Programmation orientée objet

5IOBJ-1

Jérémy Kairis

Cours 1

Programmation orientée objet

Jérémy Kairis

Présentation

Présentation

Jérémy Kairis

Diplômé IFOSUP Bac info en 2018

Co-fondateur de Key DevOps

.NET Mobile Software Designer

Contact : jeremy.kairis@ifosup.wavre.be

Github : https://github.com/jeremykairis

Présentation

Qui êtes-vous ?

Quel est votre parcours ?

Quels sont vos attentes ?

Objectif

Développer une application en POO

Pour atteindre le seuil de réussite, l'étudiant devra prouver qu'il est capable,

- de concevoir, d'installer et d'utiliser des objets appropriés à la solution ;
- de concevoir et mettre en œuvre une procédure de test partiel et intégré ;
- de justifier sa méthode de résolution ainsi que ses choix conceptuels et méthodologiques.

Acquis d’apprentissage

Maîtrise

Pour la détermination du degré de maîtrise, il sera tenu compte:

- le niveau de cohérence : la capacité à établir une majorité de liens logiques pour former un ensemble organisé,
- le niveau de précision : la clarté, la concision, la rigueur au niveau de la terminologie, des concepts et des techniques/principes/modèles,
- le niveau d’intégration : la capacité à s’approprier des notions, concepts, techniques et démarches en les intégrant dans son analyse, son argumentation, sa pratique ou la recherche de solutions,
- le niveau d’autonomie : la capacité à faire preuve d’initiatives démontrant une réflexion personnelle basée sur une exploitation des ressources et des idées en interdépendance avec son environnement.

Organisation

Cours

1. Programmation orientée objet
2. C# (console)
3. Blazor + MAUI (web + mobile)

* ASP.NET Core (minimal API), EF Core, LINQ…

Présences

Absences

Au minimum, 60 % de présence est nécessaire pour présenter l’examen.

Avec + de 40% d’absence, même justifiées, vous devrez recommencer le cours.

>= 9 absences = refus.

Pause de 15min à 10h et 11h30

Merci

Introduction

Introduction

Objectifs

- Comprendre les principes fondamentaux de la POO.
- Installer un environnement de développement (IDE) pour travailler avec C#.
- Apprendre les bases du langage C#.

Qu'est-ce que la POO ?

Définition

La Programmation Orientée Objet (POO) est un paradigme de programmation qui consiste à modéliser le monde réel à l'aide d’objets.

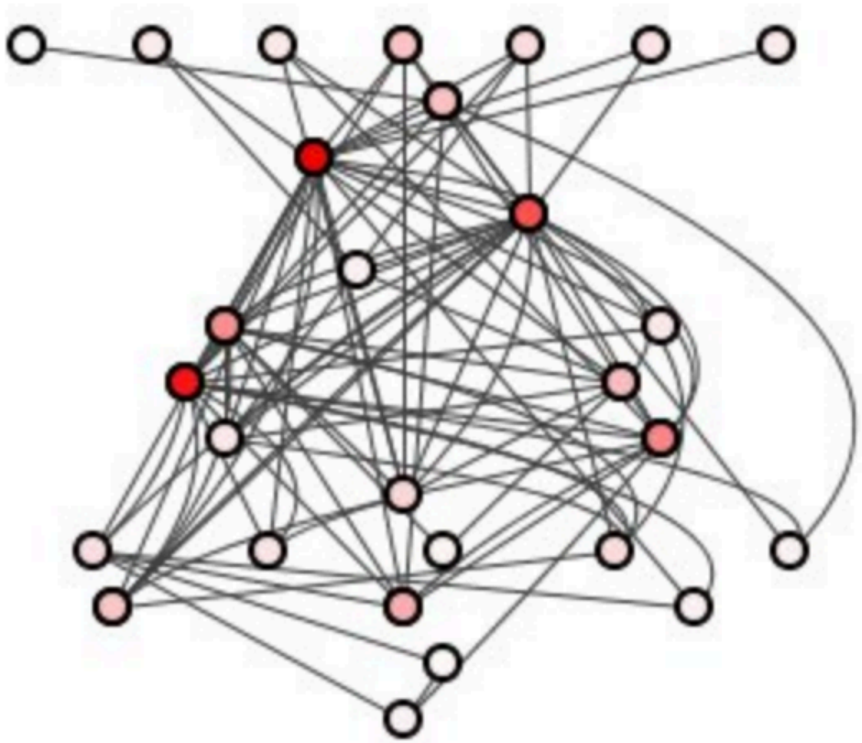
Les objets regroupent des données et des méthodes qui agissent sur ces données, permettant une organisation et une structure efficaces du code.

