C # 5

Souhila GUERFI

Introduction

- **Caractéristiques partagées avec le langage Java**
  - Syntaxe : les mots clés communs avec Java s'utilisent dans les mêmes circontances et de la même manière : public, private, protected,abstract,
  - class, interface, try, catch, finally, throw, new, return, this, if, while, do, for, foreach, enum... et le mot clé lock est l'équivalent C# du mot clé Java synchronized;
  - Références : les objets sont en fait des références ;
  - Documentation automatique : le langage C# utilise les commentaires /// au format XML ;

Introduction

- **Caractéristiques partagées avec le langage C++**
  - surcharge des opérateurs ;
  - structures (mot clé struct) ;
  - énumérations (mot clé enum) ;
  - pointeurs : il est possible, en mode *unsafe*, d'utiliser des pointeurs au lieu de références.

Introduction

- Compilation
  - Un fichier source C# porte l'extension ".cs"
  - Sous Windows, le compilateur produit un exécutable appelant l'interpréteur .Net.
  - Sous Linux, le programme produit n'est pas directement exécutable et doit être lancé en argument de l'interpréteur mono.

Introduction

- **Compilateur pour Windows**
  - Si le framework Microsoft .NET est installé, le compilateur nommé csc.exe doit se situer dans
  - C:\WINDOWS\Microsoft.NET\Framework\vnuméro_de_version_du_framework. Le framework .NET est disponible gratuitement pour les utilisateurs de Windows.
  - Il est également possible d'utiliser Visual Studio .NET pour développer, compiler et débogguer les applications C#. L'édition Express de Visual Studio est téléchargeable gratuitement sur le site de Microsoft , mais possède moins d'outils que la version complète.

Introduction

- Une autre possibilité est d'utiliser SharpDevelop qui a l'avantage d'être un logiciel libre

- **Compilateur pour Linux**
  - Mono est une implémentation libre de la plate-forme de développement Microsoft .NET. Le compilateur est nommé msc. L'interpréteur est nommé mono
  - gmcs est le nouveau compilateur C# 2.0. Il est recommandé d'utiliser ce dernier.

Introduction/Un premier programme

- Le fichier source
  - Enregistrez le programme suivant dans un document texte, intitulé par exemple « helloworld.cs » :

```
using System;
public class HelloWorld
{
 public static void Main()
 {
 Console.WriteLine("Hello world !");
 Console.ReadLine();
 }
}
```

Introduction/Un premier programm

- Détails
  - using System;
    - Le fichier source utilise l'espace de noms nommé "System".
  - Déclaration d'une classe nommée "HelloWorld".

    public static void Main()

    {
    - Déclaration d'une méthode statique nommée "Main" dans la classe HelloWorld. Cette méthode est celle qui est appelée au démarrage du programme.
  - Console.WriteLine("Hello world !");
  - Affichage de la ligne "Hello world !" sur la console. Console désignant la console, appartient à l'espace de nom System.

Introduction/Un premier programm

- Détails (suite)
  - Console.ReadLine();
  - Attendre que la touche entrée soit frappée avant de poursuivre. Cette ligne de code n'est pas nécessaire si vous lancez le programme depuis une console déjà ouverte. Dans le cas contraire (double-clic sous Windows), cette ligne de code permet de maintenir la fenêtre de console ouverte, car celle-ci se ferme une fois le programme terminé (option par défaut).
  - }
    - Fin de la méthode Main

.

Bases du langage

- Les varaibles, constantes et enumérations
  - Les variables
  - Les types numériques entiers
  - Types virgule flottante
  - Les types caractères
  - Le type bool
  - Le type Object
  - Inférence de type
  - Le type dynamic
  - Les constantes
  - Les énumérations
  - Inférence de type

Bases du langage

- Les tableaux
- Les structures
- Les structures de contrôles
- Les structures de conrôle
- Les procédures et les fonctions
- Programmation Objet

Bases du langage

- Les varaibles, constantes et enumérations
  - Les variables
    - Le nom d'une variable commence obligatoirement par une lettre ou par (_)
    - Il ya yne distinction entre miniscule et majuscules
    - Si vous avez besoin d'utiliser un mot clé en tant qu'itentifiant, il faut préfixer le nom par le caractère @

Bases du langage

- Les types numériques entiers

Type	Plage	Taille
sbyte	-128 à 127	Entier 8 bits signé
byte	0 à 255	Entier 8 bits non signé
short	-32 768 à 32 767	Entier 16 bits signé
ushort	0 à 65 535	Entier 16 bits non signé
int	-2 147 483 648 à 2 147 483 647	Entier 32 bits signé
uint	0 à 4 294 967 295	Entier 32 bits non signé
long	-9,223,372,036,854,775,808 à 9,223,372,036,854,775,807	Entier 64 bits signé
ulong	0 à 18,446,744,073,709,551,615	Entier 64 bits non signé

Bases du langage

- Types virgule flottante
  - float i = 1.5f

float	System.Single	Nombre à virgule flottante
double	System.Double	Nombre à virgule flottante sur 64 bits
decimal	System.Decimal	Nombre à virgule flottante sur 128 bits

Bases du langage

- Les types caractères
  - Le type char est utilisé pour stocker un caractère unique
  - Exemple :
  - char = ‘v'
  - string Nom = " Crochet " ;
  - **string** chaine = @" que dit il ? il dit "bonjour" "
- Obtention de la longeur d’une chaîne
- Console.WriteLine(" la chaîne contient {0} caractères " chaîne1***.Length***);

Bases du langage

- Les types caractères
- Découpage de chaîne
  - La fonction ***Substring(pos, nomb)*** retourne une portion de chaîne en foncion de la position de départ et du nombre de caractères à retourner qui lui est passé.
- Comparaison de chaînes
- La fonction Equals : ==
  - Syntaxe ***chaîne1.Equals(chaîne2)***
- La fonction chaine1.**CompareTo(chaîne2)**
- La fonction chaine1.***Insert***(entier, ‘’chaine’’)

Bases du langage

- Les types caractères
  - Changement de la casse
    - Tout en majuscule : chaine1.ToUpper()
    - Tout en minuscules : chaine2.ToLower()
  - Remplacement dans une chaîne
    - Chaine3 = chaine1.Replace(‘’hiver’’, ‘’été’’);

