SUP MTI
École Supérieure de Management, de Télécommunication et d'Informatique

RECONNUE PAR L'ÉTAT

PYTHON II

Programmation Orientée Objet

Enseignant: GHANEM

Introduction à la Programmation Orientée Objet.

Du procédural à l'objet.

a. Les limites de la prog. procédurale

- Au début, on codait une liste d'instructions que l'ordinateur exécutait de façon linéaire, comme une recette de cuisine : c'est la **programmation procédurale**. Elle est utilisée, par exemple, dans les langages Fortran, C, Pascal ou Perl.

- Puis on est passé à la **programmation orientée objet** où l'on crée et utilise des objets qui communiquent entre eux. Comme dans les langages Java, Python, C#, PHP, C++, Ruby, Swift ou l'Objective-C.

a. Les limites de la prog. procédurale

- Programmation procédurale : P.P. (Pascal, C, Fortran, etc.)

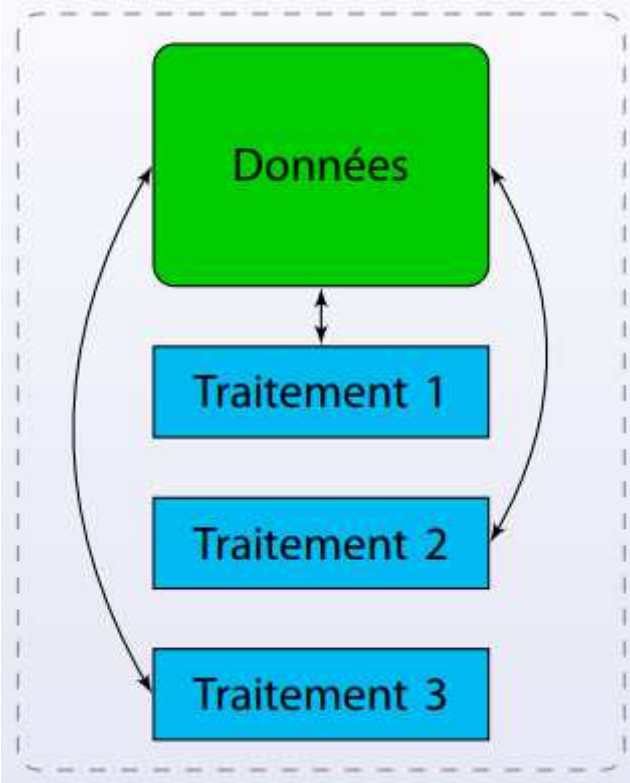
Contient simplement une série d'étapes à réaliser. N'importe quelle procédure (fonction) peut être appelée à n'importe quelle étape de l'exécution du programme.

- Programmation orientée objet : P.O.O. (Python, C++, Java, Delphi)

