Plan

Partie 1 C#

- Premiers programmes
- Types et opérations de base
- Les structures de contrôle
- Types valeur et types référence
- Classes et objets
- Composition, Héritage, polymorphisme
- Abstraction, Interfaces
- Les classes du Framework.NET
- Mécanismes utilisés en C#

Abstraction

- **Il arrive que l'on n'ait pas de code à mettre dans une méthode parce qu'il y a un manque d'information à ce niveau de l'arbre d'héritage.**
- **Par exemple pour la classe FigureGéométrique suivante il n'y a rien à mettre dans la méthode Dessine().**
  - En effet, à ce niveau de l'héritage on ne sait pas quel type de figure géométrique on instancie.
- **Une telle classe de base permet d'imposer des opérations à ses classes dérivées.**

Abstraction

- **Une classe abstraite est une classe qui doit déléguer complètement l'implémentation de certaines de ces méthodes à ses classes dérivées.**
- **Ces méthodes qui ne peuvent être implémentées s'appellent des méthodes abstraites (ou virtuelles pures).**
- **La conséquence fondamentale**
  - une classe abstraite n'est pas instanciable.
- **Une autre conséquence**
  - une méthode abstraite ne doit pas avoir une visibilité privée.

Exemple: L'abstraction

```
abstract class FigureGéométrique
{
    private String nom;
    public abstract void Dessine();
}
class Cercle : FigureGéométrique
{
    private       Point    Centre;
    private       double   Rayon;
    public Cercle( Point Centre, double Rayon)
    {this.Centre = Centre;this.Rayon  = Rayon;}
    public override void Dessine ()
    {
          // dessine un Cercle à partir de son centre et de son
    rayon
    }
}
...
// erreur de compilation !
// ne peut instancier une classe abstraite !
FigureGéométrique UneFigure = new FigureGéométrique();
// OK
FigureGéométrique UneFigure = new Cercle(new Point(1,2),3);
...
```

Les interfaces

- **Il existe des classes abstraites très particulières.**
  - Ce sont celles qui n'ont que des méthodes abstraites.
- **En C# on les appelle les interfaces.**
- **On dit qu'une classe implémente une interface au lieu de dérive d'une interface.**
- **Le point fondamental**
  - une classe peut implémenter plusieurs interfaces (et/ou dériver d'une seule classe de base).
- **Dans la déclaration d'une interface**
  - les méthodes ne peuvent avoir de niveau de visibilité. C'est aux classes qui l'implémentent d'en décider.

Exemple: Les interfaces

```
using System;

interface IA{void f(int i);}

interface IB{void g(double d);}


class C : IA, IB
{
   public void f(int i)    { Console.WriteLine("f de C {0}",i);}
   public void g(double d){ Console.WriteLine("g de C {0}",d);}
}
```