Bases du langage

- Le type bool
  - Disponible = true ;
- Le type Object
  - Dans une variable de type Object, vous pouvez stocker n’importe quoi
  - Dans une variable de type Object, vous pouvez stocker n’importe quoi

Bases du langage

- Le type dynamic
  - Il arrive parfois que le type de la variable ne soit connu qu'au moment de l'exécution de l'application. Il est, dans ce cas, possible d'utiliser le mot-clé **dynamic**
  - Exemple:

```
public static dynamic operation(dynamic o1, dynamic o2) { return o1+o2 ; }
```

Bases du langage

- Conversion de types
  - Conversion vers une chaîne de caractères
    - La fonction format de la string permet de choisir la forme du resultat
    - Exemple String.format("{ 0:p} " , 0.2);
    - Resulat : 20,00%
    - Exemple float tva = float.Parse(" 18,6 ‘’);

Bases du langage

- Inférence de type
  - Grâce à l'inférence de type, le compilaeur peut déterminer le type à uiliser pour une variable local
  - var nom = " Dupont "
- Les constantes
  - Const int ValeurMaxi = 100;
- Les énumérations
  - enum jours { Dimanche, Lundi,....}
  - Une fois définie, une énuméraion peut ensuite être utilisée comme un type de variable spécifique.
    - **jours taille;**

Bases du langage

- Les tableaux
- Déclaration d'une variable de type tableau
  - int [] chiffres ;
- Réserver une zone mémoire pour stocker le contenu du tableau.
- chiffres = new int[12]
- Tableaux à plusieurs dimensions
  - int[..] Cube ;
  - cube = new int[4,4,4]
  - int [,] grille ={ {1,2}, {3,4}}

Bases du langage

- Les tableaux
- Déclaration d'une variable de type tableau
  - int [] chiffres ;
- Réserver une zone mémoire pour stocker le contenu du tableau.
- chiffres = new int[12]
- Tableaux à plusieurs dimensions
  - int[, ,] Cube ;
  - cube = new int[4,4,4];
  - int [,] grille ={ {1,2}, {3,4}};

Bases du langage

- Les tableaux
  - La propriété ***Rank*** d'un tableau renvoie directemen la dimension du tableau.
    - Exemple ; int [,] matrice = {{1,2}, {3,4},{5,6}};
    - Console.WriteLine(" ce tableau comporte {0} dimension " matrice.Rank);
  - Recherche un élément dans un tableau
  - la fonction ***IndexOf***
  - ***string [] gouter = {" pain " , " beurre ", " moutarde ", " confiture " } ;***
  - ***Console.WriteLine(Array.IndexOf(gouter, " moutarde ");***

Bases du langage

- Trier un tableau : la procédure Sort de la classe Array
  - la fonction ***string [] gouter = {" pain " , " beurre ", " moutarde ", " confiture " } ;***
  - ***Array.Sort(gouter);***
  - ***foreach(string plat in gouter)***

{Console.WriteLine(plat);}

Bases du langage

- Les structures
  - struct Client {

    public int code ;

    public string nom ;

    public string prénom ;

    Public Adresse coordonnees; /* Adresse : struct*/

    }
  - Il est impossible d'initialiser les membres d'une structure au moment de la déclaration.
- Utilisation des structures

```
Client unClient ;
unClient.code = 999 ;
unClient.coordonnees.numero = 42 ;
```

Bases du langage

- Opérateurs

Bases du langage

- Les structures de contrôles
  - Structure if (condition) { instruction ; }
  - Structure switch (variable) {

```
case valeur1 :
                break;
case valeur2 :
                break
default :
            break;}
```

- Les structures de boucle
  - while (condition) {}
  - do{}While(codition);
  - for(initialisation; condition; instruction d'itération) {}
  - foreach(element in tableau) {}

Bases du langage

- Structure using
  - Cette structure est destinée à accueillir un bloc de code utilisant une ressource externe, comme par exemple un fichier.
  - Exemple

```
using ( StreamWriter sw = File.CreateText(path))
{ sw.WriteLine(‘’rouge’’);
   sw.WriteLine(‘’vert’’);
}
```

Bases du langage

- Les procédures et fonctions
  - Procédure

    void AffichageResultat()

    { Console.WriteLine(‘’fff’’)}
  - Fonction int calcul(){return n ;}
  - Procédure de propriétés
    - Vont nous permettre d’ajouter une propriété à une classe, un module ou une structure.
    - La syntaxe :

    public typeDelaPropriete nomPropriete

    { get {....}

    set{....}

    }

Bases du langage

- Procédure de propriétés
  - Exemple public in taux { get; set;}
- Les procédures opérateur
  - Ce type de procédure permet la redéfinition d'un opérateur standard du langage pour l'utiliser sur des types personnalisés (classe ou structure)

Exemple

```
Client c1, c2, c3
c3 = c1+c2;// erreur
```

Bases du langage

- Procédure de propriétés

  La solution :

```
public static Client operator +(Client cl1,Client cl2)
{  Client c;
 c.code = cl1.code + cl2.code;
c.nom = cl1.nom +cl2.nom;
c.prenom = cl1.prenom +cl2.prenom;
return c ;
}
```

Bases du langage

- Les arguments :
- Pour passer une variable comme paramètre à une procédure, il existe deux possibilités:
  - Passage par valeur
  - Passage par référence : *ref , out*
  - L'utilisation du mot-clé ***ref*** dans la déclaration d'une fonction impose deux containtes lors de l'appel de la fonction :
    - Ce mot-clé doit également être utilisé lors de l'appel.
    - La variable doit obligatoirement être initialisée avant l'appel.

Bases du langage

- *Exemple :*

```
double prixHt = 100 ;
double prixTtc ;
double montantTva = 0;
PrixTtc = TestStructure.CalculTTC(prixHt, 5.5, ref
montantTva) ;
```

- Le mot clé *out* présente un fontionnement similaire hormis la contraite d'initialisation obligatoire ne s'applique pas.

