un grand nombre d'interface à utiliser. Par exemple, l'interface ICollection permet de créer des collections personnalisées.

Le langage C#

- **Surcharge d'opérateurs**

Le langage C#

- S
- L

com

le

```
// OperatorOverload.cs

using System;

public class Complex
{
      public int Real;
      public int Imaginary;

      public Complex(int Real, int Imaginary)
      {
            this.Real = Real;
            this.Imaginary = Imaginary;
      }

      public static Complex operator +(Complex c1, Complex c2)
      {
            return new Complex( c1.Real + c2.Real,
                        c1.Imaginary + c2.Imaginary);
      }

      public override string ToString()
      {
            return(String.Format("{0} + {1}i", Real, Imaginary));
      }
}

public class OperatorOverload
{
      static void Main()
```

Le langage C#

- **Surcharge d'opérateurs**

```
        {
                Complex num1 = new Complex(2,3);
                Complex num2 = new Complex(3,4);
                Complex sum = num1 + num2;

                Console.WriteLine("First complex number:  {0}",num1);
                Console.WriteLine("Second complex number: {0}",num2);
                Console.WriteLine("The sum of the two numbers: {0}",sum);

                Console.ReadLine();
        }
}
```

- **Surcharge d'opérateurs – Explications**

Nous avons tout d'abord créé une classe complex contenant deux membres publics.

• Nous ne pouvons additionner deux objets directement. Il faut donc surcharger l'opérateur +. Remarquez que la méthode est statique.

• Cette méthode retourne un Complex et prend deux arguments de type Complex, mais nous pouvons très bien créer autant de méthodes que nécessaire pour les autres types (avec des int en paramètres par exemple).

• Les opérateurs du c# sont : +, -, !, ~, ++, --, true, false, *, /, %, &, |, ^, <<, >>, ==, !=, <, >, <=, >=.

Le langage C#

- **Délégations**

Le langage C#

- **Délégations**
- Les délégations sont un type un peu particulier. Elles permettent de fournir une fonction en paramètre pour déléguer l'exécution d'un morceau de programme. C'est l'équivalent des pointeurs de fonctions en C/C++.

Le langage C#

- **Délégations**

```
// Delegating.cs

using System;

public delegate void DelegateType(int i);

public class Delegating
{
      public static void Main()
      {
            DelegateType del=new DelegateType(MyFunction);
            del(10);
            del = new DelegateType(MyFunction2);
            del(25);
            Console.ReadLine();
      }

      public static void MyFunction(int i)
      {
            Console.WriteLine("I'm in MyFunction and i={0}",i);
      }

      public static void MyFunction2(int i)
      {
            Console.WriteLine("I'm in MyFunction2 and i={0}",i);
      }
}
```

- **Délégations – Explications**
- On déclare tout d'abord un delegate via le mot clé homonyme. On lui précise les arguments que la fonction déléguée devra accepter (ici un int).
- On crée ensuite un objet de ce type en lui précisant quelle est la fonction appelée. On peut alors appeler la fonction désignée par l'intermédiaire de l'objet délégué.

Le langage C#

- **Evènements**

Le langage C#

- **Evènements**
- Les événements permettent de capturer une action du programme. Ainsi, lorsque l'on clique sur un bouton, l'événement Click du bouton est levé. On peut capturer les événements émis par les classes du framework mais aussi créer ses propres événements.

Le la

- **Evèn**

```
// Eventsl.cs

using System;

public class Test
{
      public event EventHandler Changed;

      private string  msg;
      public string Msg
      {
            get
            {
                  return _msg;
            }
            set
            {
                  _msg=value;
                  if (Changed != null) Changed(this,EventArgs.Empty);
            }
      }
}
public class TestHooking
{
      Test _test;
      public TestHooking()
      {
            _test = new Test();
            StartHooking();
      }
      public void Test_OnChanged(object sender,EventArgs e)
      {
            Console.WriteLine("I hook the Changed event from _test");
      }
      public void ChangeString(string Msg)
      {
            Console.WriteLine("Changing _test.Msg to {0}",Msg);
            _test.Msg=Msg;
      }
      public void StartHooking()
```

- **Evènements – Exemple 1**

```
        {
            Console.WriteLine("Starting Hooking");
            _test.Changed += new EventHandler(Test_OnChanged);
        }
        public void StopHooking()
        {
            Console.WriteLine("Stoping Hooking");
            _test.Changed -= new EventHandler(Test_OnChanged);
        }
}
public class Events1
{
        public static void Main()
        {
            TestHooking th=new TestHooking();

            th.ChangeString("hello");
            th.StopHooking();
            th.ChangeString("Titi");
            th.StartHooking();
            th.ChangeString("Koala");
            Console.ReadLine();
        }
}
```