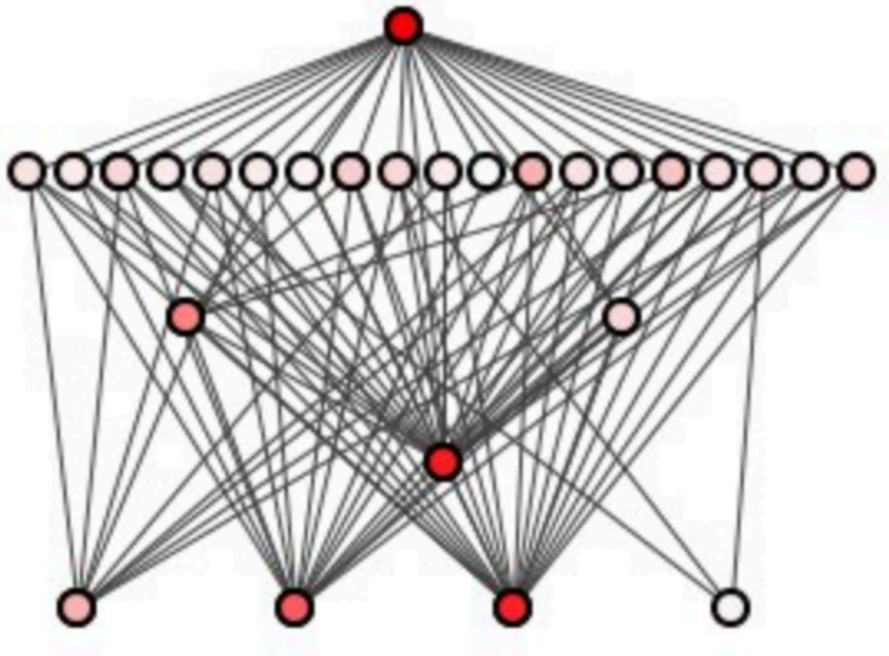
La POO favorise la réutilisation du code, l'encapsulation des données et la simplification de la gestion des programmes complexes.

L'importance de la POO

Pourquoi devriez-vous apprendre de la POO ?

Comprendre la POO est essentiel, car elle est au cœur de la manière dont nous concevons et développons des logiciels.



