



M11 : POO

CS.Net

Formateur : Driouch (cfmoti.driouch@gmail.com)
Etablissement : OFPPT/DRGC/CFMOTI
Version du 04/05/2012

1

Plan

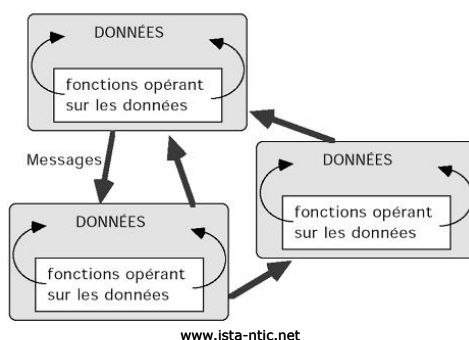
- Notion d'Objet – Classe - Instanciation
- Encapsulation (Propriété et Méthodes d'accès)
- Constructeurs (Constructeur de copie) – Destructeur (Garbage Collector)
- Héritage (Surcharge Méthode/Opérateur – Redéfinition)
- Polymorphisme
- Les Exceptions
- Les Objets Courants (en .Net)
- Objet de collection
- Les Interfaces
- La persistance (Sérialisation binaire, Objet et XML)

www.ista-ntic.net

2

Notion d'objet

- Un objet est une entité cohérente rassemblant des données et du code travaillant sur ces données.



www.ista-ntic.net

3

Classe

- Une classe est une description abstraite d'un objet. Elle peut être considérée en quelque sorte comme un moule...

Véhicule
+Marque : string(idl)
+Puissance fiscale : int
+Vitesse maximale : int
+Vitesse courante : int
+Créer un véhicule()
+Détruire un véhicule()
+Démarrer()
+Accélérer(entrée Taux : int)

Nom de la classe

Description des attributs
ou données membres

Description des méthodes
= code associé

www.ista-ntic.net

4

Instanciation

- Chaque objet correspond à une instance de la classe à laquelle il fait référence.

Véhicule
+Marque : string(idl)
+Puissance fiscale : int
+Vitesse maximale : int
+Vitesse courante : int
+Créer un véhicule()
+Détruire un véhicule()
+Démarrer()
+Accélérer(entrée Taux : int)

Marque = Peugeot
Puissance = 7
Vitesse maximale = 197
Vitesse courante = 98

Marque = Renault
Puissance = 5
Vitesse maximale = 176
Vitesse courante = 65

www.ista-ntic.net

5

Instanciation

- La création d'un objet est constituée de deux phases :
 - Une phase du ressort de la classe : allouer de la mémoire et un contexte d'exécution minimaliste. Méthode de classe
 - Une phase du ressort de l'objet : initialiser ses attributs d'instance. Méthode d'instance
- Dans les langages tels que Java, C++, VB ou C#, ces deux phases ne sont pas différenciées.
- Appel à une méthode spéciale : le constructeur

www.ista-ntic.net

6

Encapsulation

- **Abstraction de données :**
La structure d'un objet n'est pas visible de l'extérieur.
Seule son interface est accessible par le biais de messages invocables par le client.
- **Abstraction procédurale :**
L'invocation d'un message est une opération atomique.
Aucun élément d'information sur les traitements internes mis en œuvre n'est nécessaire.
- **Donc pour chaque objet créer on peut limiter les accès ou les contrôler selon nos besoins pour cet objet.**
 - Intérieur de l'objet protégé
 - Complexité dissimulée
 - Maintenance simplifiée (centralisation)
 - Échanges avec l'extérieur sont codifiés

www.ista-ntic.net

7

Propriétés

- Les propriétés d'un objet sont déclarées, comme des variables, à l'intérieur du bloc contrôlé par le mot clé **class**.

```
Public Class NomDeLaClasse
{ Public TypeDeLaPropriété NomDeLaPropriete;
  // Déclaration des méthodes de l'objet
}
```

- Les propriétés peuvent être déclarées à tout moment à l'intérieur du corps de la classe.
- Chaque déclaration de propriété (Attributs) est construite sous le modèle suivant :

```
Public TypeDeLaPropriété NomDeLaPropriete;
```

- Une propriété peut être initialisée lors de sa déclaration :

```
Public TypeDeLaPropriété NomDeLaPropriete=valeurInitiale;
```
- Les identifiants de propriété par convention commencent par une majuscule.

www.ista-ntic.net

8

Méthodes d'accès

En Programmation Orientée Objet, on évite d'accéder directement aux propriétés par l'opérateur « . ». En effet, cette possibilité ne correspond pas au concept d'encapsulation. Certains langages l'interdisent carrément.

Afin d'implanter **correctement** le concept d'encapsulation, il convient de **verrouiller** l'accès aux propriétés et de les déclarer **private**

L'Accès aux attributs membres peut se faire par des méthodes simple (Getter et Setter) ou par les Property de classe

```
public type nomClient{
    get{
        return iNomClient;
    }
    set{
        iNomClient = value;
    }
}
```

www.ista-ntic.net

9

Constructeur

Quand une instance d'une classe d'objet est créée au moment de l'instanciation d'une variable avec **new**, une fonction particulière est exécutée. Cette fonction s'appelle le **constructeur**. Elle permet, entre autres, d'initialiser chaque instance pour que ses propriétés aient un contenu cohérent.

```
Public Client(int numero, string nom){
    this.IDClient = numero;
    this.NomClient = nom;
}
```

Cela va permettre d'instancier la classe **Client** de la façon suivante :

```
Client cli = New Client(12, "Sharraf");
```

www.ista-ntic.net

10

Constructeur de copie

Le constructeur de copie permet de recopier les propriétés d'un objet existant vers une nouvelle instance de même type.

```
Public Client(Client unClient){
    this.IDClient = unClient.IDClient;
    this.NomClient = unClient.NomClient;
    this.CaClient = unClient.CaClient;
}
```

```
Client oClient1 = new Client();
Client oClient = New Client(oClient1);
```