Bases du langage

- Une autre possibilité permet de créer une procédure qui pourra prendre un nombre quelconque de paramètres. Mot-clé ***params***
- Exemple :
- public static double moyenne(***params*** double [] notes)
- La fonction peut ensuite être appelée :
- Resultat = moyenne(1,2,6,45) ;

Bases du langage

- Serialisation
  - Permet l'enregistrement d'une instance de classe, dans un fichier binaire ou XML.
  - [Serializable()] public class Personne

    { }

Bases du langage

- Serialisation

```
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
    Stream flux;
        flux = File.Open("c:\\données.bin", FileMode.Create);
        BinaryFormatter formateur;
        formateur = new BinaryFormatter();
        formateur.Serialize(flux, objetAserialiser);
        flux.Close();
```

Bases du langage

- DeSerialisation
- Exemple :

```
BinaryFormatter formatter = new BinaryFormatter();
        FileStream flux = null;
//On ouvre le fichier en mode lecture seule. De plus, puisqu'on a sélectionné le mode Open,
flux = new FileStream(path, FileMode.Open, FileAccess.Read);
Person p = formatter.Deserialize(flux) as Person;
```

Programmation Objet

- Déclaration de la classe
- La syntaxe générale de définition d'une classe est donc la suivante :

```
[attributs][modificateurs][partial] class NomClasse [: classe de base ] [, interface1, interface2, ...]
{
 //code
}
```

Programmation Objet

- Les niveaux d'accés
  - public
    - La classe pourra être utilisée dans tout votre projet mais aussi dans d'autres projets
  - internal
    - L'accés à la classe est limité au projet dans lequel elle est définie.
  - private
    - La classe ne peut être utilisée que dans le module dans lequel elle est définie.
  - protected
    - La classe ne peut être utilisée que dans une sous-classe de celle dans laquelle ellz est définie.

Programmation Objet

- abstract
  - Indique que la classe sert de la classe de base dans une relation d'héritage. Vous ne pourrez pas créer d'instances de cette classe.

- Classe partielle
  - La définition d'une classe peut être répartie sur plusieurs déclarations en utilisant le mot-clé ***partial***
  - Les differentes parties de la définition d'une classe doivent par contre être dans le même projet et faire partie du même namespace

Programmation Objet

- Consucteurs et destructeurs
  - Le constructeur est une méthode portant le même nom de la classe

```
public Personne()
        { leNom = "";
            lePrenom = "";
            leMotDePasse = "";
        }
        public Personne(string nom, string prenom, string pwd)
        {
            leNom = nom;
            lePrenom = prenom;
            leMotDePasse = pwd ;}
```

Programmation Objet

- Création d'une instance
  - Exemple : Personne p ; // p contient la vleur null
  - P = new Personne() ;

Programmation Objet

- Création de propriétés (set et get)
- Exemple

```
public string nom
{
    get
    {
        return leNom;
    }
    set
    {
        leNom=value.ToUpper();
    }
}
```

Pogrammation Objet / classe-Exemple

```
public  class Personne
  {
    private string leNom;
     private string lePrenom;
     private DateTime laDate_naiss;

      public Personne()
            { leNom = "";
                lePrenom = "";


            }

      public Personne(string nom, string
      prenom)
            {
                leNom = nom;
                lePrenom = prenom;
      }
```

```
public string nom
        {
            get
            {
                return leNom;
            }
            set
            {
                    leNom=value;
                    }
        }
public string prenom
 { get
      {
                return lePrenom ;
 }
set
            {
                lePrenom =
value;
            }
         }
```

```
   public DateTime date_naiss
          {
              get
              {
                  return laDate_naiss;
              }
              set
              {
                  laDate_naiss = value;
              }
          }


public int age
{
    get
    {
        return DateTime.Now.Year -
laDate_naiss.Year;
    }
}
}
```

Programmation objet / classe

- Propriétés indexées
  - Permettent un accès de type tableau à des groupes d'éléments
  - Elles attendent un paramètre indiquant le groupe auquel il faut accéder
  - Cette propiété ne dispose pas de nom.
  - Il est possible de lui en spécifier un ajoutant l'attribut IndexeName à la définition de la propriétés.

Programmation objet / classe

- Propriétés indexées
- exemple :

```
private Personne[] lesEnfants = new Personne[10];
    public Personne this[int index]
        {
            get
            {
                return lesEnfants[index];
            }
            set
            {
                lesEnfants[index] = value;
            }
        }
```

Programmation objet / classe

- Tester le fonctionnement de la classe personne

```
public static void Main()
{
    Personne p= new Personne();
    Personne enfant1 =new Personne();
    Personne enfant2=new Personne();
    p.nom = "dupond";
    p.prenom = "paul";
    p.date_naiss = new DateTime(1954,12,23);
    enfant1.nom = "dupond";
    enfant1.prenom = "pascal";
    enfant1.date_naiss = new DateTime(1979,10,5);
    // nous pouvons également utiliser le nom du parent pour
    // initialiser le nom de l'enfant
}
```

Programmation objet / classe

- Tester le fonctionnement de la classe personne **(suite)**

```
enfant2.nom = p.nom;
   enfant2.prenom = "marc";
   enfant2.date_naiss = new DateTime(1982,4,18);
   // nous pouvons affecter un enfant à une personne
   p[0] = enfant1;
   p[1] = enfant2;
   // vérifions que nos informations sont correctes
   Console.WriteLine("Mr {0} {1} né le {2} a 2 enfants", p.nom, p.prenom,p.date_naiss);
   Console.WriteLine("{0} {1} qui a {2} ans", p[0].nom, p[0].prenom, p[0].age);
   Console.WriteLine("{0} {1} qui a {2} ans", p[1].nom, p[1].prenom, p[1].age);
   Console.WriteLine("appuyer sur une touche pour quitter");
      Console.ReadLine();
```

Programmation objet / héritage

- La mise en oeuvre est très simple, il suffit juste de spécifier le caractère : suivi du nom de la classe dont on souhaite hériter.

- Exemple

```
class Client:Personne
   {
      protected int lecode;

      public int code
      {
         get
         {
            return lecode;
         }
         set
         {
            lecode = value;
         }
      }
   }
```

Programmation objet / héritage

- Constructeur
  - Dans une classe dérivée, si un constructeur de classe de base n'est pas appelé explicitement à l'aide du mot-clé ***base***, le constructeur par défaut, s'il existe, est appelé implicitement.
  - Exemple:

```
public Client():base() {
leCode = 0;
}
```

```
public Client(string nom, string prenom, DateTime datenais1, int code):
base(nom, prenom, datenais1)
{
        leCode = code;
}
```

Programmation objet / héritage

- Substitution de méthode
  - La substitution permet la redéfinition de méthodes ou propriétés héritées d'une classe de base.
  - La subsutitution est utilisée pour assurer le **polymorphisme** entres les classes
  - Les méthodes substituées doivent :
    - Le même nom
    - Le même nombre de paramètres et type de paramètres que les méthodes de la classe de base.

