Plan

- Introduction
- Types et opérations de base
- Les instructions de base
- Types valeur et types référence
- **Gestion des classes et des objets**
- Composition, Héritage, polymorphisme
- Abstraction, Interfaces
- Les classes du Framework.NET
- Mécanismes utilisés en C#

Définitions

- **Classe**
  - C'est un modèle à partir duquel on peut créer des objets. Une classe définit les caractéristiques d'un objet:
    - les champs que l'objet contient
    - les méthodes décrivant le comportement de l'objet
  - Ces caractéristiques spécifient comment les autres objets peuvent accéder et travailler avec les données de l'objet considéré.

- **Objet**
  - Un objet est une instance d'une classe. Il est créé à partir du "modèle" qu'est la classe.

Espaces de noms

- ***Namespaces (espaces de noms)***
  - Les namespaces servent à organiser les classes dans une hiérarchie logique, un peu à la manière des packages de Java
  - Les namespaces ne sont pas seulement utiles dans une optique d'organisation interne, mais aussi dans le but d'éviter des collisions de noms.
  - On peut donc dire qu'un namespace est un *système d'organisation permettant d'identifier des groupes de classes*.

```
namespace Exemple1
 {
    public class Client{ …..}
 }
```

Exemple1.Client

```
namespace Exemple2
 {
      public class Client{ …..}
 }
```

Exemple2.Client

Accessibilité

- **public**
  - Accès illimité. N'importe quelle autre classe peut accéder à un membre public
- **private**
  - Accès limité à la classe contenante. Seule la classe contenant le membre peut accéder au membre private.
- **internal**
  - Accès limité à cet assemblage (programme). Les classes se trouvant dans le même assemblage peuvent accéder au membre.
- **protected**
  - Accès limité à la classe contenante et aux classes dérivées de cette classe.
- **protected internal**
  - Accès limité à la classe contenante, aux classes dérivées et aux classes se trouvant dans le même assemblage (*ensemble des fichiers constituant un programme)*

Exemple d'une classe

```
class Pers                    // déclaration d'une classe appelée Pers
{
    string nom;
    int age;
    public void afficher()
        {……}


}
```

Pers est un nouvel identificateur de type qui permet de déclarer des objets de ce type, un objet de la classe Pers est appelé instance de la classe Pers

instanciation : Pers p;

p= new Pers();

Les membres d'une classe

- **Les champs**
  - un champs d'une classe définit une donnée (type valeur) ou une référence vers une donnée (type référence) qui existe pour toute instance de la classe.
- **Les méthodes : peut être vu comme un traitement sur les données d'un objet. Les méthodes peuvent être surchargées.**
  - Méthodes particulières : Constructeurs et destructeurs
- **Les propriétés**
  - couple de méthode pour intercepter les accès (lecture/écriture) aux données d'un objet avec la syntaxe d'accès aux champs.
- **Les indexeurs**
  - propriété spéciale utilisant la syntaxe d'accès à un tableau [ ].
- **Les événements**
  - facilite l'implémentation de la notion de programmation événementielle.

mot-clé this

- **Paramètre implicite des méthodes non statiques**

Exemple:

On désire implémenter dans la classe Pers, une méthode permettant de renvoyer entre deux personnes, la plus âgée

```
class Pers
{
    .......
     public Pers PlusVieux(Pers p)
    {
        if (m_Age >p. m_Age)
            return this;
        else return p;
    }
}
```

Constructeurs et destructeur

- **Constructeurs**
  - Pour initialiser les champs
  - Syntaxe : <classe>(<paramètres>) Pers(string nom,int age)
  - Constructeur sans paramètre implicite si on n'en définit pas

- **Destructeur**
  - Pour libérer explicitement les ressources (implicite avec Ramasse-miettes)
  - Syntaxe : ~<classe>()
  - Appelé automatiquement à la fin du bloc où l'objet est créé