www.ista-ntic.net

11

Exemple

```
public class Vehicule{
    private string Marque;
    private int Puissance;
    private int VitesseMax;
    private int VitesseCour;
    // Constructeur par default
    public Vehicule(){
        Marque = "Marque inconnu";
        Puissance = 0;
        VitesseMax = 0;
        VitesseCour = 0;
    }
    //Constructeur d'initialisation
    public Vehicule(String M, int P, int VM){
        Marque = M;
        Puissance = P;
        VitesseMax = VM;
        VitesseCour = 0;
    }
    //Constructeur de Copie
    public Vehicule (Vehicule Vh){
        Marque = Vh.PMarque;
        Puissance = Vh.PPuissanceF;
        VitesseMax = Vh.PVitesseM;
        VitesseCour = Vh.PVitesseC;
    }

    public string PMarque {
        get{
            return Marque;
        }
        set{
            Marque = value;
        }
    }
    public int PPuissanceF{
        get{
            return this.Puissance;
        }
        set{
            this.Puissance = value;
        }
    }
    public int PVitesseM{
        get{
            return this.VitesseMax;
        }
        set{
            this.VitesseMax = value;
        }
    }
}
```

www.ista-ntic.net

12

Exemple

```
//méthode d'accès en lecture seul
public int PVitesseC{
    get{
        return this.VitesseCour;
    }
}

public void Accelere(int v)
{
    this.VitesseCour += v;
    if (this.PVitesseC > this.PVitesseM) this.VitesseCour = this.PVitesseM;
}

public override string ToString()
{
    return "Vehicule : " + Convert.ToString(this.PMarque) + " " +
        Convert.ToString(this.PPuissanceF) + " - Vitesse : " +
        Convert.ToString(this.PVitesseC) + "/" + Convert.ToString(this.PVitesseM);
}
}
```

www.ista-ntic.net

13

Programme principale

```
using System;
class Program
{
    static void Main(string[] args)
    {
        Vehicule rn=new Vehicule();
        Console.WriteLine(rn.ToString());
        Vehicule pg = new Vehicule("Peugeot", 7, 187);
        Console.WriteLine(pg.ToString());

        pg.Accelere(60);
        Console.WriteLine(pg.ToString());

        pg.Accelere(80);
        pg.Accelere(60);
        Console.WriteLine(pg.ToString());
        Console.ReadKey();
    }
}
```

www.ista-ntic.net

14

Destructeur

En C# on ne déclenche pas explicitement la destruction d'un objet. Les instances seront détruites par le système lorsqu'elles ne sont plus référencées et qu'il sera nécessaire de récupérer des ressources mémoire. Le programme qui se charge de cette tâche s'appelle le **Garbage Collector** ou, en français, le **ramasse-miettes**.

- Les destructeurs ne peuvent pas être définis dans des struct. Ils sont utilisés uniquement avec les classes.
- Une classe peut posséder un seul destructeur.
- Les destructeurs ne peuvent pas être hérités ou surchargés.
- Les destructeurs ne peuvent pas être appelés. Ils sont appelés automatiquement.
- Un destructeur n'accepte pas de modificateurs ni de paramètres.

~NomClass() // Cette méthode s'appelle **destructeur**.

{ //programme (Exp: libération des ressources)

}

Dans le programme principale on peut forcé le Garbage Collector par la procédure: GC.Collect()

www.ista-ntic.net

15

Propriétés et Méthodes partagé

```
Public class test{
    Public static Type Attribut;
    Public static Type NomMeth(){
        ...
    }
}
```

L'utilisation de l'attribut ou la méthode se fait via la classe directement.

Test.NomMeth Ou test.Attribut sans instanciation.

Exp :

```
Private static int iCompteur;
```

```
Public test(){
    iCompteur += 1;
}
```

```
~test(){
    iCompteur -= 1;
}
```

Si en affiche la valeur de iCompteur en retrouve la nombre d'objet instancier et référencié dans le programme en mémoire.

www.ista-ntic.net

16

Les opérateurs d'accessibilité

<u>Mot Clé</u>	<u>Définition</u>
Public	Accessible partout
Private	Accès dans la class uniquement
Protected	Accès classe et classes dérivées
Internal	Accès Classe - Espace de nom - Assemblage

www.ista-ntic.net

17

Héritage

Le concept d'héritage est l'un des trois principaux fondements de la Programmation Orientée Objet, le premier étant l'encapsulation vu précédemment et le dernier étant le polymorphisme qui sera abordé plus loin dans ce document

L'héritage consiste en la création d'une nouvelle classe dite **classe dérivée** à partir d'une classe existante dite **classe de base** ou **classe parente**.

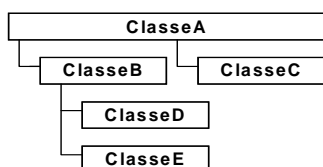
L'héritage permet de :

- **Récupérer** le comportement standard d'une classe d'objet (classe parente) à partir des propriétés et des méthodes définies dans celles-ci.
- **Ajouter** des fonctionnalités supplémentaires en créant de nouvelles propriétés et méthodes dans la classe dérivée.
- **Modifier** le comportement standard d'une classe d'objet (classe parente) en surchargeant certaines méthodes de la classe parente dans la classe dérivée.

www.ista-ntic.net

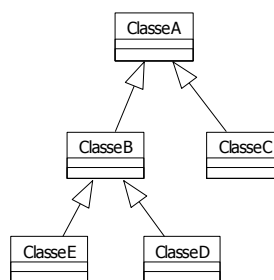
18

Représentation



Le diagramme ci-dessus constitue la représentation graphique de la **hiérarchie de classes** construite à partir de **ClasseA**.

Dans le cadre de la conception orientée objet, la méthode UML (United Modeling Language) propose une autre représentation graphique d'une telle hiérarchie :



www.ista-ntic.net

19

Exemple

```

public class Client
{
    protected string NomClient;
    protected double CAClient;
    public Client()
    {
    }
    public Client(string nom, double ca)
    {
        this.NomClient = nom;
        this.CAClient = ca;
    }
    public string Nom
    {
        get
        {
            return this.NomClient;
        }
        set
        {
            this.NomClient = value;
        }
    }
    public double CA
    {
        set
        {
            this.CAClient = value;
        }
    }
    public virtual double finance()
    {
        return this.CAClient;
    }
    public override string ToString()
    {
        return " Nom: " + this.Nom + " CA : " + Convert.ToString(this.finance());
    }
}
  