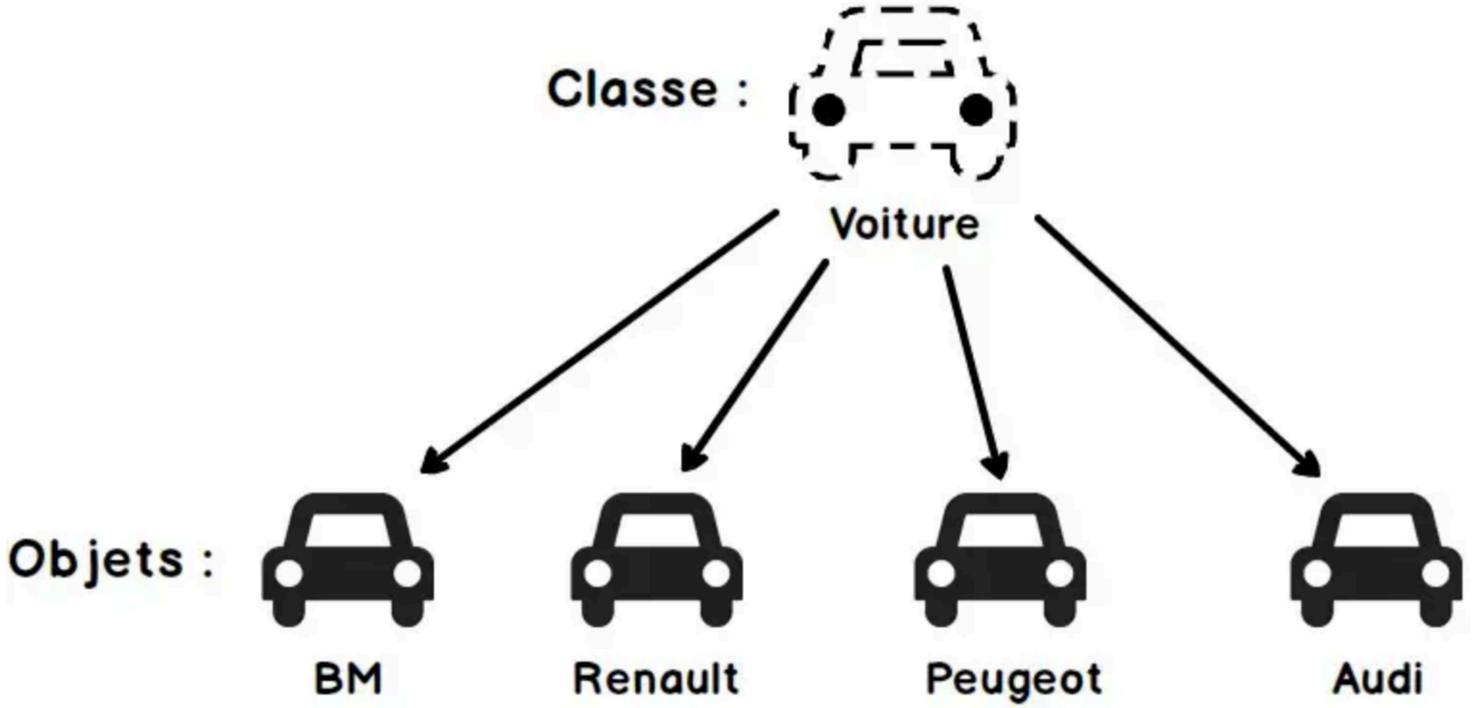


Les concepts de base

Objets et Classes

En POO, un objet est une instance spécifique d'une classe.

Une classe est un modèle qui définit les propriétés et les comportements d'un type d'objet.



Les concepts de base

Propriétés (attributs)

Les propriétés (ou attributs) d'un objet représentent ses caractéristiques.

Par exemple, une voiture peut avoir des propriétés telles que sa couleur et sa vitesse.

Les concepts de base

Méthodes (fonctions)

Les méthodes (ou fonctions) d'un objet décrivent ses actions.

Par exemple, une voiture peut avoir des méthodes pour démarrer et arrêter son moteur.

Les concepts de base

Exercice

1. Définissez des propriétés et des méthodes pour les classes suivantes :

- Personne
- Étudiant
- Animal
- Produit
- Compte bancaire
- …

Les concepts de base

Exercice

2. Créez des objets pour ces classes.

3. Appelez les méthodes des objets.

4. Quelle est la différence entre une classe et un objet ?

Avantages de la POO

Les 7 principaux

1. Structuration du code
2. Réutilisation du code
3. Encapsulation
4. Héritage
5. Polymorphisme
6. Gestion de la complexité
7. Maintenance facilitée

1. Structuration du code

Le plan

La POO apporte une structure à votre code, tout comme un plan pour une maison.

Elle permet de diviser le code en morceaux modulaires appelés "**objets**" et "**classes**".

Ces objets représentent des entités du monde réel (comme une voiture) et contiennent à la fois des données (comme la couleur d'une voiture) et des actions (comme démarrer ou arrêter la voiture).

Cette structuration facilite la gestion du code, sa lecture et sa maintenance.

2. Réutilisation du code

Mieux que le copié collé

Imaginez que vous ayez écrit un code complexe pour gérer un type de tâche, comme la gestion d'un compte utilisateur.

Vous pouvez réutiliser ce code dans d'autres parties de votre application ou même dans d'autres projets en créant une classe réutilisable.

Cela économise du temps et de l’effort.

3. Encapsulation

Le coffre-fort

L'encapsulation est le principe de cacher les détails internes d'un objet et de ne permettre l'accès qu'à travers des interfaces spécifiques.

