C# Synthèse

Les points clefs

Les génériques

- Un objectif: production des structures informatiques génériques réutilisatisables pour différents types d'objets
  - Exemple typique: les conteneurs d'objets
    - Listes, piles

```
public class Stack<T>
{
int position;
T[] data = new T[100];
public void Push (T obj) { data[position++] = obj; }
public T Pop() { return data[--position]; }
}
```

```
Stack<int> stack = new Stack<int>();
Stack<Ville> stack2 = new Stack<Ville>();
stack.Push(5);
stack.Push(10);
int x = stack.Pop(); // x is 10
int y = stack.Pop();
Stack2.Push(new Ville(« Toulon »));
```

- Application
  - Exemple du poly:Les dictionnaires
    - Type générique ->
    - collection d'objets de type clé/ Valeur
      - public struct KeyValuePair<TKey, TValue>

```
public class Dictionary<TKey, TValue> :
IDictionary<TKey, TValue>,
ICollection<KeyValuePair<TKey, TValue>>,
IEnumerable<KeyValuePair<TKey, TValue>>,
IDictionary, ICollection, IEnumerable,
ISerializable, IDeserializationCallback
```

```
// création du dictionnaire
Dictionary<UInt32,Etudiant> dicLPSE=new Dictionary<UInt32,Etudiant> ();
```

```
// ajout des entrees
dicLPSE.Add(100213,new Etudiant("Oscar","Amel",21,100213));
```

```
//Methode 1:parcours du dictionnaire pour chaque couple cle/valeur

Foreach (KeyValuePair<UInt32, Etudiant> kv in dicLPSE)
Console.WriteLine(kv.Key + "; " + kv.Value);
```

Délégations et délégués

- Objectif: Création d'objets se comportant comme une fonction
  - Notion de Foncteur que l'on retrouve en C++
    - Connexes aux pointeurs de fonctions

- Une délagation
  - Classe d'objet faisant référence à une ou plusieurs méthodes d'instances ou de classes ayant un prototype spécifique (contrat d'appel)
  - Les délégations sont du type delagate
    - delegate double T(int n);
  - La délégation peut être vu comme un contrat d'appel
    - Respect des prototypes des méthodes appelées
      - Un argument en sortie du type double
      - Un argument en entrée de type int
- Le délégué
  - Instance d'une délagation
  - Le délégué est une référence vers ou un plusieurs méthodes
  - L'invocation du délégué provoque l'appel des méthodes ciblées

```
// Etape1: Contrat de delegation
delegate double T(int n);
// Etape2: Référence vers le délégué
T de;
```

```
// Etape3 :Création du délégué
de = new T(f); //  fait référence à f

Remarque: ALTERNATIVE avec étapes 2 et 3 réunies
T de=f;
```

Délégations et délégués

- Exemples

```
using System;
class Program
{
static double f(int n) {return n/2.0;}
static double g(int n) {return n*2;}
delegate double T(int n); // délégation
static void Main()
{
T de; // déclaration de variable
de = new T(f); // de fait référence à f
double d = de(5); // exécution de f, d prenant la valeur 2.5
de = g); // de fait maintenant référence à g
d = de(100); // exécution de g, d prenant la valeur 200}}
```

- Délégations prédéfinies dans System
  - Répondent aux problématiques usuelles
    - Transformer valeur, comparer 2 valeurs entre elles, valeur correspond à critére oui/non

```
namespace System{
    public delegate void Action<T> ( T obj);
    public delegate void Predicate<T> ( T obj);
    public delegate void Converter<T,U> ( T obj);
    public delegate int Comparison<T> ( T obj,T y);
    .......
}
```

Délégations et délégués

- Mise en application
  - Comprendre les prototypes dans msdn
  - Problématique: je veux trier une liste suivants mes propres critères
  - Résultat de ma recherche sur la classe List<T>
    - public void Sort(Comparison<T> comparison)

Interprétation: ma méthode de tri devra respecter le contrat imposé par la délégation `int Comparison<in T> ( T obj,T y);`

- 2 paramètres (spécifiés in car en entrée) de type générique T
- Un entier en sortie
  - Si x>y alors on renvoie 1
  - Si x=y alors 0
  - Si x<1 alors -1