```

www.ista-ntic.net

20

Exemple

```
public class Grossiste:Client
{
    private double TxRemiseClient;
    public Grossiste():base(){
    }
    public Grossiste(string nom, double ca, double rm):base(nom,ca){
        this.TxRemise = rm;
    }
    public double TxRemise{
        get{ return this.TxRemiseClient;
        }
        set{ this.TxRemiseClient = value;
        }
    }
    public double calrm(){
        return this.CAClient * this.TxRemise;
    }
    public override double finance()
    { return this.CAClient * (1 - this.TxRemise);
    }
}
```

www.ista-ntic.net

21

Surcharge (méthodes/Opérateur)

```
public int Add(int A, int B){
    return A+B;}
public string Add(string A, string B){
    return A+B;}
```

La surcharge des méthodes avec le même nom, se fait par le changement de leur signature (le nombre de paramètre, le type des paramètre)

En peut aussi avoir une surcharge des opérateurs (+,-,x,/ ...):

```
public class Complex {
    public int real;
    public int imaginary;
    public Complex(int real, int imaginary) //constructor
    { this.real = real; this.imaginary = imaginary; }
    public static Complex operator +(Complex c1, Complex c2) {
        //test si c1 et c2 ne sont pas vide (null).
        return new Complex(c1.real + c2.real, c1.imaginary + c2.imaginary);
    }
}
```

www.ista-ntic.net

22

Redéfinition des méthodes

Méthode ~NomClass: le destructeur vu précédemment.

Méthode ToString():

```

Pour la classe fraction
public override string ToString(){
    return numerateur + "/" + denominateur;
}

```

Surcharge Méthode Equals:

```

public override bool Equals(object obj){
    return base.Equals(obj);
}

public bool Equals(Fraction fr){
    double c1, c2; //test si fr est null (fr==null)
    c1 = Me.numerateur * (fr.denominateur);
    c2 = Me.denominateur * (fr.numerateur);
    If (c1 = c2){
        return true;}
    Else{
        return False;}
}

```

www.ista-ntic.net

23

Complément héritage(Modificateur)

- **Abstract (Class):** Ce mot clé indique qu'une classe ne peut être instancier, et qu'elle ne peut donc être utilisée que comme classe de base, une classe abstret.
- **Abstract (Méthode):** utilisé pour les méthodes qu'on veut forcé leur redéfinition dans les classes dérivé, en défini seulement la signature sans corps.
- **Sealed (Class):** A l'inverse de abstract, ce mot-clé indique que cette classe ne peut pas être héritée, et peut etres instancier, c'est-à-dire servir de classe de base.
- **Virtual (Méthode):** ce mot-clé indique que cette méthode peut être redéfinie dans une classe dérivé.
- **Sealed (Méthode):** A l'inverse, ce mot-clé indique que cette méthode ne peut être redéfinie dans une classe dérivé.
- **New (Méthode-Attribut):** le mot clé new masque explicitement un membre hérité d'une classe de base(avec signateur différente et même nom). Lorsque vous masquez un membre hérité, la version dérivée du membre remplace la version de classe de base.

www.ista-ntic.net

24

Polymorphisme

Le polymorphisme est un mécanisme via lequel un objet peut prendre plus d'une forme. Par exemple, si vous avez une classe de base nommée Client, une référence de type Client peut être utilisée pour contenir un objet de n'importe laquelle de ses classes dérivées. Quand vous appelez une méthode à partir de votre objet, le système déterminera automatiquement le type de l'objet afin d'appeler la méthode appropriée.

```
Client x;
Client cl = New Client("Ali", 1000);
Grossiste clg = New Grossiste("Ahmed", 7000, 0.2);
x = cl;
Console.WriteLine(x.ToString());
x = clg;
Console.WriteLine(x.ToString());
```

www.ista-ntic.net

25

Implémentation des Interfaces

Une interface est une collection de prototypes représentant les membres (propriétés, procédures et événements) que l'interface encapsule. Les interfaces contiennent uniquement les **déclarations** des membres, les classes et les structures implémentent ces membres

Exp:

```
interface TestInterface
{
    double Propriete1{get;}
    double Methode1(int X);
}
class ClassImplementation: TestInterface
{
    private double Attribut1;
    public double Propriete1 {
        get { return Attribut1; }
        set { Attribut1 = value; }
    }
    public double Methode1(int X) {
        Attribut1 = Math.Sqrt(X);
        return Attribut1;
    }
}
```

www.ista-ntic.net

26

Les Exceptions(Gestion Erreur)

Il y a plusieurs types d'erreurs :

- **Les erreurs de syntaxe** : Elle surviennent en mode conception quand on tape le code.
Exp: A+1=B //Erreur d'affectation
2 {... et un seul }
- **Les erreurs de logique** : quand la conception du programme (logiciel) qui est incorrect, des données justes nous donne des résultats faut. Donc il faut revoir la conception.
- **Les erreurs d'exécution** : Elle surviennent en mode Run ou lors de l'utilisation de l'exécutable, une instruction ne peut pas être effectuée. Le logiciel s'arrête brutalement, c'est très gênant!! Pour l'utilisateur c'est un 'BUG'
division par zéro
string a(3) ; → a[5]=« A »
Soit une erreur de l'utilisateur, Il faut toujours vérifier ce que fait l'utilisateur et prévoir toutes les possibilités.
Exp: On lui demande de taper un chiffre, il tape une lettre ou rien puis valide

Pour éviter ces derniers il faut capté l'erreur avec Try ... Catch ... finally

Syntaxe :

```
int x = 0;
try { int y = 100/x; }
catch (ArithmeticException e) { Console.WriteLine("ArithmeticException Handler: {0}",
e.ToString()); }
catch (Exception e) { Console.WriteLine("Generic Exception Handler: {0}",
e.ToString()); }
finally { Console.WriteLine("Executing finally block."); }
```

www.ista-ntic.net

27

Les Exceptions

- Cette classe (Exception) est la classe de base pour toutes les exceptions. Lorsqu'une erreur se produit, le système ou l'application en cours d'exécution la signale en levant une exception qui contient des informations sur l'erreur. Une fois levée, une exception est gérée par l'application ou par le gestionnaire d'exceptions par défaut.
- Exp:
 - ArithmeticException
 - DivideByZeroException
 - NotFiniteNumberException
 - OverflowException

www.ista-ntic.net

28

Les Exceptions Personnalisé

On peut créer une nouvelle classe d'exception qui hérite de la classe Exception avec définition des constructeurs, et on peut faire appel à cet Exception par le mot clé throw.

```
class Personne
{
    private string _Nom;
    private int _Age;
    public string Nom{
        get{return _Nom;}
        set{_Nom = value;}
    }
    public int Age {
        get{return _Age;}
        set{
            if (value < 0)
                throw new AgeException("Erreur : Age Négatif");
            else
                _Age = value;
        }
    }
    public Personne(string No, int Ag){
        this.Nom = No;
        this.Age = Ag;
    }
    public override string ToString(){
        return this.Nom + " (Age : " + this.Age.ToString() +
        ")";
    }
}
```

```
class AgeException:Exception
{
    public AgeException() {
    }
    public AgeException(string message):base(message) {
    }
    public AgeException(string message, Exception Inner)
        : base(message, Inner)
    {
    }
}
```

