Interface Homme Machine

Nadine CULLOT (CM/TD/TP)
nadine.cullot@u-bourgogne.fr

Stéphanie BRICQ (TD/TP)
stephanie.bricq@u-bourgogne.fr

Volume horaire : 14 CM 18hTD et 16hTP

Objectifs du cours

- Eléments de base du langage C#
- La programmation orientée objet en langage C#.
- Etude du développement d'applications avec une interface graphique (de type Windows) en Visual C# (Visual Studio 2013 – Ultimate)
  - Etude des composants de Visual Studio .net
    - System.Windows.Forms
  - Connexion à une base de données
  - Développement d'applications à l'aide de l'assistant

Modalités de Contrôle des Connaissances

- Un partiel : note de contrôle continu (CC)
- Un projet : note de TP (TP)
- Un examen final : note de contrôle écrit (CE)
- Note module = (0,5*CC + 0,5*TP + CE)/2
- Projet réalisé en binôme *ou trinôme*
  - Exposé oral, Rapport Ecrit, Démonstration

Plan CM1

- Architecture .net
- Les éléments de base du C#
  - types, instructions, E/S, fonctions, pointeurs
  - modes d'exécution
- La POO en C#
  - Les classes
  - Les surcharges de méthodes et d'opérateurs

Introduction à l'architecture .net

Concepteur et architecte principal chez Microsoft :

- Anders Hejlsberg né au Danemark en 1961
- 1983 : création de la société Borland et de son produit phare Turbo Pascal
- 1990 : apparition de Delphi, environnement de programmation sous Windows, en turbo pascal
- 1996 : Microsoft embauche Anders Hejlsberg qui conçoit avec son équipe :
  - WFC (Windows Foundation Classes), classes java pour interface Windows
  - Il n'existait à l'époque que les classes AWT (Abstract Windows Toolkit) de Sun
  - Les relations avec la communauté "officielle" de java sont "difficiles" car les classes WFC étaient propres à Windows, ce qui est contraire à la philosophie java.

Introduction à l'architecture .net

- 2000 : Microsoft annonce
  - l'architecture .net et le langage C# dont Anders Hejlsberg est le principal concepteur
  - Et la disparition du langage J++ de sa gamme
- 2001 : Visual J++ réapparaît sous le nom J#
- 2002 : L'outil Visual Studio appelé Visual Studio .net est conçu :
  - Pour rendre le développement d'applications windows et web plus facile
    - Une nouvelle architecture est mise au point
    - Le langage C# est créé
    - Le langage C++ est toujours utilisé mais de façon moindre
    - Les applications doivent être écrites en C#, J# ou VB.net (version Orientée Objet des 1ères versions de Visual Basic).

Introduction à l'architecture .net

- Les développeurs utilisent un ou plusieurs langages compatibles .net (Microsoft ou autre)
  - Tous les langages ont une orientation objet et doivent respecter des règles (types utilisés, contenu et format des fichiers générés, etc.)
  - Ces règles forment le CLS (Common Language Specification) – normalisation au niveau mondial
  - Ces langages permettent d'écrire :
    - Des programmes console
    - Des applications pour interface graphique Windows
    - Des applications pour le Web (côté serveur) – ASP.NET

Introduction à l'architecture .net

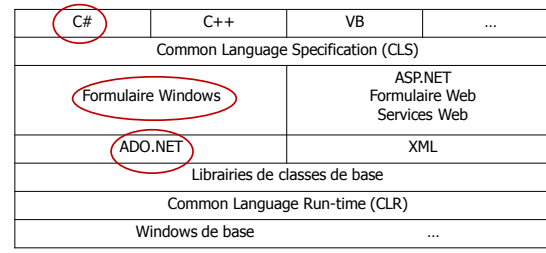
- L'outil de développement permet également de :
  - Créer des services Web qui reposent sur des protocoles standard : SOAP, XML, et HTTP
  - Manipuler des données provenant de bases de données avec ADO.NET
- Toutes les applications ont accès à un nombre très important de classes de base mais des composants extérieurs peuvent être ajoutés.
- Tous les compilateurs génèrent un code de base appelé IL (Intermediate Language).

