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#**

Construction de nouvelles classes

- **Par composition**
  - **a** est un champ de **ClB**

```
class ClA
{
    …
}

class ClB
{
    ClA a;
    …
}
```

- Par héritage
  - **ClB** hérite de toutes les propriétés (champs, méthodes, etc…) de **ClA**

```
class ClA
{
    …
}

class ClB : ClA
{
    …
}
```

Exemple : L'héritage

```
using System;
class Employé
{
    protected string nom;
    protected Employé(string nom)
    {this.nom = nom;}
}
class Technicien : Employé
{
    private string spécialité;
    public Technicien(string nom, string spécialité): base(nom)
    {this.spécialité = spécialité;}
}
class Prog
{
    public static void Main()
    {
        Employé E = new Technicien("Roger LaFleur","Electricien");
    }
}
```

Héritage et constructeur

- **Les constructeurs ne sont jamais hérités.**
- **Pour préparer les données membres héritées, le constructeur de la classe de base doit être invoqué explicitement ou implicitement (constructeur par défaut).**
- **Dans tout les cas le constructeur de la classe de base est invoqué avant celui de la classe dérivée.**
- **Dans une hiérarchie de classes les constructeurs seront appelées dans l'ordre descendant**

Surcharge des méthodes

- **On peut redéfinir les méthodes des classes ancêtres, à condition de respecter la même déclaration**
  - Respecter la même syntaxe
  - Override,new

Méthodes virtuelles et polymorphisme

- **Le polymorphisme consiste à utiliser le même nom de fonction (surcharge) avec un code différent**
  - Si une méthode est déclarée comme virtual, la bonne surcharge est automatiquement appelée à l'exécution du programme, en fonction de l'objet courant

- **Les fonctions virtuelles permettent des liaisons dynamiques c'est à dire le choix de la fonction à appeler en cours d'exécution**

Exemple : Méthodes virtuelles et polymorphisme

```
using System;
public class Employé {
   // m_Nom peut être accédé dans les méthodes des classes dérivées.
   protected string m_Nom;
   public Employé(string nom) {m_Nom = nom;}
   public virtual void GetDescription()
   {Console.Write("Nom: {0}",m_Nom);}
}
class Technicien : Employé{  // Technicien hérite de Employé.
   public Technicien(string nom):base(nom) {}
   public override void GetDescription(){
        // Appel de la méthode GetDescription() de Employé.
        base.GetDescription();
        Console.Write("  Fonction: Technicien\n");}
}
class Secrétaire : Employé{  // Secrétaire hérite de Employé.
   public Secrétaire(string nom):base(nom) {}
   public override void GetDescription(){
        // Appel de la méthode GetDescription() de Employé.
        base.GetDescription();
        Console.Write("  Fonction: Secrétaire\n");}
}
```

Exemple : Méthodes virtuelles et polymorphisme

```
...
class Prog
{
   static void Main(string[] args)
   {
        Employé [] tab = new Employé[3];
        tab[0] = new Technicien("Roger");
        tab[1] = new Secrétaire("Lise");
        tab[2] = new Technicien("Raymond");
        foreach( Employé e in tab )
               e.GetDescription();
   }
}
```

Ce programme affiche:

```
Nom: Roger  Fonction: Technicien
Nom: Lise  Fonction: Secretaire
Nom: Raymond  Fonction: Technicien
```