Délégations et délégués

```
class Rectangle
    {
      private uint _aire;

      public uint Longueur {get; set;}
      public uint Largeur {get; set;}
      public uint Aire { get { return (_aire); } }

      public Rectangle(uint longueur, uint largeur)
      {
        Longueur = longueur;
        Largeur = largeur;
       _aire = Longueur * Largeur;
        }

      public static int CompareRectangle(Rectangle r1, Rectangle r2)
         {   int valcomp=r1.Aire.CompareTo(r2.Aire);
             if (valcomp==0)
                 if (r1.Longueur>r2.Longueur) valcomp=1;
             return valcomp;
         }

    public override string ToString()
        {
         return ("Long:"+Longueur+" larg:"+Largeur+" Aire:"+Aire);
        }
    }
```

Délégations et délégués

```
class Program
    {
        static void Main(string[] args)
        {
            List<Rectangle> maList = new List<Rectangle>();
            maList.Add(new Rectangle(10, 20));
            maList.Add(new Rectangle(10, 10));
            maList.Add(new Rectangle(1, 2));
            maList.Add(new Rectangle(20, 10));

     // Le délégué Comparison pointera vers Rectangle.CompareRectangle
    // lors du tri Sort appelera au travers du délégué ma fonction de comparaison!
            maList.Sort(Rectangle.CompareRectangle);
            foreach (Rectangle rect in maList) Console.WriteLine(rect);

    // exemple de methode anonyme ( delégué sur methode non nommée)
            maList.Sort(
                              delegate(Rectangle r1,Rectangle r2)
                              {return(r1.Aire.CompareTo(r2.Aire)); }
            );

    // même exemple avec expression lambda
            maList.Sort((r1, r2) => r1.Aire.CompareTo(r2.Aire));
            foreach (Rectangle rect in maList) Console.WriteLine(rect);
            Console.ReadKey();
        }
    }
```

Aire étant un uint il n'est pas nécessaire de réecrire la méthode CompareTo

Remarque: Il est possible d'aborder le problème par le biais des Interfaces (pattern strategy !)

```
public void Sort(IComparer<T> comparer)
```

Méthodes d'extensions

- Objectif: permet d'ajouter des méthodes à des classes dont nne dispose pas du code source et/ou qui sont qualifiées de sealed
  - Pas de modification du type d'origine
  - Elle sont un type particulier de méthodes statiques
  - La syntaxe: public static zzzz *NOMMETHODE*(this,xx,yy)

```
static class SupDateTime
{
public static bool Antérieur(this DateTime d)
{
return d < DateTime.Now;
}
}
.....
DateTime dt = new DateTime(1789, 7, 14);
bool res = dt.Anterieur();
```

dt.Anterieur( );

```
arg0.Method (arg1, arg2, ...);              // Extension method call
StaticClass.Method (arg0, arg1, arg2, ...); // Static method call
```

SupDateTime.Anterieur(dt,);

La plupart des collections mettent à la disposition de l'utilisateur des méthodes d'extensions pour effectuer des traitements évolués
Les méthodes d'extensions sont très utilisées en conjonction avec des requêtes LINQ

Méthodes d'extensions

- Application
  - Je cherche à effectuer la moyenne des aires de mes rectangles
  - Que dis MSDN?

Renvoie un double

Méthode statique s'appliquant sur un IEnumerable de type générique Tsource (Tsource=Rectangle pour nous)

```
public static double Average<TSource>(
        this IEnumerable<TSource> source,
        Func<TSource, long> selector
)
```

Délégation Func de signature:
1 paramètre en entrée de type Tsource
1 paramètre en sortie de type long

La méthode cible respecte le contrat OK

Mise en oeuvre

Création du délégué ciblant la méthode OK

```
class Rectangle
    {
.....
  public static Int64 AverageRectangle(Rectangle r1)
        {
            Int64 result = r1.Aire;
            return (result);
        }
    }
```

```
//exemple de delegation de type Func<xx,yy>
Func<Rectangle, Int64> selector = Rectangle.AverageRectangle;
double result = maList.Average(selector);
```

List<Rectancle> est aussi un IEnumerable<Rectangle> (OK MSDN)

Comment ça marche?

La méthode Average va parcourir la liste .A chaque itération le délégué qui a été crée est invoqué. Votre méthode est donc appelée au travers du délégué. A chaque appel on transmet un Tsource (Rectangle) et on retourne un Int64. Ensuite la méthode Average se charge de la somme des Int64 retourné puis fait la division finale pour rerourner un double (la moyenne)