Cela signifie que les données et les fonctions qui opèrent sur ces données sont regroupées dans une seule unité, appelée classe, et sont protégées contre un accès non autorisé depuis l'extérieur de cette classe.

En d'autres termes, l'encapsulation permet de contrôler qui peut voir et modifier les données d'un objet, ce qui contribue à prévenir les erreurs et à garantir la cohérence des données.

4. Héritage

La famille

Avec la POO, vous pouvez créer de nouvelles classes en héritant des fonctionnalités d'autres classes.

Par exemple, vous pouvez créer une classe "Véhicule" de base et ensuite des classes spécifiques comme "Voiture" et "Moto" qui héritent des caractéristiques de la classe de base.

Cela favorise la réutilisation du code et la gestion de la complexité.

5. Polymorphisme

L’interface

Il permet à des objets de différentes classes d'être traités de manière uniforme.

Cela signifie que vous pouvez utiliser une interface commune pour interagir avec des objets de classes différentes.

Le polymorphisme permet de créer un code plus générique et flexible, ce qui facilite l'extension du logiciel.

6. Gestion de la complexité

Petits objets

En POO, vous décomposez votre logiciel en objets plus petits et autonomes.

Chaque objet a une responsabilité claire et ne s'occupe que de ce qu'il sait faire le mieux, ce qui rend le code plus lisible et plus facile à gérer.

Les objets communiquent entre eux à travers des interfaces bien définies.

Cela signifie que les objets interagissent les uns avec les autres de manière prévisible et contrôlée.

Par exemple, dans un jeu vidéo, vous pouvez avoir un objet "Joueur" qui interagit avec un objet "Monstre" selon des règles spécifiques.

7. Maintenance facilitée

Fiabilité

En résumé, la POO simplifie la maintenance du code en facilitant la localisation des modifications, en isolant les changements, en favorisant la réutilisation du code, en permettant des extensions faciles, en réduisant les risques et en rendant la documentation plus compréhensible.

Ces avantages sont essentiels pour garantir que les applications logicielles restent évolutives et fiables à long terme.

Le language C#

Le language C#

Présentation

C# est un langage de programmation orienté objet et fortement typé.

Il propose directement dans sa syntaxe des outils qui facilitent la création et l'utilisation de composants logiciels.

Depuis sa création, C# a évolué en intégrant de nouvelles fonctionnalités pour répondre aux besoins modernes du développement et aux nouvelles pratiques de conception.

https://docs.microsoft.com/fr-be/dotnet/csharp/tour-of-csharp

Que développer avec ?

Build anything with a unified platform

Visual Studio Family
Visual Studio Code
CLI

TOOLS

+

Windows
Linux
macOS

OPERATING SYSTEM

+

NuGet
GitHub
Components, tools, library vendors

ECOSYSTEM

Installation

Visual Studio · Code & .NET

Visual Studio :

https://visualstudio.microsoft.com/fr/

Visual Studio Code :

https://code.visualstudio.com/docs/csharp/get-started

.NET SDK

https://dotnet.microsoft.com/en-us/download

Première application

Première application

Exercice

1. Créez votre première application console
2. Exécuter la
3. Débugguez la

Première application

La structure

bin

obj

Program.cs

Nom.csproj

La structure

.csproj

Le fichier ".csproj" est essentiel pour décrire la configuration d'un projet C#.

Il est lu par l'environnement de développement intégré (IDE) tel que Visual Studio ou dotnet CLI pour gérer la compilation, la construction et le déploiement du projet.

Chaque projet C# a son propre fichier ".csproj" pour définir ses caractéristiques spécifiques.

La structure

Program.cs

Le fichier “Program.cs” est le point d’entré de votre application C# où se trouvait anciennement la méthode "Main" qui démarre l'exécution de votre programme.

C'est là que vous écrivez le code principal de votre application pour définir son comportement.

La structure

bin & obj

Le dossier "bin" contient les fichiers binaires finaux exécutables ou de bibliothèque.

Le dossier "obj" contient des fichiers temporaires et intermédiaires utilisés pendant le processus de compilation.

Vous utiliserez principalement le dossier "bin" pour exécuter les applications ou utiliser les bibliothèques.