Concevoir une application comme un système d'objets interagissant entre eux

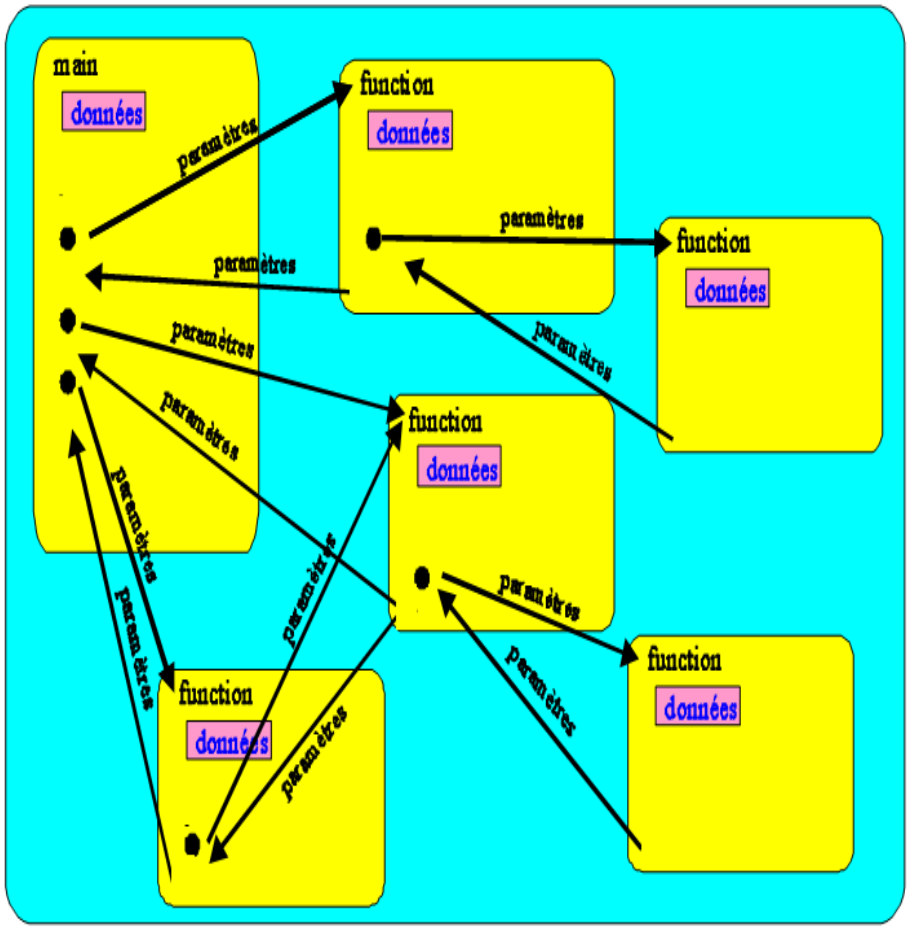
a. Les limites de la prog. procédurale

- Le programme est composé des fonctions
- Les données (variables) sont créées à l'intérieur des fonctions ou bien passées comme paramètres
- Les données et les traitement (fonctions ou modules) sont séparées



a. Les limites de la prog. procédurale

- Pour réaliser un logiciel avec le langage procédural , on commence par identifier les **fonctions principales** et les **structures de données** manipulées par ces fonctions. Ces fonctions principales pourront ensuite être « découpées » en fonctions auxiliaires afin de simplifier leur conception.
- Les modifications d'une fonction entraînent d'autres modifications dans autre fonctions.
- La portée d'une modification est trop grande et difficile à gérer.
- Redondance dans le code (la même chose est codé plusieurs fois)
- Propagation des erreurs– débogage difficile



Groupe Sup MTI

Programmation Orientée Objet (POO)

- La POO est une nouvelle façon d'organiser le code source, elle permet de le simplifier et de l'éclaircir; afin que d'autres personnes qui n'ont pas assistés à sa conception, puissant l'utiliser;

- Approche procédurale : « Que doit faire mon programme ? »

- Approche orientée-objet :« De quoi doit être composé mon programme ? »

- Cette composition est conséquence d'un choix de modélisation fait pendant la conception (comme en UML)

Programmation Orientée Objet (POO): Objet

La **programmation orientée objet** (POO) :

- Permet de créer des entités (objets) que l'on peut manipuler .
- Impose des structures solides et claires.
- Les objets peuvent interagir entre eux, cela facilite grandement la compréhension du code et sa maintenance.

Programmation Orientée Objet (POO): Objet

- Dans la **programmation orientée objet**, on regroupe au sein d'une **même entité** certaines **données** et les moyens de **traitement** de ces données.

Objet: entité identifiable du monde réel pouvant avoir ou pas une existence physique. *Exemples* : chien, table, personne, compteB…

- Un objet possède **trois composantes** :
  - + **identité** (adresse mémoire),
  - + **état** (attributs),
  - + **comportement** (Méthodes)

Personne1
Ahmad
7 ans
Parler()
Jouer()