Introduction à l'architecture .net

- Les programmes s'exécutent sous le contrôle strict du run-time .NET (appelé aussi framework).
- Le CLR (Common Language Run-time) et le C# font l'objet d'une normalisation internationale qui doit permettre de l'implémenter sous d'autres systèmes d'exploitation.
- Le CLR est une couche logicielle au dessus du système d'exploitation. Il agit comme une machine virtuelle bien que Microsoft rejette ce terme souvent associé à java.

Introduction à l'architecture .net

Aperçu général de l'architecture .NET



Les éléments de base de C#

- La langage C# est proche du langage java, mais conserve certains aspects de C et C++.
- La syntaxe du langage est proche de celle de java.
- Les types de base :
  - Entiers : byte [0,255], sbyte [-128,127], short, ushort, int, uint, long, ulong *(les entiers non signés ne sont pas conformes au CLS)*
  - Booléens : bool (true; false)
  - Réels : float, double, decimal
  - Caractères : char
  - Chaînes de caractères : string (objet de la classe String de l'espace de nom System; string désigne System.String)

Les éléments de base de C#

- C# garde la notion de structure de C/C++, mais les champs doivent être qualifiés de public pour être accessibles
  - struct personne {
    - public string nom;
    - public string prenom; }
  - personne p= new personne();
    - approche plus près de la notion de classe qu'en C/C++

Les éléments de base de C#

- Les types énumérés : le type enum

```
enum EtatCivil  {Célibataire, Marié, Divorcé, Veuf};

EtatCivil ec; // variable de type EtatCivil
ec=EtatCivil.Marié;

switch (ec) {
 case EtatCivil.Célibataire : Console.WriteLine("Célibataire");break;
... }
```

  - En interne, la valeur 0 est associée à Célibataire, 1 à Marié, etc.

```
ec=EtatCivil.Célibataire;
ec++; // passe à la valeur Marié
```

  - Il est possible d'associer des valeurs particulières, par exemple

```
enum EtatCivil  {Célibataire=10, Marié=20, Divorcé,=30 Veuf=40};
```


Les éléments de base de C#

- Les variables de type énumération peuvent être considérées comme des objets de la classe System.Enum
- Les méthodes (statiques) de cette classe peuvent être utiles par exemple
  - Pour convertir une chaîne en l'une des valeurs du type énuméré

```
string s="Marié"; EtatCivil ec;
ec=(EtatCivil) Enum. Parse(typeof(EtatCivil),s,false);
```

  - Pour récupérer la chaîne qui correspond à la valeur d'énumération

```
EtatCivil ec=EtatCivil.Marié; string s;
s= Enum .Format(typeof(EtatCivil),ec, "g");
// "g" signifie qu'on récupère le nom, sinon "d" pour récupérer la valeur associée
```


Les éléments de base de C#

- Il est possible d'afficher les différentes valeurs d'un type énuméré en utilisant la méthode statique GetNames qui retourne un tableau des chaînes des noms du type énuméré

```
foreach (string s in Enum.GetNames(typeof(EtatCivil)))
    Console.WriteLine(s);
```

- La méthode ToString peut être appliquée à un objet de type énuméré.

```
EtatCivil ec=EtatCivil.Marié;
string s = ec.ToString();
string s2= Enum.GetName(typeof(EtatCivil),ec); // idem
```


Les éléments de base de C#

- Les tableaux
  - Tableaux à 1 dimension

```
int [] tab1;
tab1 = new int [10];
```

  - Tableaux à plusieurs dimensions

```
int [,] tab2;
tab2 = new int [2,3]; //tableau à 2 lignes, 3 colonnes; tab[i,j]
```

```
int [,,] tab3;
tab3 = new int [2,3,4];
    //2 lignes, 3 colonnes et profondeur de 4 cellules; tab[i,j,k]
```

  - La libération de l'espace mémoire alloué est faite par le "ramasse-miettes"; différent de C/C++