Programmation objet / héritage

- Substitution de méthode (suite)
  - Il faut dans ce cas utiliser le mot-clé ***override*** lors de la subtitution de la classe dérivée
  - Il est également que la classe de base ait autorisé cette subtitution par l'utilisation du mot- clé ***virtual***

```
// classe Client
public  override void affichage()
        {

            Console.WriteLine("Mr {0} {1} né le {2} gagne {3} euros par mois", nom, prenom, date_naiss, code);
        }
```

```
// classe Personne
public virtual void affichage()
        {
            Console.WriteLine("Mr {0} {1} né le {2}", nom, prenom, laDate_naiss);
        }
```

Programmation objet / héritage

- Substitution de méthode (suite)
  - le mot-clé ***sealed***
    - peut être utilisé pour bloquer à partir d'un niveau donné la substitution

```
// classe Client
public  sealed override void affichage()
        {

            Console.WriteLine("Mr {0} {1} né le {2} gagne {3} euros par mois",
nom, prenom, date_naiss, code);
        }
```

Programmation objet / héritage

- Masquage de méthode
  - Lorsque des éléments d'un programme partagent le même nom, l'un d'eux peut être masquer l'autre.
  - Lors du masquage, il convient d'utiliser le mot-clé ***new,*** devant le nom du membre assurant le masquage
  - Exemple : ajouter dans la classe Client

```
public  new  int age
        {

            get
            {
                return DateTime.Now.Year - laDate_naiss.Year;
            }
        }
```

Programmation objet / héritage

- Masquage de méthode
  - fonction age est masqué et inaccessible.
  - la propriété age héritée de la classe personne est masquée.

Programmation objet

- Méthodes partielles
  - Elles sont utilisées pour fournir une notification de changement.
    - La méthode doit être une procédure
    - Le corps de la méthode doit être vide
    - La méthode ne doit pas avoir de modificateur d'accés.
  - Exemple : définir la classe Personne comme étant une classe partielle, soit une méthode partielle nomChanged()

Programmation objet

- Méthodes partielles
  - Définir la méthode nomChanged() dans un autre fichier source.

```
// fichier Personne.cs
namespace Cours
{
    public partial class Personne
    { .....
.....

partial void nomChanged();
}

}
```

```
// fichier Personne_suite.cs
namespace Cours
{
    partial class Personne
    {
        partial void nomChanged()
        {
            Console.WriteLine("ok");
        }
    }
}
```

Programmation objet

- Méthode d'extension
  - Les méthodes d'extension permettent l'ajout de fonctionnalités à une classe déjà définie.
  - Quelques règles à respecter :
    - Elles peuvent être de type procédure ou fonction mais pas de type propriété
    - Le premier paramètre doit être précédé du mot-clé ***this***
    - Au moment de l'exécution ce premier paramètre représente l'instance.
    - Elles doivent être dans une classe ***static***
    - Elles doivent elles même être ***static***

Programmation objet

- Méthode d’extention
- Exemple : Ajouter une classe Extensions

```
static class Extensions
    {
        public static void presentation(this Personne p)
        {
            Console.WriteLine("nom : {0}", p.nom);
            Console.WriteLine("prenom : {0}", p.prenom);
            Console.WriteLine("date de naissance : {0}", p.date_naiss);
        }
    }
```

Programmation objet

- Classes abstraites
  - Il est impossible de créer une instance d'une classe abstraite.
  - Elles servent essentiellement de modèle pour la création de classe.
  - Elles peuvent contenir des champs, des propriétés et des méhodes comme une classe ordinaire.

```
public abstract class Modele
{
}
```

Programmation objet

- Classes finales
  - Les classes finales sont des classes ordinares qui peuvent être instanciées mais ne sont pas utilisables comme classe de base dans une relation d’héritage.

Programmation objet

- Interface
- S'il y a plusieurs classes qui doivent implémenter la même méthode, il est plus pratique d'utiliser les interfaces.
- Elles ne contiennent aucun code.
- Exemple :

```
interface Comparable
   {  // cette fonction permet de comparer l'instance courante
//d'un objet et l'objet qui sera passé comme paramètre
       int compare(Object o1);
   }
```

Programmation objet

- Interface
- Par exemple, dans la classe Client, nous pourrions implémenter l'interface comparable de la manière suivante en choisissant de comparer deux Clients sur le nom :

Programmation objet

- Interface

```
namespace Cours
{
    class Client :Personne, Comparable
    { .....
public int compare(Object o1)
        {
            Client c;
            if (o1 is Client)
            {
                c = (Client)o1;
            }
            else
            {
                throw new InvalidCastException();
            }
            return leNom.CompareTo(c.leNom);
        }

    }
```

Programmation objet

- Interface
- Exemple 2 :

Programmation objet/Les collections

- Les collections prédifinies
  - Les différentes classes permettant la gestion de collections sont réparties dans deux espaces noms :
    - System.Collections
    - System.Collections.Generic // collections typées
- Classe Array
    - Ne fait pas partie de l'espace noms System.Collections
    - Elle implémente l'interface Ilist.
    - Les tableaux créés à partir de la classe Array sont de taille fixe.
    - Deux propriétés : Length , Rank

Programmation objet/Les collections

- ArrayList et List
  - La taille d'un ArrayList est dynamique et est automatiquement ajustée
  - Elle propose des méthodes permettant l'ajout, l'insertion et la suppression de plusieurs éléments simultanément
  - Les ArrayList n'ont qu'une seule dimension.
  - Un ArrayList doit être instancié avant de pouvoir être utilisé.
  - La propriété Capacity permet de connaître le nombre d'éléments
  - La propriété Count indique le nombre actuel
  - Les méthodes Insert et InsertRange permette de choisir l'emplacement où va s'effectuer l'ajout
  - La propriété Item, s'utilise pour atteindre un élément à une position donnée.