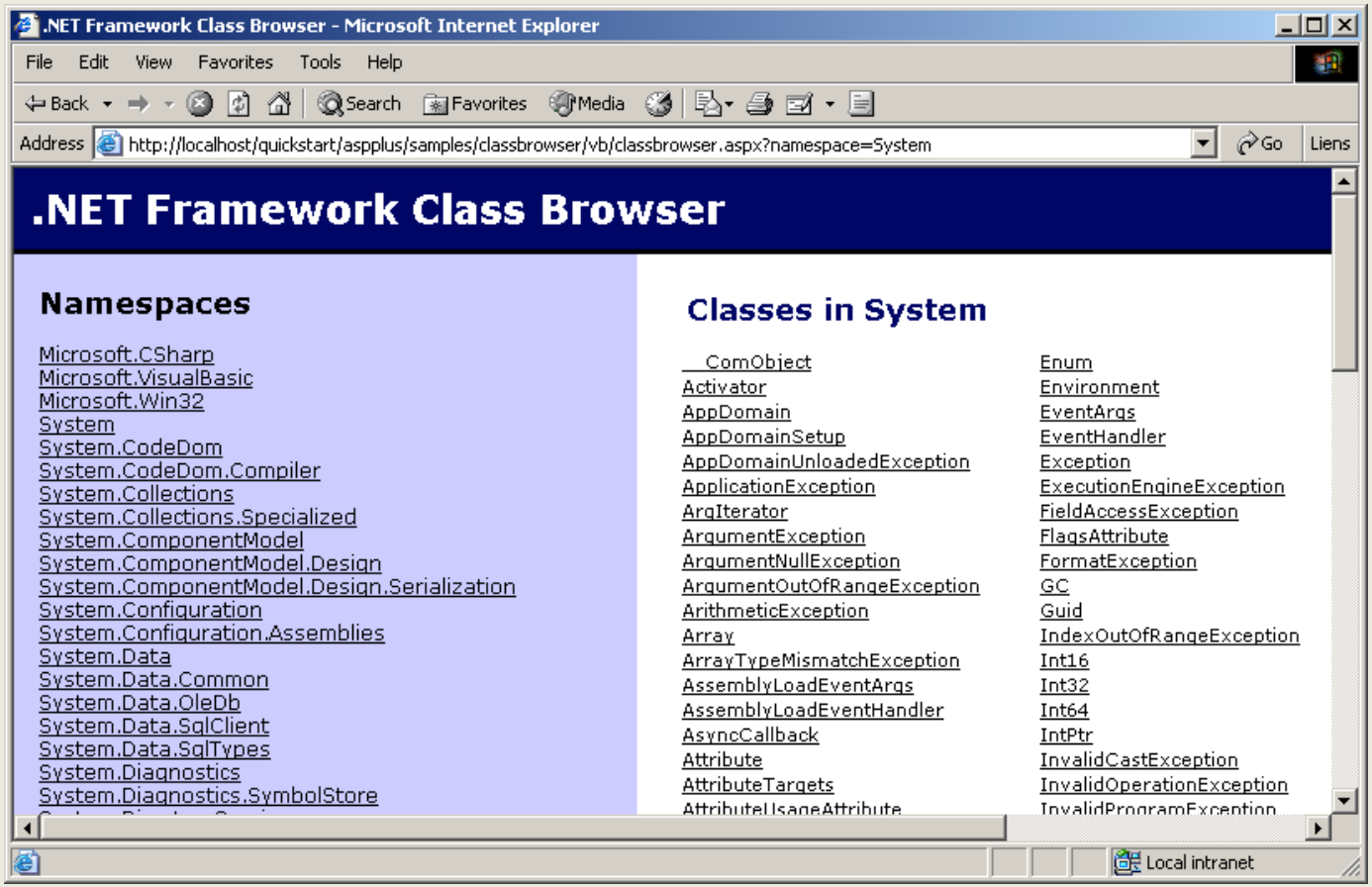
Plan

Partie 1 C#

- Premiers programmes
- Types et opérations de base
- Les structures de contrôle
- Types valeur et types référence
- Classes et objets
- Composition, Héritage, polymorphisme
- Abstraction, Interfaces
- Les classes du Framework.NET
- Mécanismes utilisés en C#

Les classes du Framework.NET

- **Regroupées dans des espaces de noms**
- **Organisées hiérarchiquement**



Classe Object

- **Toutes les classes héritent implicitement de Object**
- **Méthodes publiques**

Public Constructors

Object Constructor	Initializes a new instance of the **Object** class.

Public Methods

Equals	Overloaded. Determines whether two **Object** instances are equal.
GetHashCode	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType	Gets the Type of the current instance.
S ReferenceEquals	Determines whether the specified **Object** instances are the same instance.
ToString	Returns a String that represents the current **Object**.

Toutes les variables sont des objets

- **Tous les types de base sont des structures**

Reserved word	Aliased type
sbyte	System.SByte
byte	System.Byte
short	System.Int16
ushort	System.UInt16
int	System.Int32
uint	System.UInt32
long	System.Int64
ulong	System.UInt64
char	System.Char
float	System.Single
double	System.Double
bool	System.Boolean
decimal	System.Decimal

Toutes les variables sont des objets

- **Exemple : un entier (int) est considéré comme un objet de la structure Int32 de l'espace des noms System**

Class System.Int32

Fields

Visibility	Type	Name
public static const	Int32	MaxValue
public static const	Int32	MinValue

Methods

Visibility	Return	Name	Parameters
public	Int32	CompareTo	(Object value)
public	Boolean	Equals	(Object obj)
public	Int32	GetHashCode	( )
public	TypeCode	GetTypeCode	( )
public static	Int32	Parse	(String s , NumberStyles style)
public static	Int32	Parse	(String s)
public static	Int32	Parse	(String s , NumberStyles style , IFormatProvider provider)
public static	Int32	Parse	(String s , IFormatProvider provider)
public	String	ToString	(IFormatProvider provider)
public	String	ToString	(String format , IFormatProvider provider)
public	String	ToString	(String format)
public	String	ToString	( )

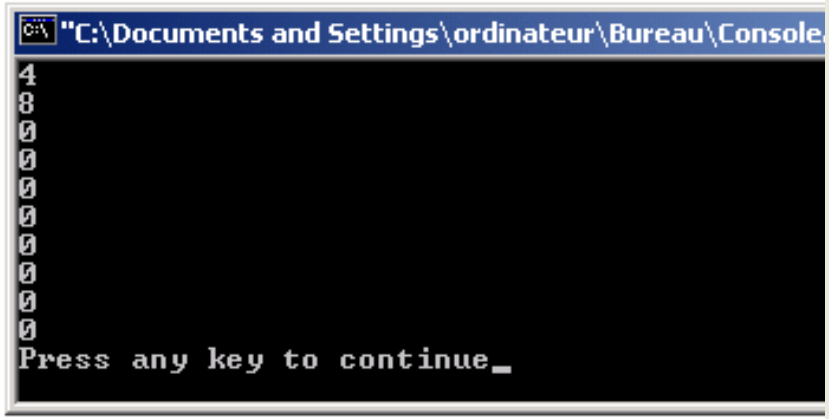
```
int i=3;
string str;
str = i.ToString();
```