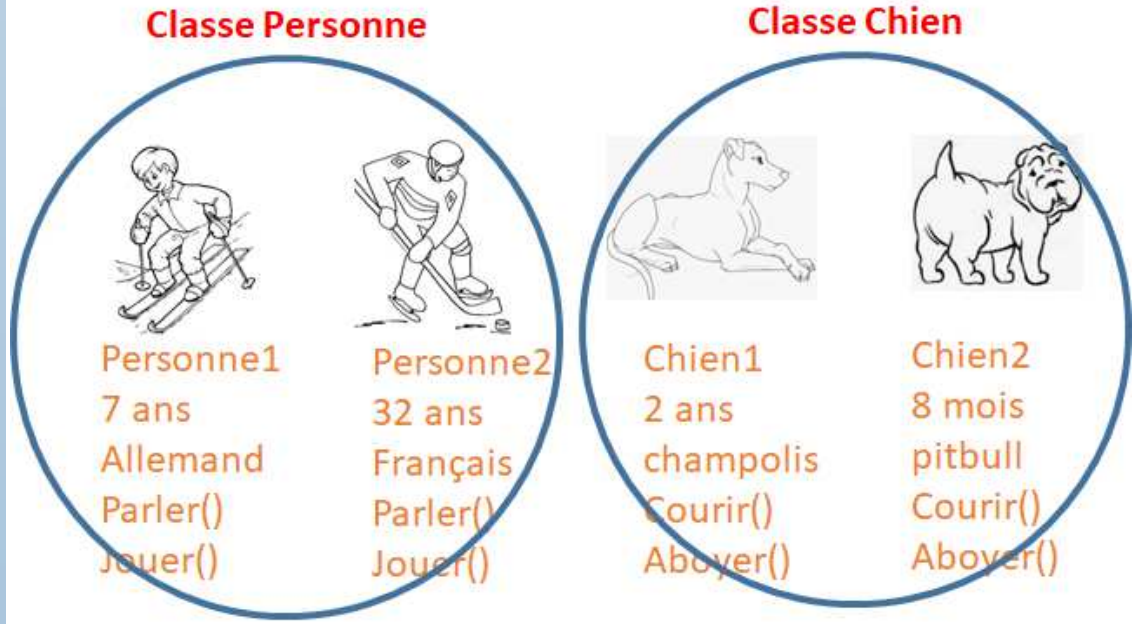
Personne2
Alex
26 ans
Parler()
Jouer()

Chien1
2 ans
champolis
Courir()
Aboyer()

Chien2
8 mois
pitbull
Courir()
Aboyer()

Programmation Orientée Objet (POO): Classe

- Des objets ayant des **propriétés communes** (attributs et méthodes) sont alors regroupés dans une structure abstraite appelée **classe**.
- Les valeurs sont différentes pour chaque objet




```
Class Chien {  …..      }

Class Personne{ ….     }
```

Programmation Orientée Objet (POO): Classe

- **Classe:**

Regroupement d'objets de même nature (mêmes **attributs** + mêmes **opérations**)

- **Objet**

A partir d'**une classe** on peut créer un ou plusieurs objets par **instanciation** ;

Chaque objet est une **instance** d'une seule classe.

Objet= instance d'une classe

Exercice : Regroupez ces mots en classes, attributs et méthodes:

- Voiture, Roue, Conducteur, Accélérer, Freiner, Race, Avion, Pilote, Altitude, Décoller, Atterrir, Carburant, Nom, Âge, Courir, Aboyer, Chien, Chat, Miauler, Vitesse, Ordinateur, Klaxonner, Processeur, Mémoire, Allumer, Éteindre, Livre, Titre, Auteur, Lire, Page, Emprunter, TailleÉcran, Taille, Marque, Redémarrer, Manger, Dormir, Sauter.

Groupe Sup MTI

indices :

- **Classes** : Identifiez 6 classes parmi les mots donnés.
- **Attributs** : Un attribut peut appartenir à deux classes différentes.
- **Méthodes** : Une méthode peut appartenir à deux classes différentes.

solution :

Groupe Sup MTI

Classe 1 : Voiture
Attributs : Roue, Conducteur, Vitesse, Carburant
Méthodes : Accélérer, Freiner, Klaxonner

Classe 2 : Avion
Attributs : Pilote, Altitude, Vitesse, Carburant
Méthodes : Décoller, Atterrir

Classe 3 : Chien
Attributs : Nom, Âge, Race, Taille
Méthodes : Aboyer, Courir, Sauter, Manger, Dormir

Classe 4 : Ordinateur
Attributs : Processeur, Mémoire, Marque, TailleÉcran
Méthodes : Allumer, Éteindre, Redémarrer

Classe 5 : Livre
Attributs : Titre, Auteur, Page
Méthodes : Lire, Emprunter

Classe 6 : Chat
Attributs : Nom, Âge
Méthodes : Miauler, Courir, Sauter, Manger, Dormir

