C# Synthèse

Classes et Objets

Vocabulaire

- Une classe
  - Permet au concepteur de définir un nouveau type d'objet
  - Elle est constituée schématiquement de:
    - Données membres -> état de l'objet
    - Fonctions membres -> traitement des données
  - Mot clé: class

```
class Etudiant
  {
     private float note;
     public string nomEtudiant;

    public SetNote(float  n)
    {
     note=n;
   }
  }
```

- Instance d'une classe
  - Appelé aussi objet (ou instance de la classe)
  - Est une réalisation particulière d'un type donnée
    - Ex: Un 'objet' Labrador est une instance de la classe d'objet Chien
  - Chaque objet possède son propre état indépendamment des autres
    - Possibilité cependant de 'partager ' une donnée grâce à *static*

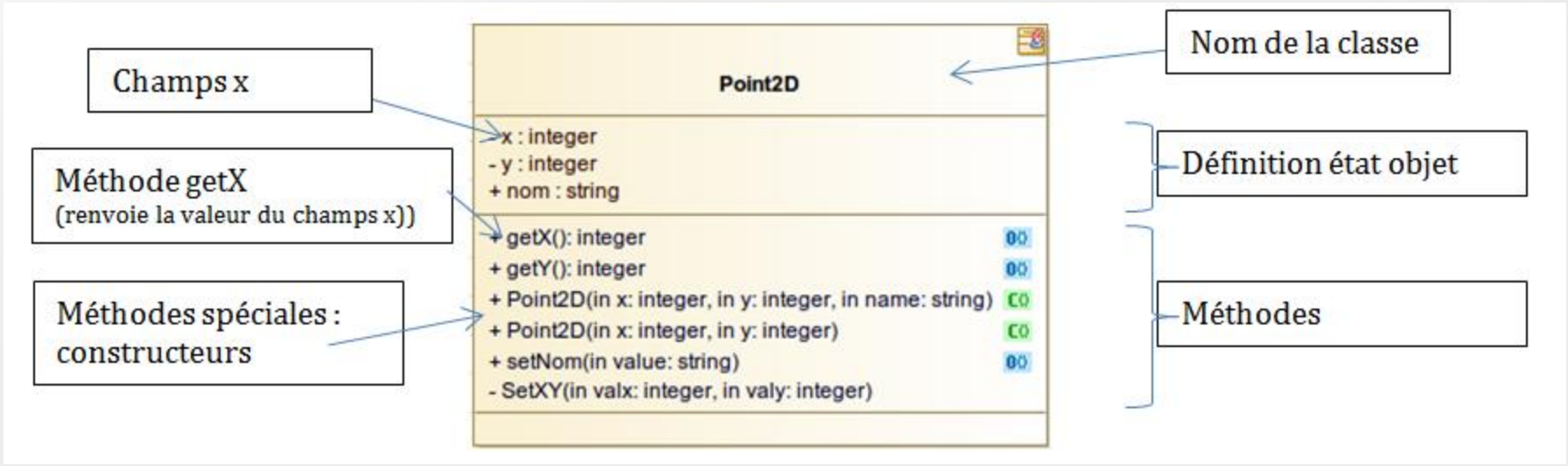
```
.....
etud1=new Etudiant();
etud2=new Etudiant();
 .....
```

- Les données membres
  - Appellées aussi champs de la classe

Représentation graphique

- Unified Modeling Language (UML)
  - Langage de modélisation

- Diagramme statique de classe



Encapsulation

- Concept important en POO
  - Regroupe au sein d'une même entité un ensemble cohérent d'éléments:
    - Données+ actions
  - Le mécanisme interne de la classe est caché vis à vis du monde externe
  - Les champs de la classe sont inaccessibles depuis l'extérieur
    - Notion de visibilité

```
public class Chat
    {
        uint _age; // par défaut Private!!
        public int _poids; // MAUVAISE PRATIQUE

        public String Crier()
        { return ("Miaouuu"); }
        public void SetAge(uint var)
        { _age = var; }
        public uint GetAge()
        {return (_age);}
    }
```

```
static void Main(string[] args)
        {
            Chat chat1, chat2;
            chat1=new Chat();
            chat2=new Chat();
            chat1.SetAge(2);
            //chat2._age = 10; KO COMPILE PAS
            chat2.SetAge(10);
            Console.WriteLine(chat1.Crier());
            chat1._poids = -10; // COMPILE MAIS ETAT
                                  //INCOHERENT
            Console.ReadKey();
        }
```

Les objets

- Ne pas confondre: objet et classe
  - Classe: définition d'un type
  - Objet: instance (réalisation, concrétisation , individu) de ce type

- Création d'instance
  - Opérateur new
    - Renvoie une référence vers l'objets crée

```
Chat chat1, chat2;
chat1=new Chat();
chat2=new Chat();

chat1=new Chat();
```

Création de 2 références vers des Chats

Création d'un Chat et affection de chat1

Création d'un autre Chat et affection de chat2

Création d'un 3èmeChat et affection de chat1

chat1 ne pointe plus vers le 1er Chat mais vers le 3ème.