Toutes les variables sont des objets

- **Les tableaux peuvent être considérés comme des objets de la classe Array de l'espace de noms System.**
- **On peut donc utiliser les membres de la classe Array.**

```
class Prog
{
    /// <summary>
    /// Point d'entrée principal de l'application.
    /// </summary>
    [STAThread]
    static void Main(string[] args)
    {
        int [ ] t = new int[10];
        t[0]= 4;
        t[1]= 8;

        for(int i=0; i<t.GetLength(0);i++)
            Console.WriteLine("{0}   ",t[i]);
```



Transtypage

- **Transtypage automatique**
  - Si `ClB` hérite de `ClA`, on peut fournir un objet b, instance de `ClB` quand on attend une instance de `ClA`.

- **Transtypage explicite**
  - Nécessaire quand on veut convertir (lorsque cela est possible) une instance d'une classe `ClA` en une instance d'une classe `ClB`, héritant de `ClA`

```
ClA a;
a = … // a est en fait une référence vers une instance de ClB
ClB b = (ClB)a;
```

Les opérateurs is et as

- **L'opérateur is sert à déterminer à l'exécution si une expression peut être transtypée dans un type donné.**
  - Cet opérateur retourne un booléen.
  - on réalise effectivement le transtypage lorsque " true" est retourné
- **L'opérateur as permet d'effectuer ces deux étapes.**
  - Si le transtypage ne peut avoir lieu, la référence nulle est retournée.

Exemple: Les opérateurs is et as

```
...



   // Ici, le transtypage peut se faire.
   if( RefA is C ){ RefC = (C)RefA; // utilise RefC... }
   // équivalent à
   RefC = RefA as C;
   if( RefC != null ){// utilise RefC...}
```

Les classes conteneurs

- **Permet d'implémenter des tableaux dynamiques, des piles, des listes chaînées ….**
- **Peut contenir tout objet dérivant de la classe Object**
- **Regroupées dans l'espace System.Collections**
  - ArrayList tableau dynamique d'objets
  - Stack pile de type LIFO
  - Hashtable dictionnaire d'objets
    - public Hashtable ()
    - public virtual void Add (Object *key*, Object *value*)
    - public virtual bool ContainsKey (Object *key*)
    - public virtual bool ContainsValue (Object *value*)

Exemple d'utilisation classe ArrayList

```
using System;
using System.Collections;
namespace Transtype
{
     class Address
     {
          string nom;
          string ville;

          public Address(string name, string city)
          {
               nom = name;
               ville = city;
          }
          public void Afficher()
          {
               Console.WriteLine("{0} {1}",nom, ville);
          }
     }
     class Prog
     {
          static void Main()
          {
               ArrayList AddressList = new ArrayList();
               AddressList.Add(new Address("Joe", "Nice")); // Transtypage implicite
               AddressList.Add(new Address("Jack", "Menton"));
               AddressList.Add(new Address("John", "Cannes"));
               for (int i=0; i<AddressList.Count; i++)
               {
                        Address adr = (Address)AddressList[i]; // Transtypage explicite
                        adr.Afficher();
               }
          }
     }
}
```

```
public virtual int Add(
    object value
);
```

```
public virtual object this[
    int index
] {get; set;}
```

Exemple d'utilisation classe ArrayList

```
namespace Transtype
{
    class Address
    {
        string nom;
        string ville;

        public Address(string name, string city)
        {
            nom = name;
            ville = city;
        }

         public override string ToString()
         {
             return nom + " " + ville;
         }
    }
    class Prog
    {
        static void Main()
        {
            ArrayList AddressList = new ArrayList();
            AddressList.Add(new Address("Joe", "Nice")); // Transtypage implicite
            AddressList.Add(new Address("Jack", "Menton"));
            AddressList.Add(new Address("John", "Cannes"));
            IEnumerator enu = AddressList.GetEnumerator();
            while (enu.MoveNext())
                Console.WriteLine(enu.Current);
            }
    }
}
```

Exemple d'utilisation classe ArrayList
Tri

```
namespace Tri
{

    class Prog
    {
        static void Main()
        {
             ArrayList a = new ArrayList();
             a.Add(6);
             a.Add(2);
             a.Add(1);
             a.Add(9);
             a.Sort(); //uttilise la méthode CompareTo redéfinie

       //ArrayList.Sort(IComparer comparer) pour plusieurs critères de tri créer une classe dérivée
    de IComparer par critère

        }
    }
}
```

Le Type Int32 implémente l'interface IComparable
Possédant la méthode int CompareTo(object);