Programme Principal :

```
Personne p;
try{
    p = new Personne("Ali", 0);
    p.Age = -5;
    //p.Age = "A";
}
catch (AgeException ex){
    Console.WriteLine(ex.ToString());}
catch (Exception ex){
    Console.WriteLine("Error : " + ex.ToString());}
Console.ReadKey();
```

www.ista-ntic.net

29

Objets String

Dim str as string

Avec ça on a un objet str de type string et chaque objet a des méthodes et des attributs

Length() : Taille d'une chaîne en nombre de caractère.

ToCharArray() : retourne un tableau de caractère.

ToUpper() : Mettre en majuscules une chaîne de caractère.

ToLower() : transforme par contre la chaîne en minuscule.

Trim() : Permet de supprimer des caractères en début et fin de chaîne.

Insert (N,str1) : Insère à la position N une sous chaîne str1.

Remove(N,L) : Supprime la sous-chaîne à partir de la position N et de longueur L.

Replace (str1,str2) : Remplace partout dans une chaîne de départ, une chaîne par une autre.

IndexOf & LastIndexOf : Indique le numéro du caractère, la position (la première occurrence) ou une chaîne à chercher est trouvée dans une autre.

Substring(n,l) : Extrait une partie d'une chaîne

...

Exp :

string str = « Bonjour »

Console.WriteLine(str.Length); → 7

Console.WriteLine(str.Replace(« jour », « soir »)); → Bonsoir

www.ista-ntic.net

30

Clone	Retourne une référence à cette instance de <u>String</u> .
Compare	Surchargé. Compare deux objets <u>String</u> spécifiés et retourne un entier qui indique la relation entre ces deux objets dans l'ordre de tri.
Concat	Surchargé. Concatène une ou plusieurs instances de <u>String</u> ou les représentations <u>String</u> des valeurs d'une ou de plusieurs instances de <u>Object</u> .
Contains	Retourne une valeur qui indique si l'objet <u>String</u> spécifié apparaît dans cette chaîne.
Copy	Crée une nouvelle instance de <u>String</u> ayant la même valeur qu'un <u>String</u> spécifié.
CopyTo	Copie un nombre spécifié de caractères à partir d'une position définie dans cette instance vers une position spécifiée dans un tableau de caractères Unicode.
EndsWith	Surchargé. Détermine si la fin d'une instance de <u>String</u> correspond à une chaîne spécifiée.
Equals	Surchargé. Détermine si deux objets <u>String</u> ont la même valeur.
Finalize	Autorise <u>Object</u> à tenter de libérer des ressources et d'exécuter d'autres opérations de nettoyage avant que <u>Object</u> soit récupéré par l'opération garbage collection. (Hérité de <u>Object</u> .)
Format	Surchargé. Remplace chaque élément de mise en forme dans un <u>String</u> spécifié par l'équivalent textuel de la valeur d'un objet correspondant.
IndexOf	Surchargé. Signale l'index de la première occurrence d'un <u>String</u> , ou d'un ou de plusieurs caractères de cette chaîne.
Insert	Insère une instance spécifiée de <u>String</u> au point d'index indiqué dans cette instance.
IsNullOrEmpty	Indique si l'objet <u>String</u> spécifié est null ou une chaîne <u>Empty</u> .
Join	Surchargé. Concatène un séparateur spécifié de <u>String</u> entre chaque élément d'un tableau <u>String</u> indiqué, donnant lieu à une chaîne concaténée unique.
LastIndexOf	Surchargé. Signale le point d'index de la dernière occurrence d'un caractère Unicode spécifié ou <u>String</u> dans cette instance.
PadLeft	Surchargé. Aligne les caractères de cette instance à droite et, à gauche, remplit en ajoutant des espaces ou un caractère Unicode spécifié pour une longueur totale spécifiée.
PadRight	Surchargé. Aligne les caractères de cette instance à gauche et, à droite, remplit en ajoutant des espaces ou un caractère Unicode spécifié pour une longueur totale spécifiée.
Remove	Surchargé. Supprime de cette instance le nombre spécifié de caractères.
Replace	Surchargé. Remplace toutes les occurrences d'un caractère Unicode spécifié ou <u>String</u> dans cette instance par un autre caractère Unicode spécifié ou <u>String</u> .
Split	Surchargé. Retourne un tableau de chaînes qui contient les sous-chaînes de cette instance, délimitées par les éléments d'une chaîne ou d'un tableau de caractères Unicode spécifié.
StartsWith	Surchargé. Détermine si le début d'une instance de <u>String</u> correspond à une chaîne spécifiée.
Substring	Surchargé. Récupère une sous-chaîne de cette instance.
ToCharArray	Surchargé. Copie les caractères dans cette instance vers un tableau de caractères Unicode.
ToLower	Surchargé. Retourne une copie de <u>String</u> en minuscules.
ToString	Surchargé. Convertit la valeur de cette instance en <u>String</u> .
ToUpper	Surchargé. Retourne une copie de <u>String</u> en majuscules.
Trim	Surchargé. Supprime, de l'objet <u>String</u> actuel, toutes les www.ista-ntic.net début et à la fin d'un jeu de caractères spécifiés. 31
TrimEnd	Supprime, de l'objet <u>String</u> actuel, toutes les occurrences situées à la fin d'un jeu de caractères spécifiés dans un tableau.
TrimStart	Supprime, de l'objet <u>String</u> actuel, toutes les occurrences situées au début d'un jeu de caractères spécifiés dans un tableau.

Exercices

- Écrire un programme qui lit une chaîne STR au clavier et un caractère C de cette chaîne, qui va chercher le nombre d'occurrence et le supprime dans la chaîne STR.
- Inverser une chaîne de caractère.
- Remplacer le premier caractère de chaque mot dans une phrase par le majuscule.