Le ramasse miette se chargera de détruire le 1er chat qui n'est plus référencé

Méthodes

- Les différentes fonctions membres (non exhaustif)
  - Méthode d'instance
    - A la différence des fonctions classiques **l'appel de la méthode s'effectue au travers d'un objet** (instance d'une classe)
      - NomdeLobjet.NomdeLamethode
    - Exemple d'appel:
      - unMatou.EsperanceDeVie(poidsDuChat,ageActuel,race)

    - En tant que méthode d'instance il est possible d'accéder aux champs privés dans le corps de la méthode
      - Mauvaise pratique

    - Pour accéder aux champs privés des méthodes sont rajoutées:
      - Les GETTERS/SETTERS
      - Objectif: contrôler les données transmises afin d'assurer la cohérence de l'état de l'objet

Méthodes

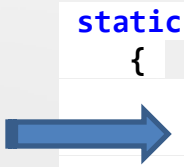
- ACCESSEURS : getter/setter
  - Point d'entrée obligatoire pour accéder aux champs de la classe
  - Objectif: assurer la cohérence de l'état de l'objet

```
public class Chat
{
    uint _age; // par défaut Private!!
    private const uint AgeMaximum = 100;

    public String Crier()
    { return (Miaouuu j'ai "+this.GetAge()+" ans"  )}

    // Accesseur
    public void SetAge(uint age)
    {
        if (age < AgeMaximum)
        {
            _age = age;
        }
        else
        {
            throw new ArgumentException("Age incohérent");
        }
    }
    public uint GetAge()
    { return (_age); }
}
```

```
static void Main(string[] args)
{
Chat unMatou = new Chat();
unMatou.SetAge(10);
 unMatou.Crier();
Console.WriteLine(
"le matou a " + unMatou.GetAge() + "
ans");
Console.ReadKey();
}
```

Méthodes

- ACCESSEURS : les Propriétés
  - Ne pas confondre les propriétés au sens général de la POO (désigne les champs de la classe) avec les propriétés C#
  - Ce sont des méthodes de types accesseurs
  - Vu de l'extérieur l'illusion est donné de manipuler un champ comme si il était publique
  - Intérêt: la facilité d'écriture



```
static void Main(string[] args)
    {
        Chat UnMatou = new Chat();
        UnMatou.AgeChat = 10; // exception non rattrapée
        Console.WriteLine("le matou a " + UnMatou.AgeChat + " ans");
        //try-catch
        try
        {
            UnMatou.AgeChat = 135; // exception rattrapée
        }
        catch (Exception ex)
        {
            Console.Error.WriteLine(ex.Message);
        }
        Console.ReadKey();
    }
```

Méthodes

- ACCESSEURS : les Propriétés AUTOMATIQUES

```
using System;

public class Chat
{

    private const uint AgeMaximum = 100;

    public String Crier()
    {
        return ("Miaouuu j'ai " + AgeChat + " ans"); // ou this.AgeChat
    }

    public uint AgeChat { get; set; }
}
```

La propriété automatique crée de manière masquée un champs privé
Exemple: private uint __xx139BackingField;

```
class Program
{
    static void Main(string[] args)
    {
        Chat UnMatou = new Chat();
        UnMatou.AgeChat = 10;
        ……
```

On a l'illusion de manipuler un champs publique cependant aucun contrôle sur l 'état du champs n'est possible!!!!!

Meilleure pratique:
Propriétés standard ou

```
public uint AgeChat { get; private set; }
```

+ un setter du type void ChangeAgeChat(uint age);

Méthodes de classe

- Les Constructeurs
  - Méthodes d'instances spéciales invoquées à la création de l'objet
  - Objectif: répondre à la question de l'initialisation de l'état de l'objet

```
public class Chat
    {
        private uint _age;
        private uint _poids;

        public Chat(uint age,uint poids)
        {
            _age = age;
            _poids = poids;
        }

        public override string ToString()
        {
            String affichage = String.Format("j'ai {0} ans et je pèse {1} kg", _age, _poids);
            return affichage;
        }
    }
```

```
static void Main(string[] args)
        {
            Chat chat1;
            chat1=new Chat(2,15);
            Console.WriteLine(chat1);
            Console.ReadKey();
        }
```