- Dans l'exercice précédent, les classes que nous avons utilisées (comme **Voiture**, **Avion**, **chien**, **Ordinateur**, etc.) **n'étaient pas liées entre elles**. Elles appartenaient à des **domaines différents** (transport, animaux, technologie) et **n'avaient pas de relations directes**. Chaque classe existait de manière indépendante, sans interaction avec les autres.
- En revanche, dans l'exercice suivant (la modélisation d'un zoo), les classes sont **étroitement liées** et **interagissent entre elles**:

- Vous devez modéliser un zoo en utilisant les concepts de la programmation orientée objet. Le zoo contient différents types d'animaux, des employés, et des enclos.

Les classes sont :

- Lion
- Oiseau
- Enclos
- Employé (par exemple, un soigneur ou un vétérinaire)

Définir les attributs et méthodes pour chaque classe

- **Attributs** : Les caractéristiques de l'objet.
- **Méthodes** : Les actions que l'objet peut effectuer.

(en utilisant les mots ci-dessous).

nom, âge, poids, estMalade, manger(), dormir(), seSoigner(), rugir(), nom, taille, listeAnimaux, ajouterAnimal(), nettoyer(), nom, poste, nourrirAnimaux(), soignerAnimaux()

solution :

Classe Oiseau :

- Attributs : nom, âge, poids, estMalade
- Méthodes :

Classe Lion:

- Attributs : nom, âge, poids, estMalade
- Méthodes : rugir()

Classe Enclos :

- Attributs : nom, taille, listeAnimaux
- Méthodes : ajouterAnimal(), nettoyer()

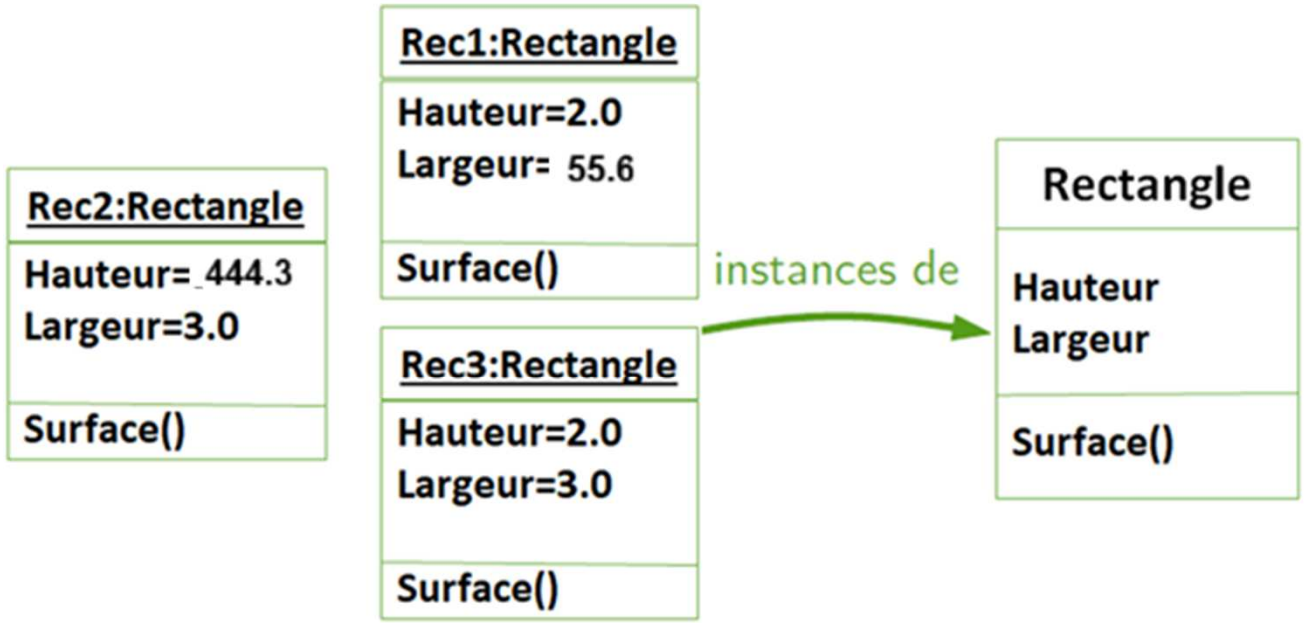
Classe Employé :