La structure

.sln

Le fichier ".sln" est un conteneur de solutions qui facilite la gestion et le développement de projets logiciels en regroupant plusieurs projets sous un même ensemble.

Il est spécifiquement conçu pour être utilisé dans des environnements de développement intégrés tels que Visual Studio.

Les variables

Les variables

Déclaration

Dans nos programmes, nous aurons souvent besoin de stocker temporairement différentes valeurs pour les utiliser dans diverses tâches.

Pour ce faire, nous utilisons des "variables" qui sont des réceptacles capables de contenir ces informations.

En C#, un langage très précis, nous devons déclarer le type de données (comme entier, booléen ou chaîne de caractères) pour nos variables au moment de leur création.

Il est important de noter que le C# fait également la distinction entre les majuscules et les minuscules, il faut donc être attentif lors de la déclaration des variables.

Les variables

Déclaration

La déclaration de variables respecte une nomenclature particulière.

Cette nomenclature autorise la déclaration de plusieurs variables du même type en les séparant par des virgules.

“***Type nom=value, nom2=value2, … ;***”

```
bool facile = true;

int a, b, c = 7;

var salut = "Bonjour !";
```

Les variables

Initialisation

L’initialisation d’une variable consiste à lui fournir sa première valeur, ensuite nous parlerons d’affectation de valeurs.

L’initialisation peut se faire à deux endroits :

- à la déclaration de la variable
- dans une méthode

Pour cela, l’opérateur “=” est utilisé.

```
int a, b, c = 7;


bool facile = true;
facile = false;


var salut = "Bonjour !";
salut = "Salut…";


a = 1;
```

Les variables

Membres et locales

Il y a 2 genres de variables :

- les variables membres qui sont définies au niveau de la classe
- les variables locales qui sont définies dans un bloc d'instructions

Lors de la déclaration d'une variable membre, celle-ci est initialisée automatiquement. Un nombre recevra automatique la valeur **0**, un booléen ***false***, etc. (***default(T)***)

```
int zero;

bool faux;

string txt;
```

Les variables

La portée

La portée d’une variable est la zone de code à partir de laquelle la variable est accessible et est définie par 2 règles :

- La variable membre est visible dans toute la classe qui la contient.
- Une variable locale est visible jusqu’à ce qu’une accolade fermante “***}***” indique la fin du bloc d’instructions ou de la méthode dans laquelle elle a été déclarée.

```
int membre = 2;

...

if (membre == 2) {
    var a = 1;
}
```

Les variables

Constantes

Le fait d'ajouter le mot "***const***" comme préfixe à la déclaration d'une variable désigne celle-ci comme constante.

Une constante est une variable dont la valeur ne peut-être modifiée pendant sa durée de vie.

Les constantes ont les caractéristiques suivantes :

- Elles doivent initialisées à la déclaration, et ne peuvent plus être modifiées
- La valeur d'une constante doit être calculable à la compilation.
- Les constantes sont implicitement statiques, on ne peut donc les préfixer du mot-clé "***static***".

```
const int MaConstante = 42;
```

Les variables

Les types *Nullable*

Dans le cadre des bases de données, les champs de tables peuvent être “***NULL***” indépendamment du type de donnée.

En .Net les types valeur primitifs ne peuvent pas avoir cette valeur “***null***”.

Pour résoudre ce problème, il suffit de les définir comme étant “***Nullable***”.

Pour cela, il y a deux possibilités :

- “***Nullable<T>***”
- *Suffix “**?**”*

```
int? valeurNullable = null;

Nullable<int> valeurNullable = null;
```

Les types de données prédéfinis

Les types de données prédéfinis

Les types valeur

Les types valeur sont des types de données dont les objets sont représentés par la valeur réelle de l'objet.

Si une instance d'un type valeur est assignée à une variable, cette variable reçoit une copie actualisée de la valeur.

Les types valeur font partie de la catégorie des “**Structures**” qui héritent de “***System.ValueType***”.

En mémoire, ceux-ci sont stockés sur la pile.

Exemples : une variable de type “***int***” est en fait une instance de type “***Int32***”,

“***long***” de “***Int64***”, “***float***” de “***Single***”, “***double***” de “***Double***”, “***bool***” de “***Boolean***”, etc.

