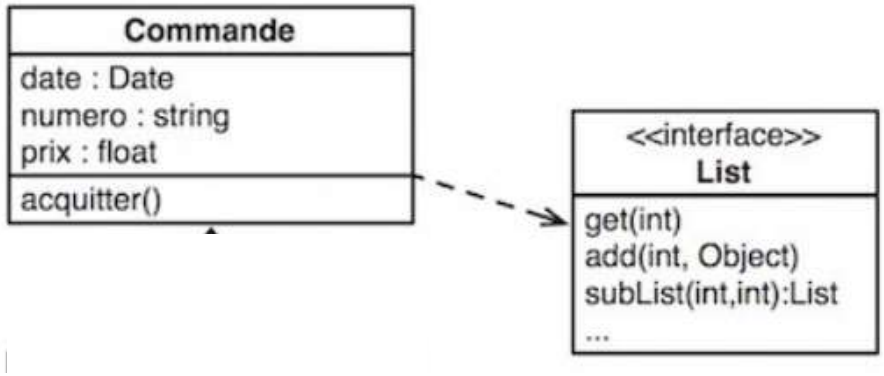
Rappel Interface

- Contrats dont toute classe peut y souscrire
- Les classes qui en « hérite » doivent l'implémenter en respectant la signature
- Une classe peut hériter de multiples interfaces

Interfaces en UML



Délégation

Aline Ellul

POO A 3

Qu’est qu’un delegate ?

- Un délégué permet de représenter des méthodes d’un même type càd ayant :
  - Le même type de valeur de retour
  - Le même nombre de paramètres
  - Les mêmes types pour chaque paramètre
- Grâce aux délégués, il est possible de déclarer et d’utiliser des variables faisant référence à une méthode (du même type que le délégué)
- On peut alors appeler la méthode référencée en utilisant ces variables sans connaître la méthode réellement appelée

Qu’est qu’un delegate ?

- Le type delegate est déclaré dans une classe par le mot clef delegate
- Exemple :

```
delegate int Operation(int valeur1, int valeur2);
```

Déclaration des méthodes et application

```
delegate int Operation(int valeur1, int valeur2);

static int Addition(int v1, int v2)
{
    return v1 + v2;
}
static int Soustraction (int v1, int v2)
{
    return v1 - v2;
}

static int AppliquerOperation(Operation o, int v1, int v2)
{
    return o(v1, v2);
}
```

Déclaration de 2 méthodes du même type que le delegate Operation

Application de l'opération avec les opérandes associées

Appel de la méthode AppliquerOperation

Programme Principal

```
int total0  = AppliquerOperation(Addition, 10, 5);

int total1 = AppliquerOperation(Soustraction, 10, 5);
```

Méthodes Anonymes

- Les méthodes anonymes sont des méthodes sans nom qui sont créées directement dans le code d'une méthode.
- Elles sont référencées et appelables grâce aux variables de type délégué
- Les méthodes anonymes doivent prendre en paramètres les mêmes paramètres que le délégué associé
- Le type de retour est déterminé par le compilateur grâce aux return contenus dans la méthode anonyme

Méthodes anonymes

```
int total2;
Operation o;
int autrevaleur = 20;
//Création d'un méthode anonyme
o = delegate (int v1, int v2) { return autrevaleur * v1 * v2; };
total2 = AppliquerOperation(o, 10, 10);
```

Lambda expression

- Une lambda est une autre façon d'exprimer un délégué anonyme plus concise
- Le mot clef delegate est remplacé par =>

- Il n'est pas utile de définir les types des paramètres si déduisibles par le compilateur

Exemple

Dans la classe Program

```
public delegate bool Operation(int valeur);

public static int GetPremier(int[] t, Operation o)
{
    int retour = -1;
    for(int i=0;i<t.Length && retour == -1;i++)
    {
        if (o(t[i])) retour = t[i];

    }
    return retour;
}
```

Programme Principal

```
int[] t = { 1, 2, 89, 65 };
int valeur = GetPremier(t, e => e < 10);
```

Rappel

- Pour utiliser la méthode Sort() sur une collection afin de trier cette collection sur le critère par défaut, il faut implémenter la méthode CompareTo de l'interface IComparable.

```
class Salarie : IComparable
{
    int numero;
    string nom;
    string prenom;
    double salaire;

    public Salarie(int numero, string nom, string prenom, double salaire)...
    public double Salaire...
    public string Nom...
    public int CompareTo(object obj)
    {
        return this.nom.CompareTo(((Salarie)(obj)).nom);
    }
    public override string ToString()
    {
        return "Nom = " + Nom;
    }
}
```

```
Salarie s = new Salarie(10, "tuto", "titi", 20000);
Salarie s1 = new Salarie(10, "toto", "titi", 10000);
Salarie s2 = new Salarie(10, "titi", "titi", 20000);

List<Salarie> sf = new List<Salarie>();
sf.Add(s);
sf.Add(s1);
sf.Add(s2);

foreach (Salarie ss in sf)
    Console.WriteLine(ss);
sf.Sort();
```

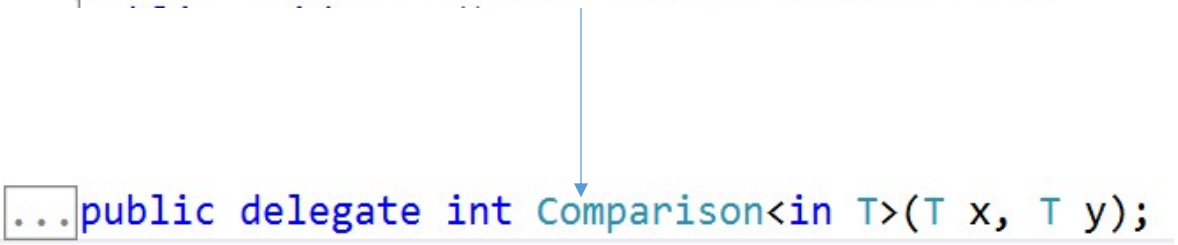
Quid d’un tri selon un autre critère ?

```
public class List<T> : IList<T>, ICollection<T>, IEnumerable<T>, IEnumerable, IList,

        ...public void Sort(int index, int count, IComparer<T> comparer);
        ...public void Sort(Comparison<T> comparison);
        ...public void Sort();
        ...public void Sort(IComparer<T> comparer);
```

Sort à la carte sur une collection

```
public void Sort(Comparison<T> comparison);
```



```
...public delegate int Comparison<in T>(T x, T y);
```

```
   Comparison<Salarie> funct0 = delegate (Salarie sa1, Salarie sa2) { return
                                                    sa1.Salaire.CompareTo(sa2.Salaire); };
   Comparison<Salarie> funct1 = new Comparison<Salarie>(delegate (Salarie sa1, Salarie sa2) {
                                                    return sa1.Nom.CompareTo(sa2.Nom); });
   Comparison<Salarie> funct2 = new Comparison<Salarie>(delegate (Salarie sa1, Salarie sa2) {
                                                    return sa1.Numero.CompareTo(sa2.Numero);
});

   sf.Sort(funct0);
```

Find sur une collection

```
T Find(Predicate<T> match);
```



```
public delegate bool Predicate<in T>(T obj);
```

```
Console.WriteLine(sf.Find(x => x.Salaire > 10));
Console.WriteLine(sf.Find(delegate(Salarie x) { return x.Salaire > 10; }));
```