- Attributs : nom, poste
- Méthodes : nourrirAnimaux(), soignerAnimaux()

Programmation Orientée Objet (POO): Instance

Instanciation

- L'instanciation est l'opération qui consiste à créer un objet à partir d'une classe



Objet= instance d'une classe

Les concepts de la POO

La programmation orientée objet offre quatre concepts

- Encapsulation
- Abstraction
- Héritage
- Polymorphisme

Notion d'encapsulation

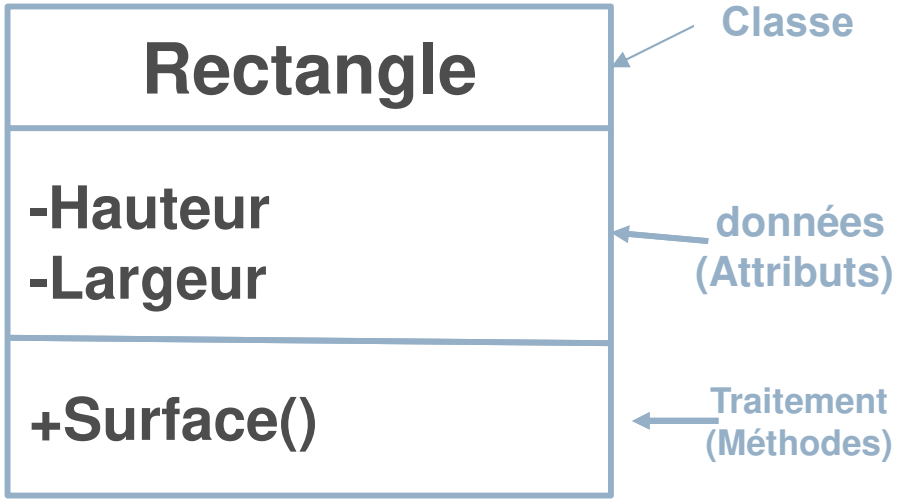
- Principe d'encapsulation

  - Regrouper des attributs et les méthodes au sein d'une même **classe**.

Exemple : la classe Rectangle encapsule les attributs hauteur, et largeur et la méthode surface().

  - Pour améliorer la lisibilité des programmes, les attributs encapsulés sont **souvent** privés (inaccessibles aux autres classes)

  - Les données et méthodes accessibles sont dites publiques



Notion d'encapsulation

Principe d'encapsulation

Principe interdisant l'accès direct aux attributs. On ne dialoguera avec l'objet qu'à travers une interface définissant les services accessibles aux l'utilisateurs de l'objet. Ce sera le rôle des méthodes.

Notion d'encapsulation

Visibilité des attributs et méthodes :

Un attribut ou une méthode sont dits **privés** si leur utilisation est interdite en dehors de la classe.

Un attribut ou une méthode sont dits **publics** si leur utilisation est autorisée en dehors de la classe.
Ce choix s'effectue lors de la déclaration de la classe.

La partie publique est appelée **interface** car tout le monde peut y accéder,

Notion d'encapsulation

Le principe d'encapsulation est donc de déclarer :

- Les attributs de façon privée.
- Les méthodes de façon publique.

Cependant, dans certains cas particuliers, on contreviendra à ce principe en imposant une visibilité différente de celle par défaut.

Notion d'encapsulation

Accès aux attributs

- Une question naturelle se pose alors : comment accéder aux attributs si ceux-ci sont déclarés de façon privée ?
- La réponse est : par l'intermédiaire de méthodes bien particulières, les « **getter** » et les «**setter** ».

Getter

- Il s'agit de méthodes publiques dont le rôle est de « **retourner** » la valeur d'un attribut.

Setter

- Il s'agit de méthodes publiques permettant de modifier le contenu d'une donnée membre protégée.

Notion d'encapsulation

Exemple : on ne peut accéder à l'attribut "**solde**" que via la méthode "**afficher solde**".