L'Objet Regex

La classe `Regex` (`System.Text.RegularExpressions`) permet l'utilisation d'expressions régulières. Celles-ci permettent de tester le format d'une chaîne de caractères. Ainsi on peut vérifier qu'une chaîne représentant une date est bien au format `jj/mm/aa`. On utilise pour cela un modèle et on compare la chaîne à ce modèle. Ainsi dans cet exemple, `j`, `m` et `a` doivent être des chiffres. Le modèle d'un format de date valide est alors `"\d\d/\d\d/\d\d"` où le symbole `\d` désigne un chiffre.

```
// une expression régulière modèle
string modèle1 = @"\d\d/\d\d/\d\d";
Regex regex1 = new Regex(modèle1);
// comparer un exemplaire au modèle
string exp1 = " 123 ";
if (regex1.IsMatch(exp1))
    Console.WriteLine("[ " + exp1 + " ] correspond au modèle [ " + modèle1 + " ]");
else
    Console.WriteLine("[ " + exp1 + " ] ne correspond pas au modèle [ " + modèle1 + " ]");

string exp2 = "01/02/10"; // " 123 ";
if (regex1.IsMatch(exp2))
    Console.WriteLine("[ " + exp2 + " ] correspond au modèle [ " + modèle1 + " ]");
else
    Console.WriteLine("[ " + exp2 + " ] ne correspond pas au modèle [ " + modèle1 + " ]");
Console.ReadKey();
```

www.ista-ntic.net

33

Caractère	Description
\	Marque le caractère suivant comme caractère spécial ou littéral. Par exemple, "n" correspond au caractère "n", "\n" correspond à un caractère de nouvelle ligne. La séquence "\\" correspond à "\", tandis que \" correspond à ".
^	Correspond au début de la saisie.
\$	Correspond à la fin de la saisie.
*	Correspond au caractère précédent zéro fois ou plusieurs fois. Ainsi, "zo*" correspond à "z" ou à "zoo".
+	Correspond au caractère précédent une ou plusieurs fois. Ainsi, "zo+" correspond à "zoo", mais pas à "z".
?	Correspond au caractère précédent zéro ou une fois. Par exemple, "a?ve?" correspond à "ve" dans "lever".
.	Correspond à tout caractère unique, sauf le caractère de nouvelle ligne.
(modèle)	Recherche le <i>modèle</i> et mémorise la correspondance. La sous chaîne correspondante peut être extraite de la collection Matches obtenue, à l'aide d'Item [0]...[n]. Pour trouver des correspondances avec des caractères entre parenthèses (), utilisez \" ou \".
x y	Correspond soit à x soit à y. Par exemple, "zi foot" correspond à "z" ou à "foot". "(zi fo)o" correspond à "zoo" ou à "foo".
{n}	n est un nombre entier non négatif. Correspond exactement à n fois le caractère. Par exemple, "o{2}" ne correspond pas à "o" dans "Bob", mais aux deux premiers "o" dans "fooooo".
{n,}	n est un entier non négatif. Correspond à au moins n fois le caractère. Par exemple, "o{2,}" ne correspond pas à "o" dans "Bob", mais à tous les "o" dans "fooooo". "o{1,}" équivaut à "o+" et "o{0,}" équivaut à "o*".
{n,m}	m et n sont des entiers non négatifs. Correspond à au moins n et à au plus m fois le caractère. Par exemple, "o{1,3}" correspond aux trois premiers "o" dans "fooooo" et "o{0,1}" équivaut à "o?".
[xyz]	Jeu de caractères. Correspond à l'un des caractères indiqués. Par exemple, "[abc]" correspond à "a" dans "plat".
[^xyz]	Jeu de caractères négatif. Correspond à tout caractère non indiqué. Par exemple, "[^abc]" correspond à "p" dans "plat".
[a-z]	Plage de caractères. Correspond à tout caractère dans la série spécifiée. Par exemple, "[a-z]" correspond à tout caractère alphabétique minuscule compris entre "a" et "z".
[^m-z]	Plage de caractères négative. Correspond à tout caractère ne se trouvant pas dans la série spécifiée. Par exemple, "[^mz]" correspond à tout caractère ne se trouvant pas entre "m" et "z".
\b	Correspond à une limite représentant un mot, autrement dit, à la position entre un mot et un espace. Par exemple, "er\b" correspond à "er" dans "lever", mais pas à "er" dans "verbe".

www.ista-ntic.net

34

\B	Correspond à une limite ne représentant pas un mot. "en*t(B" correspond à "ent" dans "bien entendu".
\d	Correspond à un caractère représentant un chiffre. Équivalent à [0-9].
\D	Correspond à un caractère ne représentant pas un chiffre. Équivalent à [^0-9].
\f	Correspond à un caractère de saut de page.
\n	Correspond à un caractère de nouvelle ligne
\r	Correspond à un caractère de retour chariot.
\s	Correspond à tout espace blanc, y compris l'espace, la tabulation, le saut de page, etc. Équivalent à "[\f\n\r\t\v]".
\S	Correspond à tout caractère d'espace non blanc. Équivalent à "[^\f\n\r\t\v]".
\t	Correspond à un caractère de tabulation
\v	Correspond à un caractère de tabulation verticale.
\w	Correspond à tout caractère représentant un mot et incluant un trait de soulignement. Équivalent à "[A-Za-z0-9_]".
\W	Correspond à tout caractère ne représentant pas un mot. Équivalent à "[^A-Za-z0-9_]".
\num	Correspond à <i>num</i> , où <i>num</i> est un entier positif. Fait référence aux correspondances mémorisées. Par exemple, "(.)1" correspond à deux caractères identiques consécutifs.
\n	Correspond à <i>n</i> , où <i>n</i> est une valeur d'échappement octale. Les valeurs d'échappement octales doivent comprendre 1, 2 ou 3 chiffres. Par exemple, "\11" et "\011" correspondent tous les deux à un caractère de tabulation. "\0011" équivaut à "\001" & "1". Les valeurs d'échappement octales ne doivent pas excéder 256. Si c'était le cas, seuls les deux premiers chiffres seraient pris en compte dans l'expression. Permet d'utiliser les codes ASCII dans des expressions régulières
\xn	Correspond à <i>n</i> , où <i>n</i> est une valeur d'échappement hexadécimale. Les valeurs d'échappement hexadécimales doivent comprendre deux chiffres obligatoirement. Par exemple, "\x41" correspond à "A". "\x041" équivaut à "\x04" & "1". Permet d'utiliser les codes ASCII dans des expressions régulières

