Programmation oriente object

Qu'est ce que la POO

La programmation orienté objet est une méthode de programmation qui permet de structurer un programme en `objet`.

Un objet est une représentation d'une entité, comme une voiture, un chat, un humain, un programme, etc...

Pourquoi POO

La POO permet de créer des objets qui ont des propriétés et des méthodes.
Cela permet :

- 📌 Reutilisation du code
- 📌 Efficacité de l'appel de fonction
- 📌 Facilité la maintenance

Pourquoi POO

- 📌 Facilité le dépannage
- 📌 Facilité la lecture du code
- 📌 Facilité la modification du code

Les classes

Les classes

Une classe est une structure de données qui permet de définir un nouvel objet.

Une classe est un modele de données.

Exemple : `Console`, `Dictionary`, `Math`

Declaration d'une classe

Une classe se déclare avec le mot clef `class` suivi du nom de la classe.

```
public class Personne { // Declaration d'une classe Personne
// ...
}
```

Classes privees et publiques

Nous pouvons déclarer les classes de deux manières différentes, en utilisant les mots clés a mettre devant `class` :

- `Public`
- `Private`

Classe Private

Une classe private ne peut être utilisée que dans la classe ou elle est déclarée. Nous parlons ici de l'encapsulation d'une classe.

Car oui, une classe peut être déclarée dans une classe.

```
private class Personne { // Classe private
// ...
}
```

Classe Public

Une classe public peut être utilisée dans toute le programme.

```
public class Personne { // Classe public
// ...
}
```

Attributs

Les attributs sont les variables d'une classe.
Elles peuvent elles aussi etre `public` ou `private`

```
class Personne {
public string nom;
public int age;
public string metier;
private float taille ;
}
```

Attributs private

De la même maniere, un attribut private ne peut etre utiliser que dans la classe ou il est déclaré.

Attributs public

Un attribut public peut être utilisée dans toute le programme.

Exemple :

```
Car peugot  = new Car();
Console.WriteLine(peugot.wheel); // affichera 4
Console.WriteLine(peugot.serial_number); // error --> attribut private


public class Car{
public int wheel = 4;
private int serial_number = 3201244;
}
```

Encapsulation

L'encapsulation c'est le faite de donner ou non accès aux attributs a l'utilisateur.

Lorsque vous creez une classe, tout les attributs n'ont pas vocations à être accessible par l'utilisateur.

Il est important d'organiser ses attributs priver/public pour protéger vos classes. Et garantir une meilleur experience a l'utilisateur.

Getter / setter

Les getters et setters permet de définir le retour et l'attribution d'un attribut.

`get` va retourner un attribut, et `set` va declarer/initialiser un attribut.

`get` et `set` sont des fonctions.

Utilisation get

Dans l'exemple précédent, une voiture aura toujours 4 roues, donc on pourrait definir le retour de `wheel` a 4.

```
Car peugot  = new Car();
Console.WriteLine(peugot.wheel);

public class Car{
public int wheel {
        get {return 4;}
}
}
```

Un getter peut aussi retourner une variable de classe (même private vu qu'il s'agit d'une fonction au sein de la classes)

```
Car peugot  = new Car();
Console.WriteLine(peugot.wheel);
Console.WriteLine(peugot.name);

public class Car{
private string car_name = "peugot";
public int wheel {
        get {return 4;}
}
public string name {
        get { return car_name;}
}
}
```

Utilisation set

Le setter permet d'attribuer une valeur à une variable de classe.

```
Car peugot  = new Car();
peugot.wheel = 15; // attribution ici
Console.WriteLine(peugot.wheel);
public class Car{
private string carname = "peugot";
private int _wheel;
public int wheel {
        get {return _wheel;}
        set {_wheel = value;} // possible grace au set
}
}
```

Methode

Une méthode est une fonction qui est déclarée dans une classe.