Les éléments de base de C#

- Les opérateurs
  - Arithmétiques : +, -, * , / (la division entière est faite si le numérateur et le dénominateur sont entiers), % (modulo)
  - Incrémentation/décrémentation : ++, --
  - Binaires : &, |, ^ (xor), ~(négation bit à bit)
  - Décalages : << (à gauche), >> (à droite)
  - Logiques : && ||
  - Comparaison : ==, !=, <=, >=, <, >

Les éléments de base de C#

- Les instructions
  - Conditionnelle if (…) {} else {}
  - Boucles
    - while (…) { }
    - do { } while (…);
    - for (initialisation; condition; incrémentation) { }
      - for (int i=0; i<5; i++) { …}
    - foreach permet de parcourir un tableau et une collection
      - int [] ti= new int [] {5,10,15,20};
      - foreach (int i in ti) Console.WriteLine(i);

Les éléments de base de C#

- L' instruction
  - à choix multiple

```
switch(val)
  {
    case "France" : …; break;
    case "Italie" : … ; break;
    default : …;
    }
```

- Le sélecteur peut de type entier, réel, caractère, chaîne ou énuméré (plus riche qu'en C/C++).

Les éléments de base de C#

- Les opérations d'entrée/sortie
- Des fonctions statiques de la classe Console permettent d'effectuer des affichages à l'écran et des saisies au clavier.
- Pour l'affichage, les méthodes **Write** et **WriteLine** permettent un affichage sans ou avec un saut de ligne.
- Pour la saisie, la méthode **ReadLine**, permet de lire une chaîne de caractères au clavier.

Les éléments de base de C#

- **Affichages – Quelques exemples**
- using System;
- Console.WriteLine("Bonjour");
- int i=10; Console.WriteLine(i);
  - WriteLine accepte un entier (int, long, uint, ulong), un caractère (char), un réel (float, double), une chaîne (string, char[]) en paramètre
  - L'argument est converti en chaîne de caractères
- int j=15; Console.Writeline("i= {1} et j={0}", j, i);
  - Affiche : i=10 et j=15
  - j est affiché à l'emplacement {0} et i à l'emplacement {1}

Les éléments de base de C#

- **Saisie de données au clavier – Quelques exemples**
- La fonction ReadLine lit les caractères saisis jusqu'à validation (touche entrée), et renvoie une chaîne de caractères (string).

```
using System;
string s1 = Console.ReadLine();
int i= Int32.Parse(s1);
string s2 = Console.ReadLine();
double d= Double.Parse(s2);
```

- Les classes associées aux types de base sont :
  - byte -> Byte; short -> Int32, long -> Int64,
  - Float -> Single; double -> Double; decimal -> Decimal

Les éléments de base de C# Les fonctions

- Les arguments d'une fonction sont passés
  - Par valeur ou référence pour des arguments de type valeur (int, float, double, string, etc. et struct)
  - Par référence uniquement pour les tableaux ou les objets
- En C#, toutes les fonctions sont dépendantes d'une classe (méthodes)
  - Ce qui est différent de C++, mais similaire à java
  - On peut faire des méthodes « static »

Les éléments de base de C# Les fonctions

- Passage d'argument par valeur

```
using System;
class prog {
static void Main {
      int a=10;
      f(a);
      Console.WriteLine(a);
      }
static void f(int x)
{ Console.WriteLine(x); x++; }
}
```

- Le programme affiche 2 fois la valeur 10

Les éléments de base de C#
Les fonctions

- Passage d'argument par référence

```
using System;
class prog {
static void Main {
    int a=10;
    f(ref a);
    Console.WriteLine(a);
    }
static void f(ref int x)
{ Console.WriteLine(x); x++; }
}
```

- Le programme affiche 10 puis 11

Les éléments de base de C#
Les fonctions

- Passage d'un tableau d'argument

```
using System;
class prog {
static void Main {
    int [] tab = {10,20,30};
    f(tab);
    foreach ( int i in tab) Console.WriteLine(i);
    }
static void f(int [] t)
{ for (int i=0; i<t.Length; i++) t[i]++; }
}
```