Le langage C#

- **Evènements – Exemple 1**
- Résultats

```
Starting Hooking
Changing  test.Msg to hello
I hook the Changed event from  test
Stoping Hooking
Changing  test.Msg to Titi
Starting Hooking
Changing  test.Msg to Koala
I hook the Changed event from _test
```

Le langage C#

- **Evènements – Exemple 1**
- Résultats

```
Starting Hooking
Changing  test.Msg to hello
I hook the Changed event from  test
Stoping Hooking
Changing  test.Msg to Titi
Starting Hooking
Changing  test.Msg to Koala
I hook the Changed event from _test
```

- **Evènements – Exemple 1 – Explications**
- Un événement se déclare en plusieurs étapes. La première est de déclarer l'événement dans la classe en l'associant à un gestionnaire d'événement (EventHandler). Ce gestionnaire est de type delegate auquel nous pourrons associer une méthode qui sera donc le traitement de notre événement.
- Ainsi dans notre exemple, dans la classe TestHooking, l'événement test.Changed est capturé via la ligne : test.Changed += new EventHandler(Test_OnChanged);

- **Evènements – Exemple 1 – Explications**
- En réalité, comme EventHandler est un type par délégation, la classe Test délègue une méthode. C'est pourquoi, pour lever l'événement Changed de la classe Test, on teste d'abord si Changed est null (donc la délégation n'est pas effectuée) avec d'appeler la fonction déléguée.

Le

- **E**

```
// Events2.cs

using System;

public delegate void FinishedCarEventHandler(
                    object sender, FinishedCarEventArgs e);

public class FinishedCarEventArgs : EventArgs
{
      private string _model;
      public string Model
      {
            get{ return  model; }
      }
      public FinishedCarEventArgs(string Model)
      {
             model=Model;
      }
}

public class CarFactory
{
      public event FinishedCarEventHandler FinishedCar;

      public void BuildCar(string Model)
      {
            Console.WriteLine("I built a {0}",Model);
            if (FinishedCar != null)
            {
                  FinishedCar(this,new FinishedCarEventArgs(Model));
            }
      }
}

public class Events2
{
      public static void Main()
      {
            CarFactory cf=new CarFactory();

            cf.FinishedCar += new FinishedCarEventHandler(OnFinishedCar);
            cf.BuildCar("R5");
            cf.BuildCar("F350");
            cf.BuildCar("Laguna");

            Console.ReadLine();
      }

      public static void OnFinishedCar(
                  object sender,FinishedCarEventArgs e)
      {
            Console.WriteLine(
                 "The factory has finished building a {0}",e.Model);
      }
}
```

Le langage C#

- **Evènements – Exemple 2**
- Sortie écran

```
I built a R5
The factory has finished building a R5
I built a F350
The factory has finished building a F350
I built a Laguna
The factory has finished building a Laguna
```

Le langage C#

- **Evènements – Exemple 2 – Explications**
- Le principe est le même que l'exemple précédent, mais nous avons développé notre propre type d'évènement.
- Nous avons pour cela créé un type par délégation pour capturer notre événement personnalisé. Chaque événement possède deux arguments : sender (object) qui est la source de l'événement, et e(EventArgs) qui sont les arguments de l'évenément.
- Nous avons alors justement hérité la classe EventArgs pour créer nos propres arguments d'évènement en rajoutant ici une chaîne de caractères Model.

Le langage C#

- **Attributs**

Le langage C#

- **Attributs**
- Un attribut permet de spécifier des paramètres supplémentaires à une méthode, une classe, une propriété. Ces paramètres peuvent servir à donner encore plus de détails dans les spécificités d'un objet, d'une classe, etc. que les simples « public », « private », etc. On peut ar exemple, spécifier qu'une classe est visible depuis un webservice avec l'attribut [WebMethod], ou encore indiquer qu'une méthode s'exécute dans un même espace mémoire en cas d'utilisations de threads avec [StaThread]

Le langage C#

- **Attributs**

```
// attributes1.cs

using System;
using System.Diagnostics;

[Flags]
public enum StateFlags
{
      WithWheels= 0x0001,
      WithEngine= 0x0002,
      WithLights= 0x0004,
      WithBreaks= 0x0008
}

public class Attributes1
{
      public static void Main()
      {
            StateFlags State;
```

Le langage C#

- **Attributs**
- Sortie écran

```
No, there is no light installed
CurrentFlags are WithEngine, WithBreaks
```

- **Attributs - Explications**
- Ici nous avons via l'attribut [Flags] (dont l'applicabilité est sur les énumérations exclusivement) indiqué au compilateur qu'il doit traiter notre énumération comme un champ de bit, et non une valeur entière.
- Le fonctionnement de la fonction diffère sans cet attribut. En effet, en effaçant cet attribut, même si le programme fonctionne, il sera incapable d'afficher correctement State. Si le flag est mis, il reconnaîtra la combinaison des deux valeurs. On voit donc comment avec les attributs on peut influencer le déroulement du programme.