Méthodes de classe

- Méthode de classe
  - Précédée dans la définition de classe par l'attribut static
  - Elle s'appelle au travers de la classe elle-même
    - Nomdelaclasse. NomdeLamethode
  - Ne nécessite donc pas l'instantiation d'une classe
    - Ex: La classe Console (qui est aussi static) définit la méthode statique WriteLine *ex:Console.WriteLine("coucou")*

```
public class Imprimante
    {    // champs
         private static uint nombreImprimantes;
         private string imprimanteNom;

              …….
        // methode statique
               public static uint NombreTotalImprimanteGEII()
               {
                   return (nombreImprimantes);
               }
```

Syntaxe

C# | C++ | F# | VB

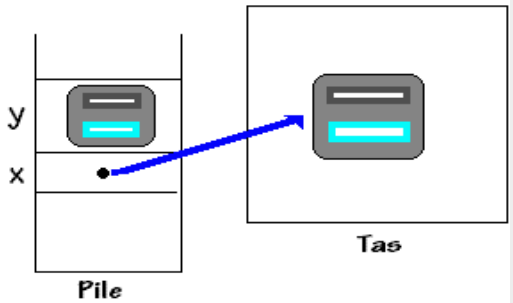
```
public static class Math
```

Méthodes

	Nom
	Abs(Decimal)

Référence et valeur: rappel

- Classification des variables
  - Variable de type valeur
    - La variable contient la valeur de la donnée
    - *Les types prédéfinis, les structures et les enum sont de type valeur*
      - Ex: int mavariable=100;
  - Variable de type référence
    - La variable de type référence stocke une référence aux données
      - principe d'indirection (analogie pointeurs)
    - Une référence établit "un lien" vers l'objet visé
    - une référence *n'est pas obligatoirement l'adresse de l'objet référencé*
      - *cependant les concepteurs du C# ont opté pour cette option technologique*
    - *Exceptés les cas plus hauts (variables), tout le reste est référence*



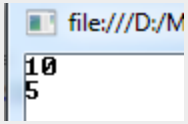
***y est de « type » variable**. Le contenu de la variable reflète le contenu de la variable y*

***x est de type référence**. Le contenu de la variable contient "un lien" vers la donnée situé ailleurs en mémoire.*

Passer des arguments

- Passage par valeur

```
static void Method( int varlocal,MonInt test2)
        {
        test2.vartest = 10;
        varlocal = 10;
        }

       public class MonInt
        {
            public int vartest=5;
        }
        static void Main(string[] args)
        {

            MonInt testint = new MonInt();
            int var = 5;
            Method(var,testint);
            Console.WriteLine(testint.vartest);
            Console.WriteLine(var);
            Console.ReadKey();
        }
```



Passer des arguments

- Passage par valeur d'une référence

```
class PassingRefByVal

    static void Change(int[] pArray)
    {
        pArray[0] = 888;  // This change affects the original element.
        pArray = new int[5] {-3, -1, -2, -3, -4};   // This change is local.
        System.Console.WriteLine("Inside the method, the first element is: {0}", pArray[0]);
    }
    static void Main()
    {
        int[] arr = {1, 4, 5};
        System.Console.WriteLine(
              "Inside Main, before calling the method, the first element is: {0}", arr [0]);
        Change(arr);
        System.Console.WriteLine(
                    "Inside Main, after calling the method, the first element is: {0}", arr [0]);
    }
}
```

Inside Main, before calling the method, the first element is: 1
Inside the method, the first element is: -3
Inside Main, after calling the method, the first element is: 888

Passer des arguments

- Méthodes: passage de paramètres par référence
  - Création variable locale de type référence
  - copie de la référence de l'objet transmis (référence sur la valeur référence)

```
class PassingRefByRef
{
    static void Change(ref int[] pArray)
    {
        // Both of the following changes will affect the original variables:
        pArray[0] = 888;
        pArray = new int[5] {-3, -1, -2, -3, -4};
        System.Console.WriteLine("Inside the method, the first element is: {0}", pArray[0]);
    }

    static void Main()
    {
        int[] arr = {1, 4, 5};
        System.Console.WriteLine(
              "Inside Main, before calling the method, the first element is: {0}", arr[0]);
        Change(ref arr);
        System.Console.WriteLine(
              "Inside Main, after calling the method, the first element is: {0}", arr[0]);
    }
}
```

```
Inside Main, before calling the method, the first element is: 1
Inside the method, the first element is: -3
Inside Main, after calling the method, the first element is: -3
```