- Le programme affiche : 11, 21 et 31

Les éléments de base de C#
Les fonctions

- Passage d'argument en out
- Il indique que la variable passée en argument est initialisée par la fonction appelée.

```
using System;
class prog {
static void Main {
    int a;  f(out a);
    Console.WriteLine(a);    }
static void f(out int x)
{  x= 10; }
}
```

- Le programme affiche : 10

Les éléments de base de C#
Les fonctions

- Passage d'argument out pour l'initialisation d'un tableau.

```
using System;
class prog {
static void Main {
    int [] tab;
    f(out tab); // out nécessaire car tab n'est pas encore alloué
    foreach ( int i in tab) Console.WriteLine(i);
    }
static void f(out int [] t )
{ t = new int [2]; t[0]=10; t[1]=20; }
}
```

- Le programme affiche : 10 et 20

Les éléments de base de C#
Les pointeurs

- C# garde les possibilités de C de manipuler des pointeurs
  - Mais la fonction/méthode dans laquelle on utilise un pointeur ou une adresse doit être qualifiée de « unsafe »
- Rappels
  - int i; // i est un entier
  - int *pi; // pi est un pointeur sur entier
  - pi = &i; // pi contient l'adresse de i
  - int j = *pi; // le contenu de pi est affecté à j

Les éléments de base de C#
Les pointeurs

- Utilisation de pointeurs

```
using System;
class prog {
unsafe static void Main {
    int i =10;  // initialisation non obligatoire ici
    f(&i); // &i signifie adresse de i (pointeur)
    Console.WriteLine(i);
    }
unsafe static void f(int *pi )
{ *pi = 20;  }
}
```

- Le programme affiche : 20

Les éléments de base de C#
Les modes d'exécution

- Mode managé : tous les compilateurs .net génèrent du code intermédiaire MSIL pour une exécution sous le contrôle du CLR (Run-Time de .net)
  - Qui gère l'allocation mémoire, et la libération automatique (ramasse-miettes), les contrôles de validité et de sécurité, les dépassements de bornes pour les tableaux etc.
- Mode non managé : mode d'exécution des compilateurs avant .net
  - Génération de code machine directement exécutable.
- Mode unsafe : code généré par les compilateurs .net (si fonctions unsafe). Ce code s'exécute en mode géré mais peut accéder à directement à des zones mémoires (pointeurs).
  - Mode intermédiaire entre managé et non managé.

Les concepts de base de la POO
Les classes

- L'encapsulation en POO consiste à décrire un ensemble d'objets à l'aide :
  - D'une classe, avec les propriétés des objets et les opérations/méthodes de manipulation de ces objets
    - Des droits d'accès sont définis sur les propriétés/attributs et sur les méthodes.

Les concepts de base de la POO
Les classes

- On s'intéresse plus particulièrement à la description d'un livre qui est identifié *(pour simplifier)*
  - par numéro d'isbn, son titre, sa date de parution, son prix, son éditeur et son ou ses auteurs.
- On souhaite définir :
  - Des méthodes d'observation pour connaître les propriétés d'un livre (accesseurs en lecture),
  - Des méthodes de mise à jour / modification pour modifier les valeurs de ses propriétés (accesseurs en écriture),
  - Des méthodes d'entrées/sorties pour saisir et afficher un livre.

Les concepts de base de la POO
Les classes

- Les classes peuvent être regroupées dans un espace de nom (namespace).
- Mais ce n'est pas une obligation. Les classes sont alors dans l'espace global.
  - On peut utiliser l'opérateur de résolution de portée :: pour accéder aux membres de la classe

```
namespace A
    { class B { ….} }
class C
    { public int N; ….}
```

Accès à l'attribut N sous la forme C::N

Les concepts de base de la POO
Les classes

- Dans la description des attributs de la classe, on utilise la classe DateTime qui permet de manipuler des dates et un tableau de chaînes de caractères pour le nom de auteurs *(pour simplifier).*
- Les attributs sont définis avec des droits d'accès « privés », ce qui est aussi le mode par défaut
  - string isbn;

```
public class Livre
 {
private string isbn;
private string titre;
private float prix;
private DateTime
dateParution;
private string [] tabA;
private int nbA;
 …
 }
```


Les concepts de base de la POO
Les classes

- Les champs d'une classe peuvent être initialisés lors de leur déclaration et ils peuvent être qualifiés de "const" ou de "readonly"
  - public const int max1=100;
  - public readonly int max2;
    - Dans ce cas, l'attribut devra être initialisé dans le constructeur et ne pourra pas être modifié ensuite.

