# OOP - Introduction

Dans ce premier module, on découvre **pourquoi la programmation orientée objet (POO) existe** et les **concepts fondamentaux** qui la composent. L'objectif est de comprendre **le problème que la POO résout** avant d'introduire ses outils.

La **programmation orientée objet (POO)** est une manière de programmer qui consiste à **modéliser un programme sous forme d'objets**, où chaque objet regroupe **des données (attributs)** et **des comportements (méthodes)** liés à une même entité.
👉 Version encore plus directe :
La POO consiste à organiser le code en objets qui représentent des choses du monde réel et qui savent agir sur leurs propres données.

# Pourquoi la POO ?

## Le problème du procédural

Un programme procédural fonctionne souvent "à plat" :

```
string nom = "Aragorn";
int pointsDeVie = 100;
int attaque = 15;
int defense = 10;
bool estVivant = true;
```

Tout est :

- dispersé
- dupliqué si on a plusieurs personnages
- difficile à maintenir

## Si on ajoute un 2e personnage ?

```
string nom2 = "Legolas";
int pointsDeVie2 = 80;
int attaque2 = 18;
int defense2 = 8;
```

➡️ Problèmes :

- duplication de variables
- code illisible
- difficile à faire évoluer

## L'idée de la POO

La POO permet de :

- regrouper les données **logiquement**

- créer des modèles réutilisables
- manipuler des entités “réelles” (personnage, voiture, capteur…)

👉 On passe de :

“plein de variables isolées”

à :

“un objet cohérent qui représente quelque chose”

---

# Classe : le modèle

Une **classe** est un **plan**, un **modèle**, une **recette**.

Elle décrit :

- ce que possède un objet (ses données)
- ce qu’il peut faire (ses actions — plus tard)

## Exemple

```
class Personnage
{
    public string nom;
    public int pointsDeVie;
    public int attaque;
    public int defense;
    public bool estVivant;
}
```

👉 Cette classe ne crée rien encore 👉 Elle définit juste la structure

---

## À retenir

Une classe = → une description → un type personnalisé

---

# Objet : une version concrète

Un **objet** est une **instance réelle** d’une classe.

## Exemple

```
Personnage p1 = new Personnage();
```

👉 Ici :

- `Personnage` = le modèle
- `p1` = un objet réel

## Initialisation

```
p1.nom = "Aragorn";
p1.pointsDeVie = 100;
p1.attaque = 15;
p1.defense = 10;
p1.estVivant = true;
```

# Instance : même idée que “objet”

👉 En pratique :

- **objet** = terme général
- **instance** = terme technique

✔ Une instance = un objet créé à partir d’une classe

# Visualisation mentale (très important)

| Concept | Image mentale |
| --- | --- |
| Classe | Plan de maison |
| Objet | Maison construite |
| Instance | Une maison réelle basée sur le plan |

# Plusieurs objets à partir d’une classe

```
Personnage p1 = new Personnage();
Personnage p2 = new Personnage();
```

👉 Même structure 👉 Données différentes

```
p1.nom = "Aragorn";
p2.nom = "Legolas";
```

# Comparaison procédural vs POO

## Procédural

```
string nom1 = "Aragorn";
string nom2 = "Legolas";
```

## POO

```
Personnage p1 = new Personnage();
Personnage p2 = new Personnage();
```

👉 En POO :

- plus structuré
- plus lisible
- plus évolutif

---

# Pourquoi c’est essentiel en pratique

## Cas réel (ton contexte)

Dans ton projet :

- capteurs (BME688, GPS…)
- joueurs
- questions
- séries

👉 chacun devient une classe

Exemple :

```
class CapteurTemperature
{
    public float temperature;
}
```

👉 Chaque capteur devient un objet

---

# Notions clés à retenir

- La POO organise le code autour d’**objets**
- Une **classe** est un modèle
- Un **objet / instance** est une version concrète
- On peut créer plusieurs objets à partir d’une classe
- La POO évite la duplication et améliore la lisibilité

Vidéo à la une à regarder sur Youtube: ▶ http://youtu.be/gABYMZbfGok