Constructeurs et destructeur

```
public class Article
{
    private int prix;
    public Article(int prix)    { this.prix = prix; }          // ctor 1
    public Article()            { this.prix = 0; }             // ctor 2 surharge
    ~Article()                  { /* rien à désallouer!*/ }    // dtor
}
```

Propriétés

- Accesseurs spéciaux de type méthodes

```
public class UneClasse
{
    // Un champ privé de type int.
    private int champPrivé = 10;

    // Une propriété publique de type int.
    public int ChampPrivé
    {
        get{  return champPrivé;}
        set{ champPrivé = value;}
    }
    static void Main()
    {
        UneClasse A = new UneClasse();
        A. ChamPrivé = 56;          // L'accesseur set est appelé.
        int i = A. ChampPrivé;     // L'accesseur get est appelé.
    }
}
```

Indexeurs

- **Permet de considérer un objet collection comme un tableau**
  - L'opérateur [ ] peut être appliqué à l'objet
  - Plusieurs indexeurs possible avec des types d'index différents

```
class Objet
{
……..
}
class ObjetCollection
{
   private Objet[] objetList;
……..……..
   public Objet this[int index]
           {
                   get {return objetList[index];}
                   set {objetList[index] = value;}

           }
……..……..
```

indexeurs

```
using System;

        class Address
        {
                private string nom;
                private string ville;
                public Address(string name, string city)
                {
                        nom = name;
                        ville = city;
                }
                public void afficher()
                {
                        Console.WriteLine("Nom : {0} Ville : {1}", nom, ville);
                }
        }
```

indexeurs

```
class AddressBook
        {
                private Address[] addressList;
                private int taille;
                private int nbElt;

                public AddressBook(int size)
                {
                        addressList = new Address[size];
                        taille=size;
                        nbElt=0;
                }
                public void add(Address ad)
                {
                        if(nbElt< taille)
                        {
                          addressList[nbElt]=ad;
                          nbElt++;
                        }
                }
                public Address this[int index]
                {
                        get {return addressList[index];}
                        set {addressList[index] = value;nbElt++;}

                }

        }
```

indexeurs

```
class Prog
        {
                static void Main()
                {
                        AddressBook book = new AddressBook(10);
                        book[0]=new Address("Lecat","Vence");
                        book.Add(new Address("Dupont", "Nice"));

                        book[0].afficher();
                        book[1].afficher();

                }
        }
```

Les membres statiques

- **Les champs, les propriétés, les méthodes (y compris les constructeurs mais pas le destructeur) et les événements d'une classe ont la possibilité d'être déclaré avec le mot-clé static.**
- **Ce mot-clé signifie que le membre appartient à la classe, et non aux objets, comme c'est le cas des membres non statiques.**
  - Un membre statique a un niveau de visibilité.
  - Une méthode statique n'a pas accès aux méthodes, champs, propriétés et événements non statiques de la classe.
  - Une méthode statique n'est pas accessible à partir d'une référence
  - Une méthode statique est accessible dans toutes les méthodes (statiques ou non) de la classe par son nom, et à l'extérieur de la classe (si son niveau de visibilité le permet) par la syntaxe :
    - NomDeLaClasse.NomDeLaMéthodeStatique()

Les membres statiques

```
public class Article
{
    static int nbArticles;              // Un champ statique.
    static Article()                    // Le constructeur statique appelé
                                        //automatiquement à la 1ere construction
    { nbArticles = 0;}

    int prix = 0;                       // Un champ privé non statique.

    public Article( int prix )          // ctor non statique
    {
        this.prix = prix;
        nbArticles++;
    }
    ~Article()                          // dtor
    {  nbArticles--;}
}
```

Surcharge des opérateurs

- **Objectif : définir des méthodes appelées en utilisant les opérateurs classiques**
  - écrire a+b au lieu de Somme(a, b)
- **Opérateurs concernés**
  - Unaires : +, -, !, ++, --, true, false
  - Binaires : +, -, *, /, %, &, |, ^, <<, >>, ==, !=, >, <, >=, <=
- **Syntaxe**
  - public static <return type> operator <operateur>(<parameters>) {…}