Programmation objet/Les collections

- ArrayList et List
  - Les méthodes RemoveAt ou RemoveRange; la première attend comme paramètre l'index de l'élément à supprimer, la deuxième exige en plus le nombre d'éléments à supprimer.
  - La propriété Clear supprime tous les éléments

Programmation objet/Les collections

- Exemple

```
ArrayList liste;
   Client c;
   liste = new ArrayList();
   Console.WriteLine("capacité initiale de la liste {0}", liste.Capacity);
   Console.WriteLine("nombre d'éléments de la liste {0}", liste.Count);
   Console.WriteLine("ajout d'un client");
   c = new Client("client1", "prenom1", new DateTime(1964,12,23), 1001);
   liste.Add(c);
   Console.WriteLine("capacité de la liste {0}", liste.Capacity);
   Console.WriteLine("nombre d'éléments de la liste {0}", liste.Count);
   Console.WriteLine("ajout de quatre clients");
   c = new Client("client2", "prenom2", new DateTime(1964,12,23), 1002);
   liste.Add(c);
   c = new Client("client3", "prenom3", new DateTime(1964,12,23), 1003);
   liste.Add(c);
```

Programmation objet/Les collections

- Exemple

```
c = new Client("client4", "prenom4", new DateTime(1964, 12, 23), 1004);
 liste.Add(c);
 c = new Client("client5", "prenom5", new DateTime(1964, 12, 23), 1005);
 liste.Add(c);
 Console.WriteLine("capacité de la liste {0}", liste.Capacity);
 Console.WriteLine("nombre d'éléments de la liste {0}", liste.Count);
 Console.WriteLine("affichage de la liste des clients");
 foreach ( Client cl in liste)
 {
    cl.affichage();
    Console.WriteLine();
 }
 Console.WriteLine("effacement des clients 1002, 1003, 1004");
 liste.RemoveRange(1, 3);
 Console.WriteLine("capacité de la liste {0}", liste.Capacity);
 Console.WriteLine("nombre d'éléments de la liste {0}", liste.Count);
```

Programmation objet/Les collections

- Exemple

```
Console.WriteLine("affichage de la liste des clients");
  foreach ( Client cl in liste)
  {
     cl.affichage();
     Console.WriteLine();
  }
  Console.WriteLine("affichage du deuxième client de la liste");
  ((Client)liste[1]).affichage();
  Console.WriteLine();
  Console.WriteLine("effacement de tous les clients");
  liste.Clear();
  Console.WriteLine("capacité de la liste {0}", liste.Capacity);
  Console.WriteLine("nombre d'éléments de la liste {0}", liste.Count);
  Console.ReadLine();
```

Programmation objet/Les délégués

- Les délégués sont des méthodes appelables sans connaissance préalable de l'objet cible.
- les délégués permettent d'appeler une fonction à la place d'une autre, ce qui peut laisser sceptique. Les délégués répondent pourtant à un besoin précis : prévoir dans le code d'appeler une méthode, sans savoir lors de la phase de programmation quelle sera cette méthode, celle-ci n'étant précisée que lorsque le programme est lancé. Par exemple, on peut avoir besoin de lancer un tri, mais ne pas être en mesure de déterminer le type de tri (Bulle, Quick, Merge).avant que l'application ne tourne. L'idée est que ce n'est plus l'objet initial qui gère l'activité, il peut la déléguer à une autre fonction qui prend en charge l'ensemble du processus [http://www.journaldunet.com/]

Programmation objet/Les délégués

- La déclaration d'un délégué est précédée du mot-clé ***delegate*** et la déclaration contient que le prototype de la méthode.
- Exemple

  public delegate int Calcul(int i, int j) ;

  - Pour créer une instance de délégué, il suffit d'assigner une méthode dont la signature est conforme au délégué :

```
public class Exemple
{
 public Exemple(int  i ,  int j){
          Calcul C = new Calcul(Addition) ;
          int result = C.invoke(i,j);
}
Public int Addition(int i , int j) {       return i+j ; }
}
```

Programmation objet/Les délégués

- Les délégués peuvent être utilisés comme paramètres d'une méthodes.
- Exemple

```
public delegate void Afficher(int  i) ;
public class Exemple
{
public void Addition (int i ,int j, Afficher  CB)
{
                CB(i+j);
}
public void Affiche(int i)
{
               Cosole.WriteLine(i);
}
public Exemple(int i , int j ) {
            Affficher A = new Afficher(Affiche) ;
           Addition(i, j, A) ;
}
}
```

Programmation objet/Les délégués

Exemple 2: Soit deux méthodes : compareCode, compareNbCommandes

// déclaration délégué

public delegate int comparaison(Client c1, Client c2) ;

```
public static int compareCode(Client c1, Client c2)
{
    if (c1.code < c2.code)
     {
        return -1;
     }
     if (c1.code > c2.code)
     {
        return 1;
     }
     else
     {
        return 0;
     }
}
```

```
public static int compareNbCommandes(Client c1, Client c2)
{
    if (c1.nombreCommandes < c2.nombreCommandes)
    {
        return -1;
    }
    if (c1.nombreCommandes > c2.nombreCommandes)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

Programmation objet/Les délégués

- modifier la fonction de tri, elle prenne comme paramètre le tableau à trier, la fonction utilisée pour la comparaison de deux élément du tableau :

```
public static void tri(Client[] tablo,comparaison comparateur)
    {
        Client o;
        for (int i = 0; i < tablo.Length - 1; i++)
        {
            for (int j = i + 1; j < tablo.Length; j++)
            {
                if (comparateur.Invoke(tablo[j],tablo[i]) < 0)
                {
                    o = tablo[j];
                    tablo[j] = tablo[i];
                    tablo[i] = o;
                }
            }
        }
    }
```

Programmation objet/Les délégués

- Pour utiliser la fonction tri, créer tout d'abord l'instance une instance du délégué :

Comparaison cmp = new Comparaison(compareCode);

- Appeler la fonction tri
- tri(tab , cmp);

Programmation objet/ Les expressions lambda

- Une expression lambda est un morceau de code.
- Elle peut être assimilée à une méthode car elle accepte des paramètres et peut retourner une valeur.
- Les expressions lambda sont particulièrement utiles lorsque vous souhaitez appliquer un traitement métier sur un objet ou une collection d'objets sans pour autant altérer le code de ces objets. [http://www.dotmyself.net/]
- Les expressions lambda sont particulièrement utiles pour écrire des expressions de requête LINQ.

Déclaration et utilisation des expressions lambda

- Une expression lambda a la forme suivante:

(parameters) => expression-or-statement-block

Programmation objet/ Les expressions lambda

- Exemple

```
String log = "";
 List<int> ints = new List<int> { 0, 10, 20, 30 }; ints.foreach(item => log += item);
Console.WriteLine(log);
```

- L'expression lambda (item => log += item) concatène les éléments de la collection passée en paramètre. Pour ce faire, elle utilise la variable **log** déclarée dans son contexte.

Programmation objet/ Les expressions lambda

- Exemple ci-dessous appelle la fonction tri en lui passant, comme deuxième paramètre, une expression lambda respectant la signature du délégué attendu et effectuant le tri sur le nom du client.