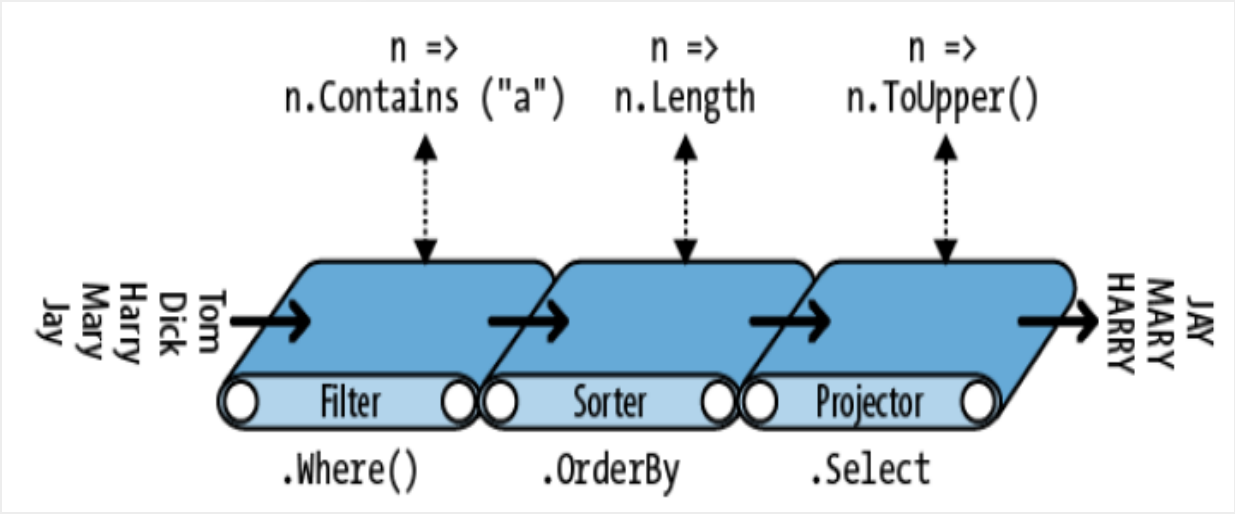
```
// exemple avec methodeanonyme: version délégué
double result2 = maList.Average(delegate(Rectangle r) { return (r.Aire); });
// exemple avec methodeanonyme: version lambda expression
double result3 = maList.Average((Rectangle r1) => r1.Aire);
Console.WriteLine("Moyenne des aires {0} {1}{2}",result,result2,result3);

// KO ambiguité double result4 =
maList.Average(Rectangle.AverageRectangle);
```

Méthodes d'extensions

- Application
  - Requete LINQ

```
string[] names = { "Tom", "Dick", "Harry", "Mary", "Jay" };
            IEnumerable<string> filtered = names.Where(n => n.Contains("a"));
            IEnumerable<string> sorted = filtered.OrderBy(n => n.Length);
            IEnumerable<string> finalQuery = sorted.Select(n => n.ToUpper());
            foreach (string name in finalQuery)
                Console.WriteLine(name);
```



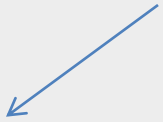
Events

- Delegation standart fournie

```
public delegate void EventHandler<TEventArgs>(object source, TEventArgs e)
```

- Création d'événements personnalisés

```
public event EventHandler<ThresholdReachedEventArgs>
ThresholdReached;
```



```
public class ThresholdReachedEventArgs :
EventArgs
    {
        public int Threshold { get; set;
}
        public DateTime TimeReached {
get; set; }
    }
```

Events

- La classe cliente s'enregistre

```
static void Main(string[] args)
        {
            Counter c = new Counter(4);
            c.HandlerThresholdReached += c_ThresholdReached;

            Console.WriteLine("press 'a' key to increase total");
```

Lorsque la valeur limite sera atteinte l'objet c invoquera le délégué HandlerThresholdReached qui "pointe" vers c_ThresholdReached

```
static void c_ThresholdReached(object sender, ThresholdReachedEventArgs e)
{
Console.WriteLine("The threshold of {0} was reached at {1}.", e.Threshold,
e.TimeReached);
Environment.Exit(0);
 }
```

c_ThresholdReached respecte le contrat d'appel imposé par la délégation standart EventHandler

Events

- Signalisation de l'évévement

Test de la valeur
Si valeur> limite
Alors
Création de l'argument evénément
Invocation délégué au travers de OnThresholdReached

```
public void Add(int x)
        {
            total += x;
            if (total >= threshold)
            {
                ThresholdReachedEventArgs args = new
ThresholdReachedEventArgs();
                args.Threshold = threshold;
                args.TimeReached = DateTime.Now;
                OnThresholdReached(args);
            }
        }

        protected virtual void OnThresholdReached(ThresholdReachedEventArgs e)
        {
            if (handlerThresholdReached!= null)
            {
                handlerThresholdReached (this, e);
            }
        }
```

Invocation du délégué encapsulé dans la méthode **OnThresholdReached** (pattern classique)