Elle peut être `public` ou `private`. (toujours pour rendre la methode utilisable en dehors de la classe ou non)

Methode

```
Personne steve = new Personne();
Console.WriteLine(steve.Presentation());

public class Personne {
public string nom = "Steve";
public int age = 32;
public string metier = "developper";
private float taille = 1.76F; // F suffixe pour float

public string Presentation() {
        return ("Je m'appel " + nom + ", j'ai " + age + " et mon metier est " + metier);
}
}
```

Methode avec des Parametres

Une méthode peut prendre des paramètres, tout comme une fonction.

```
Personne steve = new Personne();
Console.WriteLine(steve.Presentation("Steve", 32, "developper"));

public class Personne {
public string Presentation(string nom, int age, string metier) {
        return ("Je m'appel " + nom + ", j'ai " + age + " et mon metier est " + metier);
}
}
```

Overload

Une methode peut être redéfinie avec des argument différents.

C'est ce qu'on appel l'overload (surcharge en FR?) de méthode.

```
Personne steve = new Personne();
Console.WriteLine(steve.Presentation("Steve", 32, "developper"));
Console.WriteLine(steve.Presentation("Steve", 32));

public class Personne {
public string Presentation(string nom, int age, string metier) { // methode de base presentation
        return ("Je m'appel " + nom + ", j'ai " + age + " et mon metier est " + metier);
}

public string Presentation(string nom, int age) { // overload de la methode presentation avec 2 arguments au lieu de 3
        return ("Je m'appel " + nom + ", j'ai " + age);
}
}
```

Constructeur

Un constructeur est une méthode qui s'execute a l'instanciation d'une classe.

`new` permet d'instancier une classe.

Le constructeur permet d'instancier des variables de classe en même temps que la declaration de la variable de classe.

Le constructeur est une méthode qui porte le nom de la classe

Passant de

```
Personne steve = new Personne();
Console.WriteLine(steve.nom);
public class Personne {
public string nom = "Steve";
public int age = 32;
public string metier = "developper";
private float taille = 1.76F;
}
```

```
Personne steve = new Personne();
Console.WriteLine(steve.nom);

public class Personne {
public string nom;
public int age;
public string metier;
private float taille;

public Personne() { // constructeur
        nom = "Steve";
        age = 32;
        metier = "developper";
        taille = 1.76F;
}
}
```

Parametres du constructeur

Le constructeur peut prendre des paramètres, comme un fonction.

Toute les personnes ne s'appelle pas Steve et n'ont pas 32 ans.

Nous allons créer un constructeur qui va `set` notre personne, histoire de la rendre un peu plus unique

```
Personne steve = new Personne("Steve", 32, "developper", 1.73F);
Console.WriteLine(steve.nom);

public class Personne {
public string nom;
public int age;
public string metier;
private float taille;

public Personne(string c_nom, int c_age, string c_metier, float c_taille) { // constructeur
        nom = c_nom;
        age = c_age;
        metier = c_metier;
        taille = c_taille;
}
}
```

Parametres du constructeur

Maintenant vous pouvez créer une variable de type Personne en la nommant comme vous le souhaitez sans avoir à toucher au code dans la classe.

Overload de Constructeur

Un constructeur peut etre redéfini avec des argument différents.
Comme une fonction.

C'est ce qu'on appel l'overload de constructeur.

```
Personne steve = new Personne("Steve", 32, "developper", 1.73F);
Personne Lola = new Personne("Lola", 29);
Console.WriteLine(steve.nom);
Console.WriteLine(Lola.nom);

public class Personne {
public string nom;
public int age;
public string metier;
private float taille;

public Personne(string c_nom, int c_age, string c_metier, float c_taille) { // constructeur
        nom = c_nom;
        age = c_age;
        metier = c_metier;
        taille = c_taille;
}

public Personne(string c_nom, int c_age) { // overload du constructeur
        nom = c_nom;
        age = c_age;
}
}
```

Static

Attribut Static

Un attribut static est une variable qui appartient à la classe et non à l'objet.

Elle ne sera pas accessible via une variable; mais par le type.

```
Console.WriteLine(Personne.nbPersonne); // affichera 0
Personne Bob = new Personne("Bob");
Console.WriteLine(Personne.nbPersonne); // affichera 1
Personne Lea = new Personne("Lea");
Console.WriteLine(Personne.nbPersonne); // affichera 2
// Console.WriteLine(Bob.nbPersonne); // affichera error
// car la static n'est pas accessible via Bob


public class Personne {
public string nom;
public static int nbPersonne = 0;

public Personne(string c_nom) {
        nom = c_nom;
        nbPersonne++;
}

}
```