Programmation oriente Objet

Une classe

Une classe est un type de donnees qui permet de stocker des donnees.

Elle est declaree comme suit :

```
class NomClasse{
        // Champs
        // Methode
}
```

C'est un moule qui permet de creer des objets.

Instance de classe

Une instance de classe est un objet cree a partir d'une classe.

Elle est declaree comme suit :

```
NomClasse nomObjet = new NomClasse();
```

Il est possible de creer plusieurs instances d'une meme classe.

Il y a donc une difference fondamentale entre une classe et une instance de classe.

Encapsulation

Encapsulation

L'encapsulation est un concept qui permet de proteger les donnees d'une classe.

L'encapsulation nous permet de choisir quelles donnees sont accessibles depuis l'exterieur et lesquelles ne le sont pas.

Elle se base sur 3 niveaux d'accessibilite : `public`, `private`, `protected`.

Private

Les donnees `private` ne sont accessibles que depuis la classe elle meme.

Elles ne sont pas accessibles depuis l'exterieur.

Ainsi, il est impossible de modifier les donnees `private` ou utiliser les methodes `private` depuis l'exterieur.

Public

Les donnees `public` sont accessibles depuis l'exterieur.

Ainsi, il est possible de modifier les donnees `public` depuis l'exterieur.

Tout le monde peut modifier les donnees `public`, et utiliser les methodes `public`.

Protected

Les donnees `protected` sont accessibles depuis la classe elle meme et les classes qui en heritent.

Ainsi, il est possible de modifier les donnees `protected` depuis l'exterieur mais uniquement dans une classe derive.

Champs

Les champs sont des variables qui sont stockees dans une classe.

Elles sont declarees comme suit :

```
class NomClasse{
        private int nombre;
}
```

Accesseur

Les accesseurs sont des methodes qui permettent de modifier ou recuperer les donnees d'une classe.

```
private int nombre;
public int Nombre{
        get{
                return nombre;
        }
        set{
                nombre = value;
        }
}
```

Valeur par defaut

Les champs peuvent etre initialises avec une valeur par defaut.

```
private int nombre = 10;
```

Cela permet de ne pas avoir a initialiser les champs dans le constructeur.

Readonly

Il est possible de declarer un champ en `readonly`.

Cela permet de proteger le champ et de ne pas pouvoir le modifier.

```
private readonly int nombre = 10;
```

Il ne sera donc pas possible de modifier le champ nulle part dans le code.

Cela permet de proteger les donnees.

Methode

Methode specifique

Il existe des methodes specifiques qui sont appelees lors de la creation ou destruction d'un objet.

Elles sont declarees comme suit :

```
class NomClasse{
        public NomClasse(){
                // Constructeur
        }
        ~NomClasse(){
                // Destructeur
        }
}
```

Constructeur

Le constructeur est une methode qui est appelee lors de la creation d'un objet.

Il sert a initialiser les donnees de l'objet.

Les constructeurs peuvent etre overloader.

Destructeur / Finalizer

Le destructeur est une methode qui est appelee lors de la destruction d'un objet.

Il sert a liberer les ressources de l'objet.

Il est appele automatiquement par le garbage collector.

Il est possible de forcer la destruction d'un objet en utilisant le mot clef `using`.

Syntaxe d'une methode

Une methode est declaree comme suit :

```
portee modifier typeRetour  NomMethode(typeParametre1 nomParametre1){
        // Code
}
```

Exemple :

```
public void NomMethode(){
        // Code
}
```

Heritage

L'heritage permet de creer des classes derivees a partir d'une classe de base.

Cela permet de reutiliser le code d'une classe de base.

De plus, il est possible d'ajouter des methodes ou des donnees a une classe derivee.

Heritage simple

L'heritage simple permet de creer une classe derivee a partir d'une classe de base.

```
class NomClasseBase{
        // Code
}

class NomClasseDerivee : NomClasseBase{
        // Code
}
```

Polymorphisme

Polymorphisme

Le polymorphisme designe la possibilite d'utiliser une meme methode pour des objets differents qui ont des types differents.

Le polymorphisme est possible grace a l'heritage.

Le principe est que le type reconnu par le compilateur est le type de la classe de base.

Classe abstraite

Une classe abstraite est une classe qui ne peut pas etre instanciee.

Elle sert de base a d'autres classes.

Elle peut contenir des methodes abstraites.

Quand une classe herite d'une classe abstraite, elle doit implementer toutes les methodes abstraites.

Le but est de forcer les classes derivees a implementer certaines methodes.

Abstract

La methode `abstract` est une methode qui n'a pas de corps.

Elle sert a forcer les classes derivees a implementer cette methode.

Pour pouvoir utiliser le mot clef `abstract`, la classe doit etre `abstract`.

```
class abstract NomClasseBase{
        public abstract void NomMethode(); // Methode
        public abstract int Nombre{get;set;} // Propriete
}
```

Virtual

La methode `virtual` est une methode qui peut etre redefinie dans une classe derivee.

Elle laisse la possibilite de redefinir la methode dans une classe derivee sans la forcer a l'instar de `abstract`.

Exemple :

```
public virtual void NomMethode(){
        // Code
}
```

Override

La methode `override` est une methode qui permet de redefinir une methode d'une classe de base.

Elle sert a redefinir une methode d'une classe de base.

```
class NomClasseBase{
        public virtual void NomMethode(){
                // Code
        }
}
public override void NomMethode(){
        // Code
}
```

Interface

Une interface est une classe abstraite qui ne contient que des methodes abstraites.

Elle sert a definir un comportement.

Il est possible d'implementer plusieurs interfaces en meme temps.

Exemple :

```
interface NomInterface{
        void NomMethode();
}
```

Static

Le mot clef `static` permet de creer des methodes ou des champs qui ne sont pas lies a un objet.

Ils sont lies a la classe directement.

Cela permet de creer des methodes ou des champs qui ne sont pas lies a un objet, ils ne disparaitront pas lors de la destruction de l'objet.

Methode static

Le mot clef static sur une methode permet de creer une methode qui n'est pas liee a un objet.

Ainsi, il est possible d'appeler la methode sans creer d'objet.

```
class NomClasse{
        public static void NomMethode(){
                // Code
        }
}

NomClasse.NomMethode(); // Appel de la methode
```