www.ista-ntic.net

35

Regex (exemple)

modèle	signification
\c	un chiffre
\c?	0 ou 1 chiffre
\c*	0 ou davantage de chiffre
\c+	1 ou davantage de chiffre
\c{2}	2 chiffres
\c{3,}	au moins 3 chiffres
\c{5,7}	entre 5 et 7 chiffres

modèle	signification
^modèle	le modèle commence la chaîne
modèle\$	le modèle finit la chaîne
^modèle\$	le modèle commence et finit la chaîne
modèle	le modèle est cherché partout dans la chaîne en commençant par le début de celle-ci

chaîne recherchée	modèle
une date au format jj/mm/aa	\d{2}/\d{2}/\d{2}
une heure au format hh:mm:ss	\d{2}:\d{2}:\d{2}
un nombre entier non signé	\d+
un suite d'espaces éventuellement vide	\s*
un nombre entier non signé qui peut être précédé ou suivi d'espaces	\s*\d+\s*
un nombre entier qui peut être signé et précédé ou suivi d'espaces	\s*([+ -]?\d+)\s*
un nombre réel non signé qui peut être précédé ou suivi d'espaces	\s*\d+(\.\d+)?\s*
un nombre réel qui peut être signé et précédé ou suivi d'espaces	\s*([+ -]?\d+(\.\d+)?)\s*
une chaîne contenant le mot <i>jour</i>	\bjour\b

chaîne recherchée	modèle
une chaîne se terminant par un point d'exclamation	!\$
une chaîne se terminant par un point	\.\$
une chaîne commençant par la séquence //	^//
une chaîne ne comportant qu'un mot éventuellement suivi ou précédé d'espaces	^\s*\w+\s*\$
une chaîne ne comportant deux mot éventuellement suivis ou précédés d'espaces	^\s*\w+\s*\w+\s*\$
une chaîne contenant le mot <i>jour</i>	\bjour\b

www.ista-ntic.net

36

Les collections

Le .NET Framework fournit des classes spécialisées pour le stockage et la récupération des données. Ces classes fournissent la prise en charge des piles, files d'attente, listes et tables de hachage. La plupart des classes de collection implémentent les mêmes interfaces.

L'usage de Collection est une alternative aux tableaux. Fait partie de l'espace de nom System.Collections ou System.Collections.Generic

Une collection fonctionne plutôt comme un groupe d'éléments dans laquelle il est possible d'ajouter ou d'enlever un élément à n'importe quel endroit sans avoir à se préoccuper de sa taille ni où se trouve l'élément.

Le nombre d'élément n'est pas défini au départ comme dans un tableau. Dans une collection il n'y a que les éléments que l'on a ajoutés. Les éléments sont repérés grâce à un index ou avec une Clé unique

- Type collection Array : Décrit les fonctionnalités des tableaux qui leur permettent d'être traités comme des collections.
- Types collection ArrayList et List : Décrit les fonctionnalités des listes génériques et non génériques, qui constituent les types de collection les plus fréquemment utilisés.
- Types collection Hashtable et Dictionary : Décrit les fonctionnalités des types de dictionnaires génériques et non génériques basés sur le hachage.
- Types collection SortedList et SortedDictionary : Décrit les types de dictionnaire triés et les types hybrides qui associent les fonctionnalités d'un dictionnaire à celles d'une liste.
- Types collection Queue : Décrit les fonctionnalités des files d'attente génériques et non génériques.
- Types collection Stack : Décrit les fonctionnalités des piles génériques et non génériques.

www.ista-ntic.net

37

Array

La classe **Array** : Fournit des méthodes pour la création, la manipulation, la recherche ainsi que le tri des tableaux et sert de classe de base pour tous les tableaux.

```
→ static void Main(string[] args)
{ // Création et initialisation d'un tableau d'entier et un tableau d'objet
  int[] myIntArray = new int[5] { 1, 2, 3, 4, 5 };
  Object[] myObjArray = new Object[5] { 26, 27, 28, 29, 30 };

  // Affichage des valeurs initiales pour les deux tableaux.
  Console.WriteLine("Initialement,");
  Console.WriteLine("Tableau d'entier:"); PrintValues(myIntArray);
  Console.WriteLine("Tableau d'Objet:"); PrintValues(myObjArray);
  // Copie les deux premiers éléments du premier tableau dans le deuxième.
  Array.Copy(myIntArray, myObjArray, 2);
  // Prints the values of the modified arrays.
  Console.WriteLine("\nAprès la fin du copie,");
  Console.WriteLine("Tableau d'entier:"); PrintValues(myIntArray);
  Console.WriteLine("Tableau d'Objet:"); PrintValues(myObjArray);

  // Copie les deux derniers éléments du premier tableau dans le deuxième.
  Array.Copy(myObjArray, myObjArray.GetUpperBound(0) - 1, myIntArray, myIntArray.GetUpperBound(0) - 1, 2);
  //GetUpperBound : retourne la position du dernier élément de la dimension spécifiée
  // Prints the values of the modified arrays.
  Console.WriteLine("\nAprès la copie des deux derniers éléments de l'Objet dans l'entier,");
  Console.WriteLine("Tableau d'entier:"); PrintValues(myIntArray);
  Console.WriteLine("Tableau d'Objet:"); PrintValues(myObjArray);
}

→ public static void PrintValues(Object[] myArr)
{ foreach (Object i in myArr)
  { Console.WriteLine(i); }
  Console.WriteLine(); }

→ public static void PrintValues(int[] myArr)
{ foreach (int i in myArr)
  { Console.WriteLine(i); }
  Console.WriteLine(); }
```

www.ista-ntic.net

38

ArrayList

```
using System;
using System.Collections;
static void Main(string[] args)
{
    // Creates and initializes a new ArrayList.
    ArrayList myAL = new ArrayList();
    myAL.Add("Hello");
    myAL.Add("World");
    myAL.Add("!");

    // Displays the properties and values of the ArrayList.
    Console.WriteLine("myAL");
    Console.WriteLine("    Count:  {0}", myAL.Count);
    Console.WriteLine("    Capacity: {0}", myAL.Capacity);
    //Capacity : Obtient ou définit le nombre d'éléments que ArrayList peut contenir
    Console.WriteLine("    Values:");
    PrintValues(myAL);
    Console.ReadKey();
}
public static void PrintValues(IEnumerable myList)
{
    foreach (Object obj in myList)
        Console.WriteLine("    {0}", obj);
}
```

www.ista-ntic.net

39

Hashtable

Représente une collection de paires clé/valeur qui sont organisées en fonction du code de hachage de la clé.

clé	valeur
clé1	valeur1
clé2	valeur2
..	...