Les concepts de base de la POO
Les classes

- Les droits d'accès sur les membres d'une classe peuvent être :
- public
  - Toutes les méthodes quelle que soit leur classe ont accès à un champ ou une méthode « public ».
- private
  - Seules les méthodes de la classe ont accès un champ ou une méthode « private ». C'est la valeur par défaut si les droits ne sont pas précisés.
- protected
  - Seules les méthodes de la classe ou de ses classes dérivées ont accès à un champ ou une méthode « protected ».
- internal
  - Seules les méthodes du même assemblage ont accès à un tel champs ou méthode .

Les concepts de base de la POO
Les classes

- Les méthodes d'observation de la classe livre.
- Elles permettent de connaître les propriétés du livre.
- La syntaxe est :

Droitsaccès Type-retour nom_méthode(paramètres)
{ corps de la méthode }

```
public string getIsbn()
{return isbn;
}

public string getTitre()
{ return titre;}

public DateTime getDateP()
{return dateParution;
}
public int getNbAuteur()
{return nbA;}

public string getAuteur(int i)
{return tabA[i];
}
```


Les concepts de base de la POO
Les classes

- Les méthodes de mise à jour/modification de la classe livre permettent de modifier les valeurs des propriétés du livre.
- Les attributs de la classe sont directement accessibles dans les méthodes de la classe.
- L'ajout d'un auteur est fait directement à partir de l'auteur passé en paramètre et supposé déjà initialisé avant l'appel de la méthode.

```
public void setIsbn(string isb)
{isbn = isb;}

public void setTitre(string t)
{titre = t;}

public void setPrix(float p)
{prix = p;}

public void ajoutAuteur(string a)
{ if (nbA <tabA.Length)
    tabA[nbA++]=a;          }

public void setDateP(DateTime d)
{   dateParution=d;}
```


Les concepts de base de la POO
Les classes

- **En C#, il existe une autre façon « usuelle » de définir des champs et des accesseurs avec des notations simplifiées.**

```
public class Livre
   {
  private string isbn;  // nom du champs
  public string ISBN  // nom de la propriété
     { get { return isbn; } // accesseur en lecture
       set { isbn = value; } // accesseur en écriture
     }
    …
  }
  …  // Usage de ces accesseurs
  Livre l= new Livre(); l.ISBN="12335666" ;
  Console.WriteLine(l.ISBN);
```


Les concepts de base de la POO
Les classes

- En C#, il existe une autre façon « usuelle » de définir des champs et des accesseurs avec des notations simplifiées.

```
public class Anniversaire
   {
  private int mois;  // nom du champs
  public int Mois // nom de la propriété
     { get { return mois; } // accesseur en lecture
       set { if (mois >=1 && mois <=12) mois= value;
             else mois=1;}              // accesseur en écriture
     }
    …
  }
  …  // Usage de ces accesseurs
  Anniversaire l= new Anniversaire(); l.Mois=12 ;
  Console.WriteLine(l.ISBN);
```


Les concepts de base de la POO
Les classes

- En C#, il existe une autre façon « usuelle » de définir des champs et des accesseurs avec des notations simplifiées.

```
public class Livre
   {
    public string Titre  // nom de la propriété
     { get ; set ;  } // accesseurs simplifiés, génération automatique
                      du champs associé en privé
    …
  }

  …  // Usage de ces accesseurs
  Livre l= new Livre(); l.Titre="12335666" ;
  Console.WriteLine(l.Titre);
```

