informatique

ICT320

POO

PROGRAMMATION ORIENTEE OBJET

ANASS BENFARES &
XAVIER CARREL

ANASS BENFARES

Déroulement

- Accès à la Roadmap du cours : https://roadmap.sh/r/embed?id=66714f98c0f2325c34220bba

- 5 périodes de cours avec moi et des périodes de projet avec M.Melly.
- Théorie puis exercices pratiques.
- Informations concernant les évaluations à ce lien
- Ne pas oublier, les conventions de codage ETML !

Sujets

- Héritages
  - Concept
  - Classe de base / classe dérivée
  - Classe Abstraite
  - TypeOf
  - Mot clé « base »

Héritage - Concept

L'héritage est un principe fondamental de la programmation orientée objet.Il permet à une classe d'hériter des caractéristiques (attributs et méthodes) d'une autre classe.

Avantages :

- Réutilisation du code
- Polymorphisme

Classe de Base / Classe derivée

Une classe de base est celle dont les membres sont hérités.

Une classe dérivée hérite les membres de la classe de base et peut en définir de nouveaux.

Syntaxe en C# :

```
1   #Exemple1
    0 références
2   class ClasseDerivee : ClasseDeBase { }
3   # Exemple2
    1 référence
4   class Animal { }
    0 références
5   class Chien : Animal { }
```

Classe de Base ➔ Classe derivée

La classe dérivée peut utiliser les méthodes de la classe de base,

Mais l’inverse n’est pas possible !

La relation est **unidirectionnelle**

Base ➔ Dérivée

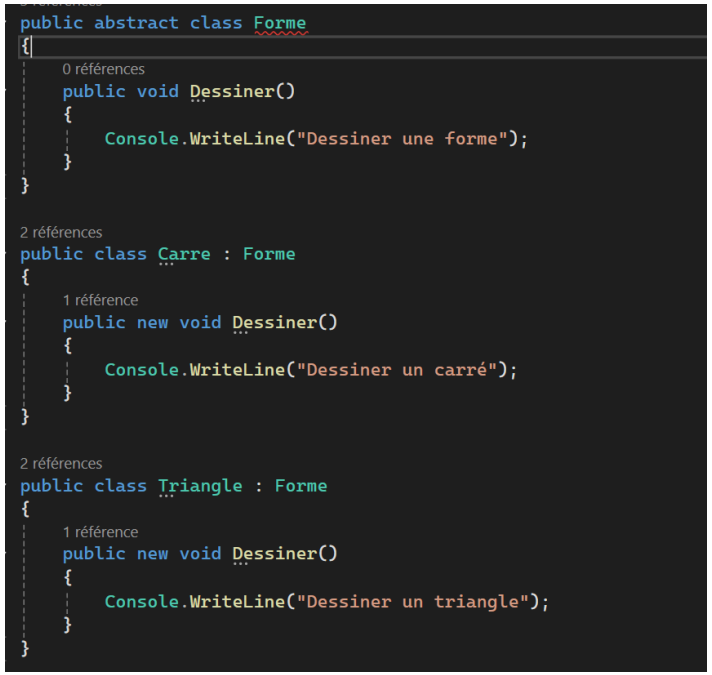
Visibilité

Visibilité	Classe	Classes dérivées	Extérieur
`public`	X	X	X
`protected`	X	X	
`private`	X		

Classe abstraite

Une classe abstraite est une classe qui ne peut pas être instanciée.

Elle peut contenir des méthodes abstraites (sans implémentation) qui doivent être définies dans les classes dérivées.



```
public abstract class Forme
{
    0 références
    public void Dessiner()
    {
        Console.WriteLine("Dessiner une forme");
    }
}

2 références
public class Carre : Forme
{
    1 référence
    public new void Dessiner()
    {
        Console.WriteLine("Dessiner un carré");
    }
}

2 références
public class Triangle : Forme
{
    1 référence
    public new void Dessiner()
    {
        Console.WriteLine("Dessiner un triangle");
    }
}
```

Classe abstraite

Utilisation :

```
Carre monCarre = new Carre();
Triangle monTriangle = new Triangle();

monCarre.Dessiner();     // Affiche : Dessiner un carré
monTriangle.Dessiner(); // Affiche : Dessiner un triangle
```

Classe abstraite

Utilisation :

```
Carre monCarre = new Carre();
Triangle monTriangle = new Triangle();

monCarre.Dessiner();     // Affiche : Dessiner un carré
monTriangle.Dessiner(); // Affiche : Dessiner un triangle
```

```
Forme maForme = new Forme();
```

TypeOf

```
4 références
class Vehicule
{
    0 références
    public string Marque { get; set; }
    0 références
    public void Demarrer()
    {
        Console.WriteLine("Le véhicule démarre.");
    }
}
```

Classe de base: **Véhicule**

```
class Voiture : Vehicule
{
    0 références
    public int NombreDePortes { get; set; }
}
2 références
class Moto : Vehicule
{
    0 références
    public bool AUnSidecar { get; set; }
}
```

- Classe dérivée : **Voiture**
- Classe dérivée : **Moto**

Main:

```
Vehicule v1 = new Voiture();
Vehicule v2 = new Moto();

if (v1.GetType() == typeof(Voiture)) {
    Console.WriteLine("C'est une voiture");
}

if (v2.GetType() == typeof(Moto))
{
    Console.WriteLine("C'est une moto");
}
```

Mot clé « Base »

Si on souhaite créer une **Moto**, et qu'on veut s'assurer que les méthodes ou propriétés ont été définis correctement dans la classe **Véhicule**.

Par exemple si on affiche un message dans le constructeur qui donne le type d'objet instancié.

Lorsque l'on va crée une moto nous aurons d'abord « Véhicule crée » puis « Moto crée ».

Exercices pratiques

- Drones