Nom	Description
Add	Ajoute un élément avec la clé et la valeur spécifiées dans Hashtable.
Clear	Supprime tous les éléments de Hashtable.
Contains	Détermine si Hashtable contient une clé spécifique.
ContainsKey	Détermine si Hashtable contient une clé spécifique.
ContainsValue	Détermine si Hashtable contient une valeur spécifique.
Equals	Détermine si l'objet Object spécifié est égal à l'objet Object en cours. (Hérité de Object.)
GetType	Obtient le Type de l'instance actuelle. (Hérité de Object.)
KeyEquals	Compare un Object spécifique avec une clé spécifique dans Hashtable.
Remove	Supprime de Hashtable l'élément ayant la clé spécifiée.
ToString	Retourne un String qui représente le Object en cours. (Hérité de Object.)
Count	Obtient le nombre de paires clé/valeur contenues dans Hashtable.
Item	Obtient ou définit la valeur associée à la clé spécifiée.
Keys	Obtient ICollection contenant les clés de Hashtable.
Values	Obtient ICollection contenant les valeurs de Hashtable.

www.ista-ntic.net

40

HashTable (Exp)

```
static void Main(string[] args)
{ // Creation et initiation d'un nouveau Hashtable.
  Hashtable myHT = new Hashtable();
  myHT.Add(0, "zero"); myHT.Add(1, "one");
  myHT.Add(2, "two"); myHT.Add(3, "three");
  Console.WriteLine("le Hashtable contient les valeur suivant:");
  PrintIndexAndKeysAndValues(myHT); // Affichage des valeurs du Hashtable.
  string rep1= "est dans le Hashtable";
  string rep2= " n'est pas dans le Hashtable";
  // Recherche d'une Clé spécifique.
  int myKey = 2;
  Console.WriteLine("la Clé \"{0}\" {1}.", myKey, myHT.ContainsKey(myKey) ? rep1:rep2);
  myKey = 6;
  Console.WriteLine("la Clé \"{0}\" {1}.", myKey, myHT.ContainsKey(myKey) ? rep1 : rep2);
  String myValue = "three"; // Recherche d'une valeur spécifique.
  Console.WriteLine("la valeur \"{0}\" {1}.", myValue, myHT.ContainsValue(myValue) ? rep1 : rep2);
  myValue = "nine";
  Console.WriteLine("la valeur \"{0}\" {1}.", myValue, myHT.ContainsValue(myValue) ? rep1 : rep2);
  Console.ReadKey();}

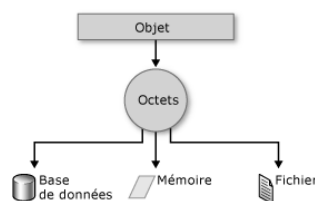
public static void PrintIndexAndKeysAndValues(Hashtable myHT)
{ int i = 0;
  Console.WriteLine("\tINDEX-\tKEY-\tVALUE-");
  foreach (DictionaryEntry de in myHT)
  { Console.WriteLine("\t{0}:\t{1}\t{2}", i++, de.Key, de.Value);
    Console.WriteLine();}
```

www.ista-ntic.net

41

La persistance des objets

- Cette illustration affiche le processus global de la sérialisation :
- L'objet est sérialisé à un flux qui contient non seulement les données, mais également des informations sur le type d'objet, notamment sa version, sa culture et son nom d'assembly. À partir de ce flux, il peut être stocké dans une base de données, dans un fichier ou en mémoire.



www.ista-ntic.net

42

Sérialisation binaire et XML

■ **Sérialisation binaire** : La sérialisation binaire utilise le codage binaire afin de produire une sérialisation compacte destinée notamment au stockage ou au flux réseau socket. Il n'est pas convenable de faire passer les données dans un pare-feu mais les performances sont meilleures lors du stockage des données. **(Voir le Cours LPS)**

■ **Sérialisation XML** : La sérialisation XML sérialise les champs et les propriétés publics d'un objet, ou les paramètres et valeurs de retour des méthodes, en un flux XML conforme à un document de langage XSD (XML Schema Definition) spécifique. La sérialisation XML favorise des classes fortement typées avec des propriétés et des champs publics convertis en XML. System.Xml.Serialization contient les classes nécessaires pour la sérialisation et la désérialisation XML

www.ista-ntic.net

43

Sérialisation binaire (Exemple)

Pour lire ou écrire dans un fichier, on utilise les deux objets StreamReader et StreamWriter dans l'espace de nom System.IO. Soit la classe Client suivante:

```
public class Client
{
    protected int id;
    protected string NomClient;
    protected double CAClient;
    public Client() { }
    public Client(int id, string nom, double ca)
    {
        this.ID = id;
        this.NomClient = nom;
        this.CAClient = ca;
    }
    public int ID {
        get { return id; }
        set { id = value; }
    }
    public string Nom{
        get{return this.NomClient;}
        set{this.NomClient = value;}
    }
    public double CA{
        set{this.CAClient = value;}
    }
    public virtual double finance() {
        return this.CAClient;
    }
    public override string ToString() {
        return this.ID.ToString() + "-> Nom: " + this.Nom + " CA : " + Convert.ToString(this.finance());
    }
    public string ToFile(){
        return this.ID.ToString() + ";" + this.Nom + ";" + Convert.ToString(this.finance());
    }
}
```