Les concepts de base de la POO
Les classes

- En C#, il existe une autre façon « usuelle » de définir des champs et des accesseurs avec des notations simplifiées.

```
public class Anniversaire
   {
   private int mois;  // nom du champs
   public int Mois // nom de la propriété
     { get { return isbn; } // accesseur en lecture
        protected set { if (mois >=1 && mois <=12) mois= value;
                  else mois=1;}              // accesseur en écriture
     } // seules les méthodes de la classe ou de ses dérivées
                        peuvent utiliser la méthode set
    ...
   }
   ...  // Usage de ces accesseurs
   Anniversaire l= new Anniversaire(); l.Mois=12 ;
   Console.WriteLine(l.ISBN);
```


Les concepts de base de la POO
Les classes

- Les constructeurs sont des méthodes particulières qui permettent l'initialisation des valeurs d'un objet.
  - Le constructeur d'un objet est appelé à l'aide de la méthode « new ».
  - On peut définir plusieurs constructeurs pour un objet :
    - Un constructeur dit « par défaut » qui n'a pas de paramètre.
      - **Livre liv = new Livre();**
    - Des constructeurs avec paramètres qui permettent d'initialiser certains attributs de l'objet avec les valeurs des paramètres.
      - **Livre liv = new Livre("978-2-7460-5884-2");**
- Le constructeur est une méthode qui porte le même nom que la classe.

Les concepts de base de la POO
Les classes

- La classe Livre possède deux constructeurs :
- Un constructeur par défaut qui initialise le numéro d'isbn à "Inconnu", et le nombre d'auteurs à 0. Il alloue le tableau des auteurs.
- Un constructeur avec un paramètre qui permet de fixer le numéro d'isbn.

```
public Livre()
{
    tabA= new string[20];
    isbn="Inconnu"; nbA=0;

}

public Livre (string isb)
{
    tabA= new string [20];
    isbn=isb; nbA=0;
}
```


Les concepts de base de la POO
Les classes

```
public void saisie()
    {string s; int jour, mois, an;
      Console.Write("isbn ?"); isbn=Console.ReadLine();
      Console.Write("Titre ?"); titre=Console.ReadLine();
      Console.Write(" Prix ? "); s=Console.ReadLine(); prix = Single.Parse(s);
      Console.Write(" date de parution ? ");
      s=Console.ReadLine(); jour = Int32.Parse(s);
      s=Console.ReadLine(); mois = Int32.Parse(s);
      s=Console.ReadLine(); an = Int32.Parse(s);
      dateParution = new DateTime(an, mois, jour);
      do
      { Console.Write(" Nombre d'auteurs <= ?" + tabA.Length);
        s=Console.ReadLine(); nbA = Int32.Parse(s);  }
      while (nbA < 0 || nbA > tabA.Length);
      for (int i = 0; i < nbA; i++)
      {Console.Write("Auteur numéro " + i + "?");
        tabA[i]=Console.ReadLine(); }          }
```


Les concepts de base de la POO
Les classes

- La méthode « affiche » affiche toutes les informations d'un livre.

```
public void affiche()
{   Console. WriteLine("isbn : " + isbn);
    Console. WriteLine(" Titre :" + titre);
    Console. WriteLine(" Prix : " + prix);
    Console. WriteLine(" Date :" + dateParution.ToString("d"));
    for (int i=0; i<nbA; i++)
          Console. WriteLine(tab[i]);
}
```


Les concepts de base de la POO
Les classes

- Exemples de création d'objets de type Livre.
  - Livre liv1 = new Livre();
    - // déclare un livre – appel du constructeur par défaut
  - Livre liv2 = new Livre ("3-456-456-7");
    - // appel du constructeur avec paramètres
- Utilisation: appel des méthodes
  - liv1.saisie(); liv1.affiche();
  - using liv2;
  - { saisie(); affiche();}

Les concepts de base de la POO
Les classes

- **Les destructeurs**
- Ils existent en C# mais ne sont pas automatiquement appelés lors de la "disparition" d'une variable.
  - C'est l'environnement .net qui s'en charge lorsque « cela l'arrange … »
- Il existe un "paliatif" pour forcer l'appel d'un destructeur, il faut
  - Surcharger la méthode Dispose de l'interface IDispose,
  - Créer une zone de validité pour les objets (qui ont un destructeur) en utilisant using (objet) {…}
- L'appel de la méthode Dispose est fait à la fin de la zone using, automatiquement.
- Le destructeur a le même nom que la classe mais précédé du symbole ~