```
public delegate int comparaison(Client c1, Client c2) ;
Client[] tablo;
tablo = new Client[5];
tablo[0] = new Client("client2", "prenomClient2", new DateTime(1962,12,23), 2);
tablo[1] = new Client("client1", "prenomClient1", new DateTime(1965, 11, 18), 1);
tablo[2] = new Client("client5", "prenomClient5", new DateTime(1970, 05, 23), 5);
tablo[3] = new Client("client3", "prenomClient3", new DateTime(1982, 06, 02), 3);
tablo[4] = new Client("client4", "prenomClient4", new DateTime(1960, 02, 18), 4);
```

Programmation objet/ Les expressions lambda

```
Console.WriteLine("affichage du tableau original");
for (int i = 0; i < tablo.Length; i++)
    {
        Console.WriteLine(tablo[i].nom + " " + tablo[i].code);
    }
// utilisation de l'expression lambda
tri(tablo, (Client c1, Client c2) => c1.nom.CompareTo(c2.nom));

Console.WriteLine("affichage du tableau trié");
for (int i = 0; i < tablo.Length; i++)
    {
        Console.WriteLine(tablo[i].nom + " " + tablo[i].code);
    }
```

Programmation objet/ Les classes anonymes

- C# fournit la possibilité de créer des objets sans écrire de code pour la définition de la classe correspondante.
- C’est le compilateur qui va prendre en charge la génération de la classe.
- En général, les types anonymes sont utilisés dans la clause select d'une expression de requête pour retourner un sous-ensemble des propriétés de chaque objet [msdn]

Programmation objet/ Les classes anonymes

- **Exemple**

```
var produit = new {    nom = ‘’biscuit’’ , prix = 1.56} ;
 Console.WriteLine(produit.GetType().Name);
```

- La variable produit fait référence à une instance de classe contenant deux propriétés nom et prix.
- Les types anonymes sont des types d' classe qui dérivent directement d' objet, et qui ne peuvent pas être castés à aucun type sauf objet. [msdn]

Programmation objet/ LINQ

- Présentation de LINQ

- Premières requête LINQ
  - La manipulation d'une requête LINQ est constituée de trois actions
    - L'obtention des données.
    - La création de la requête elle même
    - L'exécution de la requête.
  - La requête contient trois clauses:
    - from : indiquant l'origine des données.
    - Where : spécifie la ou les conditions
    - Select ; indique quelles sont les informations retournées

Programmation objet/ LINQ

- Exemple

```
var requete = from unClient in listesClients
where unClient.nom.StartsWith(‘’A’’)
select unClient ;
```

  - Une requête LINQ est stockée dans une variable puis est exécutée ultérieurement.

```
// exécution de la requete
foreach (var unClient in requete )
{
    Console.WriteLine(unClient.nom);
}
```

Programmation objet/ LINQ

- Les opérateurs de requête
- Tri de donnée (orderby , descending (ordre croissant), ascending)

- Opérations sur des ensembles de données
  - ***distinct*** placé à la fin de la clause select indique que les doublons seront éliminés.
- Filtrage (Equals...)
- Projections
  - Correspond à la transformation d'un l'objet on une nouvelle forme.

Programmation objet/ LINQ

- Projections

Exemple

```
var requeteProjection = from unClient in listeClients
                        from uneCommande in listeCommande
                        where unClient.Equals(uneCommande.LeClient);
                        select new { cli = unClient, cmd = uneCommande};
Foreach (var r in requeteprojection)
{   Console.WrieLine(r.cli.nom+ ‘’    ‘’  + r.cmd.dateCommande ) ;}
```

- Partitionnement (**Skip** (la deuxième partie de la liste , **Take**)

Programmation objet/ LINQ

- Partitionnement (**Skip** (la deuxième partie de la liste , **Take**)
- Exemple les clients ayant passés au moins dix commandes

```
var requeteProjection = from unClient in listeClients
                        orderby unClient.LesCommandes.Count descending
                         select unClient ;
foreach (var unClient in requeteprojection.TakeWhile(unClient =>
unClient.LesCommandes.Count >= 10))
{   Console.WrieLine(unClient.nom) ;}
```

- Jointures et regroupements
- Agrégations

Programmation objet/Accès aux bases de données

Quatre fournisseurs de données sont disponibles dans le Framawork .NET :

- SQL Server
- OLE DB
- ODBC
- Oracle

- Ils proposent quatre classes de base, nécessaires pour le dialogue avec la base de données :
  - La classe Connection permet d'établir une connexion avec le serveur de base de données.
  - La classe Command permet de demander l'exécution d'une instruction ou d'un ensemble d'instructions SQL à un serveur.

Programmation objet/Accès aux bases de données

La classe DataReader procure un accès en lecture seule et défilement, en avant seulement, aux données.

La classe DataAdapter est utilisée pour assurer le transfert des données vers un système de cache local à l'application (le DataSet) et mettre à jour la base de données, en fonction des modifications effectuées localement dans DataSet.

Exemple SQL Server : toutes les classes sont disponible dans l'espace de noms System.Data.SqlClient

Programmation objet/Accès aux bases de données

- Connexion à une base
  - La classe SqlConnection gére une connexion vers un serveur SQL Server.
  - Créer une intance de la classe

    SqlConnection ctn ;
  - Intialiser en appelant le constructeur

    Ctn = new SqlConnection(‘’ Chaîne de connexion’’);
    - Le format de la chaîne de connexion est consttitué d’une série de couples mot-clé/ valeur séparés par des points virgules.
    - **Data Source :** Nom ou l’adresse du serveur, le numéro du port peut être spécifié à la suite du nom ou l’adresse réseau. S’il n’est pas indiqué, le numéro de port est égal à 1433
    - **Initial Catalog** : Nom de la base
    - **Integrated Security :** si cette valeur est positionnée sur false alors un nom d’utilisateur est un mot de passe doivent être fournis dans la chaîne de connexion.
    - **Pwd , User ID**
    - **.....**

Programmation objet/Accès aux bases de données

- Connexion à une base
- Ctn = new SqlConnection(‘’Data Source = localhost; initial Catalog = Northwind ; integrated Security = true’’);
- Exécution d’une commande
  - La classe **SqlCommand** est utilisée pour demander au serveur l’exécution d’une commande
  - Elle peut être instanciée, en utilisant un de ses constructeur ou une instance peut être obtenue par la méthode CreateCommand de la connexion.