www.ista-ntic.net

44

Sérialisation binaire (Exemple)

```

class Program
{
    private static Hashtable HtClient=new Hashtable();
    private static Client Clt;
    static void Main(string[] args)
    {
        try
        {
            Load();
            foreach (Client Clt in HtClient.Values )
                Console.WriteLine(Clt.ToString());
            Save();
            Console.ReadLine();
        }
        catch (Exception e) {
            Console.WriteLine(" Erreur : " + e.Message);}
    }
    private static void Save(){
        StreamWriter fluxInfos = null; // le fichier texte
logique
        try{
            // création du fichier texte
            fluxInfos = new StreamWriter("Client.dat",
false);
            fluxInfos.AutoFlush = true;
            foreach (Client Clt in HtClient.Values)
                fluxInfos.WriteLine(Clt.ToString());
        }
        catch(Exception e){
            Console.WriteLine("Erreur: " + e.Message);}
        finally{
            try{ fluxInfos.Close();}
            catch{}
        }
    }
}

private static void Load(){
    StreamReader fluxInfos = null; // le fichier texte logique
    string ligne = "";
    string[] Tligne = null;
    try{
        // création du fichier texte
        fluxInfos = new StreamReader("Client.dat");
        do{
            ligne = fluxInfos.ReadLine();
            if (ligne != null)
            {
                Tligne = ligne.Split(';');
                if (Tligne.Length == 3)
                {
                    Clt = new Client(Convert.ToInt32(Tligne[0]),
Tligne[1], Convert.ToDouble(Tligne[2]));
                    HtClient.Add(Clt.ID, Clt);
                }
            }
        } while (ligne != null);
    }
    catch (Exception e){
        Console.WriteLine("Erreur : " + e.Message);}
    finally{
        try{
            fluxInfos.Close();
        }
        catch{}
    }
}

```

www.ista-ntic.net

45

Sérialisation Objet (Exemple)

```

[Serializable()]
public class Client
{
    protected int id;
    protected string NomClient;
    protected double CAClient;
    public Client() { }
    public Client(int id, string nom, double ca)
    {
        this.ID = id;
        this.NomClient = nom;
        this.CAClient = ca;
    }
    public int ID {
        get { return id; }
        set { id = value; }
    }
    public string Nom{
        get{return this.NomClient;}
        set{this.NomClient = value;}
    }
    public double CA{
        set{this.CAClient = value;}
    }
    public virtual double finance() {
        return this.CAClient;
    }
    public override string ToString() {
        return this.ID.ToString() + "-> Nom: " + this.Nom + " CA : " + Convert.ToString(this.finance());
    }
    public string ToFile(){
        return this.ID.ToString() + ";" + this.Nom + ";" + Convert.ToString(this.finance());
    }
}

```

www.ista-ntic.net

46

Sérialisation Objet (Exemple)

```

class Program
{
    private static Hashtable HtClient=new Hashtable();

    static void Main(string[] args)
    {
        try
        {
            if(File.Exists("Client.dat")){
                Deserialize();
            }
            else
            {
                HtClient.Add(0, new Client(0, "Ali", 200));
                HtClient.Add(1, new Client(1, "Ahmed", 140));
                HtClient.Add(2, new Client(2, "Imane", 530));
            }
            foreach (Client Clt in HtClient.Values )
            {
                Console.WriteLine(Clt.ToString());
                Serialize();
                Console.ReadLine();
            }
        }
        catch (Exception e) {
            Console.WriteLine("Erreur : " + e.Message);
        }
    }

    private static void Deserialize()
    {
        FileStream fluxInfos = null; // le fichier texte logique
        try{
            // création du fichier texte
            fluxInfos = new FileStream("Client.dat",
            FileMode.Open);
            BinaryFormatter formatter = new BinaryFormatter();

            HtClient =
            (Hashtable)formatter.Deserialize(fluxInfos);
        }
        catch(Exception e){
            Console.WriteLine("Erreur : " + e.Message);
        }
        finally{
            try{fluxInfos.Close();}
            catch{}
        }
    }

    private static void Serialize()
    {
        FileStream fluxInfos = null; // le fichier texte logique
        try{
            // création du fichier texte
            fluxInfos = new FileStream("Client.dat", FileMode.Create);

            BinaryFormatter formatter = new BinaryFormatter();

            formatter.Serialize(fluxInfos, HtClient);
        }
        catch (SerializationException e){
            Console.WriteLine("Erreur : " + e.Message);
        }
        finally{
            try{fluxInfos.Close();}
            catch { }
        }
    }
}

```

www.ista-ntic.net

47

Sérialisation XML (Exemple)

```

public class Employee
{
    public string EmpName;
    public string EmpID;
    public Employee(){ }
    public Employee(string newName, string newID){
        this.EmpName = newName;
        this.EmpID = newID;
    }
    public override string ToString()
    {
        return this.EmpID + " " + this.EmpName ;
    }
}

```

www.ista-ntic.net

48


```

public class Employees:ICollection
{
    public string CollectionName;
    private ArrayList empArray = new ArrayList();

    public Employee this[int index]
    {
        get { return (Employee)empArray[index]; }
    }
    public void CopyTo(Array a, int index)
    {
        empArray.CopyTo(a, index);
    }
    public int Count
    {
        get { return empArray.Count; }
    }
    public object SyncRoot
    {
        get { return this; }
    }
    public bool IsSynchronized
    {
        get { return false; }
    }
    public IEnumerator GetEnumerator()
    {
        return empArray.GetEnumerator();
    }
    public void Add(Employee newEmployee)
    {
        empArray.Add(newEmployee);
    }
}

```

www.ista-ntic.net 49

Exemple (Suite) : Main()

```

Employees Emps = new Employees();
Emps.CollectionName = "Employees";
Employee John100 = new Employee("John00", "100xxx");
Emps.Add(John100);
Employee John101 = new Employee("John01", "101xxx");
Emps.Add(John101);

```

```

XmlSerializer x = new XmlSerializer(typeof(Employees));
//Serialisation Emps
TextWriter writer = new StreamWriter("Coll.xml");
x.Serialize(writer, Emps);
writer.Close();

```

```

//Déserialisation Emps0
Employees Emps0 = new Employees();
XmlReader XmlRid = XmlReader.Create("Coll.xml", new
XmlReaderSettings());
Emps0 = (Employees)x.Deserialize(XmlRid);
XmlRid.Close();

```

```

// Affichage
foreach (Employee E in Emps0)
    Console.WriteLine(E.ToString());
Console.ReadKey();

```

```

<?xml version="1.0" encoding="utf-8"?>
<ArrayOfEmployee
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Employee>
    <EmpName>John00</EmpName>
    <EmpID>100xxx</EmpID>
  </Employee>
  <Employee>
    <EmpName>John01</EmpName>
    <EmpID>101xxx</EmpID>
  </Employee>
</ArrayOfEmployee>

```

www.ista-ntic.net

50