Les concepts de base de la POO
Les classes

- **Champs, méthodes et classes statiques**
  - Un champ "static" d'une classe est partagé par tous les objets de la classe et peut être initialisé et manipulé dans les méthodes de la classe
  - Une méthode "static" ne peut accéder qu'aux champs statiques d'une classe, peut appeler d'autres méthodes statiques.
  - Une classe "static" ne peut contenir que des attributs statiques.
  - L'appel à un membre statique d'une classe se fait directement avec le nom de la classe.

Les concepts de base de la POO
Les classes

```
class Compteur
{ private static int compt =0;
   public static int incr() { return compt++;}
   …
}
// utilisation
Console.WriteLine(Compteur.incr());
Console.WriteLine(Compteur.incr());
```


Les concepts de base de la POO
Les classes
Surcharge des méthodes et des opérateurs

- Les méthodes d'une classe peuvent être « surchargées »
  - C'est à dire que l'on peut définir dans une même classe plusieurs méthodes de même nom mais qui diffèrent par leurs paramètres.
  - Supposons que l'on souhaite définir une deuxième méthode d'affichage :
    - public void affiche (int);
    - Avec un paramètre qui indique l'attribut à afficher.

Les concepts de base de la POO
Les classes
Surcharge des méthodes et des opérateurs

- La classe Livre peut alors être complétée:
  - class Livre

```
{ ..
    public void affiche(int  i)
    { …
    }
}
```

  - Le compilateur choisit la bonne méthode à l'aide du profil des méthodes.

Les concepts de base de la POO
Les classes
Surcharge des méthodes et des opérateurs

- C# offre un mécanisme puissant de surcharge des opérateurs (inexistant en java, existant en C++).
- Les opérateurs de C# qui peuvent être surchargés sont notamment :
  - Les opérateurs arithmétique +, -, /, *, %
  - Les opérateurs de comparaison < <= > >= != ==
  - Les opérateurs d'incrémentation ++ --

Les concepts de base de la POO
Les classes
Surcharge des méthodes et des opérateurs

- La surcharge d'un opérateur consiste à définir cet opérateur pour un objet d'une classe définie par le programmeur.
- On s'intéresse à la comparaison de deux livres, en surchargeant les opérateurs == et !=.
  - La comparaison de deux livres se fera sur leurs numéros d'isbn.
- Pour implémenter complètement la comparaison d'objets, il est nécessaire en C# de :
  - Redéfinir la méthode "Equals" de la classe "Object" dans la classe Livre.
  - Redéfinir les opérateurs == et !=
  - Redéfinir la méthode "GetHashCode" de la classe "Object" pour éviter les problèmes si l'on crée une collection Hashtable de Livre.

Les concepts de base de la POO
Les classes
Surcharge des méthodes et des opérateurs

- Surcharge de la méthode Equals de classe Object dans la classe "Livre".

```
public override bool Equals(Object o)
      {   if (o != null)
          {Livre liv = (Livre) o;
          return (this.isbn == liv.isbn);}
           return false;
      }
```

La comparaison des livres est faite sur leurs numéros d'isbn.

Les concepts de base de la POO
Les classes
Surcharge des méthodes et des opérateurs

- Surcharge des opérateurs == et != pour la classe Livre en utilisant la méthode Equals.

```
public static bool operator ==(Livre l1, Livre l2)
      {
         if ((Object) l1 == null) return (Object) l2 == null)
         return l1.Equals(l2);
      }
public static bool operator !=(Livre l1, Livre l2)
      {
         return ! (L1 == l2);
      }
```


Les concepts de base de la POO
Les classes
Surcharge des méthodes et des opérateurs

- Surcharge de la méthode GetHashCode de la classe Object pour la classe Livre.

```
public override int GetHashCode()
      {
         return this.ToString().GetHashCode();
      }
```

- Comparaison de deux livres :

```
if (l1 == l2) Console.WriteLine("isbn égaux");
        else Console.WriteLine("isbn différents");
```