```
SqlCommand cmd;
 Cmd = new SqlCommand();
 Cmd.Connection = ctn ;
 Cmd.CommandText = ‘’select * from products’’ ;
```

Programmation objet/Accès aux bases de données

- Exécution d'une commande

```
SqlCommand cmd;
// constructeur surchargé
cmd = new SqlCommand(''select * from products'' , ctn);
```

- Ou SqlCommand cmd;
- cmd = ctn.CreateCommand();
- cmd.CommandText = ''select * from products'' ;

Programmation objet/Accès aux bases de données

- Lecture d'information
  - Une instruction SQL, renvoyant un ensemble d'enregistrement, doit être exécutée par la méthode **ExecuteReade**r. Cette méthode retourne un objet **DataReader** qui va permettre, par la suite, la lecture des informations.

```
SqlConnection ctn;
SqlCommand cmd;
ctn = new SqlConnection();
ctn = new SqlConnection("Data Source=localhost;Initial
                    Catalog=Northwind;Integrated Security=True");
ctn.Open();
cmd = ctn.CreateCommand();
cmd.CommandText = "select count(orderid) from orders where
                customerid='FRANK'";
// si l'instruction SQL ne renvoie qu'une valeur, ExecuteScalar() se charge de l'exécution
Console.WriteLine("le client FRANK à passé {0} commande(s)",
cmd.ExecuteScalar());
```

Programmation objet/Accès aux bases de données

- Lecture d'information

```
SqlCommand cmd;
   SqlConnection ctn;
   SqlDataReader lecteur;
      ctn = new SqlConnection();
      ctn.ConnectionString = "Data Source=localhost;Initial
                        Catalog=Northwind;Integrated Security=true";
      ctn.Open();
      cmd = new SqlCommand();
      cmd.Connection = ctn;
      cmd.CommandText = " select * from categories";
      lecteur = cmd.ExecuteReader();
      while (lecteur.Read())
      {
          Console.WriteLine("numero de la categorie:{0}" + "\t" +
"Nom:{1}", lecteur.GetInt32(0),    lecteur["CategoryName"]);
      }
      lecteur.Close();
      ctn.Close();
```

Programmation objet/Accès aux bases de données

- Modification des informations (méthode ExecuteNonQuery())

```
SqlCommand cmd;
        SqlConnection ctn;
        ctn = new SqlConnection();
        ctn.ConnectionString = "Data Source=localhost;Initial
                        Catalog=Northwind;Integrated Security=true";
           ctn.Open();
           cmd = new SqlCommand();
           cmd.Connection = ctn;
           cmd.CommandText = "Insert into shippers (companyname,phone)
                        values ('DHL','02 40 41 42 43')";
           Console.WriteLine("{0} ligne(s) ajoutée(s) dans la table",
                        cmd.ExecuteNonQuery());
           ctn.Close();
```

Programmation objet/Accès aux bases de données

- Utilisation de paramètres
- Exemple : recherche d'un client par son code

```
SqlCommand cmd;
        SqlConnection ctn;
        SqlDataReader lecteur;
        string codeClient;


            ctn = new SqlConnection();
            ctn.ConnectionString = "Data Source=localhost;Initial
                    Catalog=Northwind;Integrated Security=true";
            ctn.Open();
            cmd = new SqlCommand();
            cmd.Connection = ctn;
            Console.Write("saisir le code du client recherche :");
            codeClient = Console.ReadLine();
            cmd.CommandText = " SELECT * from Customers WHERE CustomerID =
                        '" + codeClient + "'";
            lecteur = cmd.ExecuteReader();
```

```
while (lecteur.Read())
            {
                Console.WriteLine("nom du client:{0}",
                lecteur["ContactName"]);
            }
            lecteur.Close();
            ctn.Close();
```

Programmation objet/Accès aux bases de données

- Utilisation de paramètres
- Soit :

cmd.CommandText = " SELECT * from Customers WHERE CustomerID = **'" + codeClient + "';**

- L'utilisation de paramètre simplifie l'écriture ci-dessus
- Les paramètre peuvent être nommés anonyms.
- Les paramètres anonyme est introduit dans la requête par le caractère ?
- Les paramètres nommés sont spécifiés par le caractère @ suivi du nom du paramètre

Programmation objet/Accès aux bases de données

- Utilisation de paramètres
- Avant l'exécution de la Sql Command, il faut créer les objets SqlParameter et les ajouter à la collection .
- Exemple
- SqlParameter paramCodeClient ;
- paramCodeClient = new SqlParameter(''@code'', codeClient);
- La propriété Direction de la classe SqlParameter indique
- le mode d'utilisation du paramètre:
- Input : si l'information est passée au code SQL pour son exécution
- Output si l'exécution du code SQL qui va modifier la valeur du paramètre
- Ou les deux InputOutput.

Programmation objet/Accès aux bases de données

- Exemple :

```
public static void TestRequeteParam()
   {
      SqlCommand cmd;
      SqlConnection ctn;
      SqlDataReader lecteur;
      string codeClient;
      SqlParameter paramCodeClient;
      SqlParameter paramNbCommandes;


         ctn = new SqlConnection();
         ctn.ConnectionString = "Data Source=localhost;Initial
                                 Catalog=Northwind;Integrated Security=true";
         ctn.Open();
         cmd = new SqlCommand();
         cmd.Connection = ctn;
```

Programmation objet/Accès aux bases de données

- Exemple :

