# C# - Types et conversions
# 420-216-FX

## Types primitifs

Source : http://msdn.microsoft.com/fr-fr/library/ms228360%28v=vs.90%29.aspx

| Nom court | Type .NET | Description | Nombre de bits | Plage de valeurs |
|---|---|---|---|---|
| **byte** | Byte | Entier non signé | 8 | 0 à 255 |
| **sbyte** | SByte | Entier signé | 8 | - 128 à 127 |
| **short** | Int16 | Entier signé | 16 | - 32 768 à 32 767 |
| **ushort** | UInt16 | Entier non signé | 16 | 0 à 65535 |
| **int** | Int32 | Entier signé | 32 | - 2 147 483 648 à 2 147 483 647 |
| **uint** | UInt32 | Entier non signé | 32 | 0 à 4 294 967 295 |
| **long** | Int64 | Entier signé | 64 | - 922337203685477508 à 922337203685477507 |
| **ulong** | UInt64 | Entier non signé | 64 | 0 à 18446744073709551615 |
| **float** | Single | Type virgule flottante à simple précision. Base 2. | 32 | -3.402823e38 à 3.402823e38 |
| **double** | Double | Type virgule flottante à double précision. Base 2. | 64 | -1.79769313486232e308 à 1.79769313486232e308 |
| **char** | Char | Caractère Unicode unique | 16 | Symboles Unicode utilisés dans le texte |
| **bool** | Boolean | Type booléen logique | 8 | *true* ou *false* |
| **string** | String | Séquence/chaîne de caractères | | |
| **decimal** | Decimal | Type fractionnaire ou intégral précis qui peut représenter des nombres décimaux avec 29 bits significatifs. Base 10. | 128 | ±1,0 × 10e-28 à ±7,9 × 10e28 |

- Un **type non signé** ne permet pas de représenter les nombres négatifs mais permet de représenter **deux fois plus de nombres positifs** incluant le **zéro** (comparez *short* avec *ushort*).

- Il n'est **pas** recommandé d'initialiser une variable de type primitif si cette valeur n'est pas utilisée. Par contre, dans certains cas précis, il est nécessaire d'initialiser une variable. Par exemple :
  - pour un compteur ;
  - pour un accumulateur ;
  - et lorsqu'on crée une chaîne de caractères par concaténation en commençant avec la chaîne vide :

```
String chaine = "";
chaine += ...;
```

- Un **montant** devrait être représenté avec le type **decimal** plutôt que float ou double afin de ne pas obtenir d'erreurs dans les décimales sur le résultat des calculs impliquant des montants.

- Attention : un caractère est délimité par des **apostrophes** et non pas des guillemets. Exemple :

```
char uneLettre = 'a';
```

## Type objet

- Classe mère de tous les objets : **Object**
- On ne peut pas utiliser un objet si on ne lui a pas assigné une valeur préalablement. Pour indiquer qu'un objet ne contient rien, on peut lui assigner la valeur **null**. Par contre, on ne peut pas appeler une méthode pour un objet qui est à **null**; dans un tel cas, on lèverait l'exception ***NullReferenceException***.
- Il n'est **pas** recommandé d'initialisé un objet à null si cette valeur n'est pas utilisée.
- Méthode que tous les objets possèdent : **ToString()**. Permet la conversion d'un objet vers une chaîne de caractères (string). Par défaut, affiche le type de l'objet (sa classe avec son *namespace*) c'est donc très peu utile. On peut la redéfinir (voir l'exemple de code).

## Le type *enum*

- Permet de créer de nouveaux types correspondant à nos besoins et limitant les plages de valeurs permises.
- Par exemple, il pourrait être intéressant de regrouper un ensemble de valeurs représentant des états civils afin d'éventuellement y accéder en utilisant leurs noms. Ceci permettrait à un développeur d'utiliser le nom du groupe dans lequel ont été définies ces valeurs :

```
public enum EtatCivil
{
	Marie,
	Veuf,
	Celibataire,
	Divorce,
	Religieux
}
```

- L'énumération doit être mise dans un fichier portant son nom suivit de l'extension "cs". Pour l'exemple ci-dessus, "EtatCivil.cs".
- "`EtatCivil`" est un type qui peut être utilisé comme tous les autres types.
- On donne une valeur à un variable de type "enum" de la manière suivante :

```
EtatCivil monEtat = EtatCivil.Marie;
```

## Conversions

- **Conversion implicite** (étendue) : vers un type ayant une plus grande plage de valeurs. Exemple :

```
byte b = 200;
int i = b;
```

```
float f = 12.345f;
double d = f;
```

- **Conversion explicite** (*casting*, transtypage) : vers un type ayant une plus petite plage de valeurs. Il y a un risque de perte d'information. Exemple :

```
double d = 88.999;
float f = (float)d;
```

- **Conversion vers un nombre à partir d'une chaîne** : Méthodes "**Parse**" ou bien **TryParse**" de la classe du type cible. Exemple :

```
String intChaine = "456";
int entier1 = Int32.Parse(intChaine);
bool succes = Int32.TryParse(intChaine, out entier2);
```

- Utilisation de la classe "**Convert**" pour différents types de conversion. Exemples de méthodes : "ToInt32", "ToFloat", etc.

## Questions ?

- Pour chaque énoncé ci-dessous, donnez le type le plus approprié si on désire utilisez le moins de bits possible pour représenter la donnée :
    1. Lorsque la compagnie *TrucEnVrac* fait une commande à un fournisseur, elle doit spécifier la quantité de chaque article désiré; cette quantité ne doit pas être nulle et ne doit jamais dépasser 50 000.
    2. La même compagnie *TrucEnVrac* doit aussi spécifier les prix des produits qu'elle vend; les prix seront toujours compris entre 0.00$ et 1 000 000.00$.
    3. Vous travaillez avec un scientifique qui vous demande de faire des calculs complexes avec beaucoup de précision. Il aimerait obtenir au moins 10 chiffres significatifs (en notation scientifique) lorsque votre système informatique fournira la réponse au calcul demandé.
    4. Au début des fichiers audio, il y a des métadonnées en format ID3 contenant de l'information sur la pièce musicale (titre, artiste, album, genre, etc.) On désire représenter le genre dont la plage de valeur s'étend de 1 à 79.
    5. Dans la ligue nationale de Hockey, on veut représenter la statistique **+/-** d'un joueur. Il est à tout fin pratique impossible que cette valeur soit inférieure à -300 ou bien supérieure à +300.