Les types de données prédéfinis

Le caractère

Bien qu'un caractère puisse être précisé sous forme littérale en utilisant les simples guillemets, il peut aussi être spécifié en utilisant l'Unicode, l'hexadécimal, le code ASCII ou la séquence d'échappement.

Exemples : "**\0**", "**\n**", "**\r**", "**\t**", "\\", etc.

```
char caractere = ‘A';

char caractereUnicode = '\u0041';

char caractereAscii = (char)65;

char caractereEchappement = '\n';
```

Les types de données prédéfinis

Le valeurs possibles et par défaut

Type	Represents	Range	Default
bool	Boolean value	True or False	***False***
byte	8-bit unsigned integer	0 to 255	***0***
char	16-bit Unicode character	U +0000 to U +ffff	***\0'***
decimal	128-bit precise decimal values with 28-29 significant digits	(-7.9 x 1028 to 7.9 x 1028) / 100 to 28	***0.0M***
double	64-bit double-precision floating point type	(+/-)5.0 x 10-324 to (+/-)1.7 x 10308	***0.0D***
float	32-bit single-precision floating point type	-3.4 x 1038 to + 3.4 x 1038	***0.0F***
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	***0***
long	64-bit signed integer type	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	***0L***
sbyte	8-bit signed integer type	-128 to 127	***0***
short	16-bit signed integer type	-32,768 to 32,767	***0***
uint	32-bit unsigned integer type	0 to 4,294,967,295	***0***
ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615	***0***
ushort	16-bit unsigned integer type	0 to 65,535	***0***

Les types de données prédéfinis

Les valeurs littérales

Les valeurs littérales sont des valeurs primitives et peuvent être typées via un suffixe comme : ***0.0M***, ***0.0D***, ***0.0F***, ***0L***, etc.

```
decimal valeurDecimal = 0.0M;

double valeurDouble = 0.0D;

float valeurFloat = 0.0F;

long valeurLong = 0L;
```

Les types de données prédéfinis

Les types références

Les types référence sont des types de données dont les objets sont représentés par une référence (comme un pointeur) à la valeur réelle de l'objet.

Si un type référence est assigné à une variable, celle-ci référence (ou pointe vers) la valeur d’origine. Aucune copie n'est effectuée.

Il n’existe que 2 types prédéfini de type référence, ceux-ci font partie de la catégorie “**Classes**” et sont stockés sur le tas.

La classe “***Object***” est également appelée classe de base universelle.

Les types de données prédéfinis

La classe *String*

Bien que “***string***” soit de la catégorie “**Classes**” et donc de type référence, son fonctionnement interne est celui d’un type valeur.

En effet, lorsque nous nous affectons une valeur à une “***string***”, nous lui passons une copie de cette dernière.

Les types de données prédéfinis

La classe *String*

Il y a plusieurs possibilités pour concaténer des chaines de caractères :

- L'opérateur "**+**"
- La classe "***StringBuilder***"
- La méthode "***string.Format***"
- L'interpolation de chaine ***$"{...}***"

```
// Saisie d'une chaîne de caractères depuis la console
Console.Write("Entrez votre nom : ");
var nom = Console.ReadLine();
Console.Write("Entrez votre âge : ");
var age = Console.ReadLine();
// Affichage des données saisies
Console.WriteLine("Bonjour, " + nom + "! Vous avez " + age + " ans.");
Console.WriteLine($"Bonjour, je m'appelle {nom} et j'ai {age} ans.");
// Attendre que l'utilisateur appuie sur une touche pour fermer la console
Console.ReadKey();
```

Exercices

Variables & Types de données

1. Demandez à l'utilisateur son nom et son âge, stockez ces informations dans des variables appropriées et affichez-les.
2. Calculez et affichez la somme, la différence, le produit et le quotient de deux nombres que vous stockez dans des variables.
3. Demandez à l'utilisateur de saisir une valeur décimale, stockez-la dans une variable de type "double", puis affichez-la avec une précision de deux décimales.
4. Écrivez un programme qui convertit un nombre de jours en nombre d'heures, de minutes et de secondes, et affichez ces résultats.