```
Console.Write("saisir le code du client recherche :");
        codeClient = Console.ReadLine();
        cmd.CommandText = " SELECT * from Customers WHERE CustomerID =
                        @Code;select @nbCmd=count(orderid) from
                        orders where customerid=@code";
        paramCodeClient = new SqlParameter("@Code", codeClient);
        paramCodeClient.Direction = ParameterDirection.Input;
        cmd.Parameters.Add(paramCodeClient);
        paramNbCommandes = new SqlParameter("@nbCmd",null);
        paramNbCommandes.Direction = ParameterDirection.Output;
        paramNbCommandes.SqlDbType=SqlDbType.Int;
        cmd.Parameters.Add(paramNbCommandes);
        lecteur = cmd.ExecuteReader();
```

Programmation objet/Accès aux bases de données

- Exemple :

```
while (lecteur.Read())
        {
            Console.WriteLine("nom du client:{0}",
                              lecteur["ContactName"]);
        }
        lecteur.Close();
        Console.WriteLine("ce client a passe {0} commande(s)",
                          cmd.Parameters["@nbCmd"].Value);
        ctn.Close();
    }
```

Programmation objet/Accès aux bases de données

- Utilisation du mode non connecté

  Dans le mode non connecté, la liaison avec le serveur de base de données n'est pas permanente. Il faut donc conserver localement les données sur lesquelles on souhaite travaillller. Les principales classses sont représentées sur le shéma suivant :

  DataSet

  C'est le conteneur de plus haut niveau, il joue le même rôle que la base de données.

  DataTable

  Comme son nom l'indique, c'est l'équivalent d'une table de base de données.

  DataRow

  Cette classe joue le rôle d'un enregistrement (ligne)

  DataColumn

  Cette classe remplace un champ colonne d'une table

Programmation objet/Accès aux bases de données

- Utilisation du mode non connecté

  UniqueConstraint

  C’est équivalent de la clé primaire d’une table

  ForeignKeyConstraint

  C’est l’équivalent de la clé étrangère

  DataRelation

  Represente un lien parent/enfant entre deux DataTable

Programmation objet/Accès aux bases de données

- Utilisation du mode non connecté
- SqlDataAdapter: Représente un ensemble de commandes de données et une connexion à une base de données qui permettent de remplir DataSet et de mettre à jour une base de données SQL Server. Cette classe ne peut pas être héritée.
- Fill : La méthode **Fill** prend comme arguments un **DataSet**à remplir, ainsi qu'un objet **DataTable**, ou le nom du **DataTable** à remplir avec les lignes retournées par **SelectCommand**.

Programmation objet/Accès aux bases de données

- Utilisation du mode non connecté

```
public static void  TestDataSet1()
{
  SqlCommand cmd;
  SqlConnection ctn;
  DataSet ds;
  SqlDataAdapter da;
 ctn = new SqlConnection();
   ctn.ConnectionString = "Data Source=localhost;Initial
                          Catalog=Northwind;Integrated Security=true";
```

Programmation objet/Accès aux bases de données

- Utilisation du mode non connecté

```
public static void  TestDataSet1()
{
  SqlCommand cmd;
  SqlConnection ctn;
  DataSet ds;
  SqlDataAdapter da;
 ctn = new SqlConnection();
   ctn.ConnectionString = "Data Source=localhost;Initial
                          Catalog=Northwind;Integrated Security=true";
```

Programmation objet/Accès aux bases de données

- DataTableMapping, classe
- Contient une description d'une relation mappée entre une table source et DataTable. Cette classe est utilisée par DataAdapter lors du remplissage de DataSet.

```
SqlCommand cmd;
  SqlConnection ctn;
  DataSet ds;
  SqlDataAdapter da;
  DataTableMapping mappage;
  ctn = new SqlConnection();
  ctn.ConnectionString = "Data Source=localhost;Initial
                         Catalog=Northwind;Integrated
Security=true";
  cmd = new SqlCommand();
  cmd.Connection = ctn;
  cmd.CommandText = " SELECT
CustomerId,ContactName,Address,city from
                    Customers";
```

Programmation objet/Accès aux bases de données

- DataTableMapping, classe

```
ds = new DataSet();
 da = new SqlDataAdapter();
 da.SelectCommand = cmd;
 mappage = new DataTableMapping("Customers", "Clients");
 mappage.ColumnMappings.Add("CustomerId", "CodeClient");
 mappage.ColumnMappings.Add("ContactName", "Nom");
 mappage.ColumnMappings.Add("Address", "Adresse");
 mappage.ColumnMappings.Add("city", "Ville");
 da.TableMappings.Add(mappage);
 da.Fill(ds, "Customers");
 foreach ( DataColumn dc in ds.Tables["Clients"].Columns)
 {
    Console.Write(dc.ColumnName + "\t");
 } .
```

Programmation objet/Accès aux bases de données

- lecture dans une DataTable

```
public static void TestLectureDataTable()
{
   SqlCommand cmd;
   SqlConnection ctn;
   DataSet ds;
   SqlDataAdapter da;
   IEnumerator en;

  ctn = new SqlConnection();
   ctn.ConnectionString = "Data Source=localhost;Initial
                          Catalog=Northwind;Integrated Security=true";
  cmd = new SqlCommand();
  cmd.Connection = ctn;
  cmd.CommandText = " SELECT ContactTitle,ContactName from Customers";
```

Programmation objet/Accès aux bases de données

```
ds = new DataSet();
  da = new SqlDataAdapter();
  da.SelectCommand = cmd;
  da.Fill(ds, "Customers");
  // on recupere l'enumerateur sur les lignes de la DataTable
  en = ds.Tables["Customers"].Rows.GetEnumerator();
   // on se replace au debut de la table (par securite)
   en.Reset();
   // on boucle tant que la méthode MoveNext nous indique qu'il reste des lignes
   while (en.MoveNext())
   {
      // on accede aux champs par le nom
      Console.Write(((DataRow)en.Current)["ContactName"] + "\t");
      // ou par le numero
      Console.WriteLine(((DataRow)en.Current)[0]);
    }
   Console.ReadLine(); }
```