Compte
-nom : string -numéro : int -solde : double
+crediter(entrée x : double) +debiter(entrée x : double) +affichersolde() +affichernuméro() +affichernom() +changernom(entrée n : string)

Notion d'abstraction

- **Définition :**
  L'abstraction consiste à identifier les caractéristiques essentielles d'un objet en ignorant les détails non pertinents pour le contexte et à **exposer seulement les fonctionnalités essentielles** à l'utilisateur.
- **Exemple concret :**
- Quand vous utilisez une voiture, vous n'avez pas besoin de savoir comment fonctionne le moteur
- Vous utilisez simplement le volant, les pédales, le levier de vitesse

En POO :

- L'utilisateur sait **QUOI** utiliser mais pas **COMMENT** c'est implémenté
- Séparation entre **interface** (ce qu'on voit) et **implémentation** (ce qui est caché)

Notion d'abstraction

Principe d'abstraction

- Cacher la complexité et montrer seulement l'essentiel
- Définir une interface simple qui masque l'implémentation complexe
- L'utilisateur sait CE QUE fait l'objet, pas COMMENT il le fait

Exemple :

- Classe "Voiture" avec méthode démarrer()
- L'utilisateur appelle ma_voiture.démarrer()
- Il n'a pas besoin de connaître le système d'allumage électronique

Notion d'abstraction

Exemple du Thiéb Sénégalais
(ou couscous marocain)

Méthode principale que l'utilisateur appelle :

préparer_thiéb()

À l'intérieur, la cuisinière exécute discrètement :

vérifier_ingrédients_thiéb()
vérifier_poisson_frais()
contrôler_quantité_riz()

Notion d'abstraction

PUIS en fonction des vérifications :

```
si (poisson_manquant == vrai) :
    envoyer_au_marché_acheter_poisson()
sinon si (riz_insuffisant == vrai) :
    ajuster_quantité_riz()
sinon :
    préparer_sauce_poisson()
    cuire_riz_première_fois()
```

Ensuite, décision de cuisson :

```
vérifier_consistance_sauce()
contrôler_texture_riz()
```

Notion d'abstraction

PUIS :

```
si (sauce_trop_liquide == vrai) :
    réduire_sauce_feu_doux()
sinon si (riz_trop_cuit == vrai) :
    ajuster_temps_cuisson()
sinon :
    assembler_riz_et_sauce()
    faire_cuire_thiéb_final()
```

Méthodes techniques internes :

```
préparer_condiments()
faire_revenir_oignons()
ajouter_pâte_tomate()
mijoter_poisson()
contrôler_épices()
ajuster_piment()
vérifier_cuisson_parfaite()
```

Notion d'abstraction

Finalement :

laisser_reposer_thiéb()

dresser_plat_traditionnel()

ajouter_légumes_accompagnement()

Notion d'abstraction

Ce que voit l'invité :

Il appelle simplement : "Je veux du thiéb" 

Il ignore totalement les 15 étapes techniques de préparation

Il ne sait pas qu'on a vérifié le poisson, ajusté les épices, contrôlé la cuisson du riz

L'abstraction en action :

Interface simple : servir_thiéb_délicieux()

Complexité cachée : toutes les vérifications, ajustements et techniques culinaires

Notion d’abstraction

L'utilisateur profite du résultat sans se soucier du "comment c'est fait"

Analogie:
"Commander du thiéb au restaurant, c'est comme utiliser l'abstraction : vous dites juste 'Je veux du thiéb' et le chef gère toute la complexité derrière !"

Pause ⏸

Avant d'approfondir les autres caractéristiques de la POO…
…et puisque vous commencez maintenant à avoir une idée concrète de ce qu'est la programmation orientée objet…
…nous allons faire une pause dans l'apprentissage de la POO proprement dite.

Objectif :
Prendre du recul pour **comparer et contextualiser** les différents paradigmes de programmation.

Puis nous reviendrons à la POO avec une vision plus éclairée !

Groupe Sup MTI

Les Paradigmes de Programmation

1. **Programmation Impérative (Comment faire)**
   - Je donne des instructions étape par étape
2. **Programmation Déclarative (Quoi obtenir)**
   - Je décris le résultat sans dire comment
3. **Programmation Fonctionnelle (Transformations pures)**
   - J'enchaîne des fonctions sans effets secondaires
4. **Programmation Orientée Objet (Objets qui collaborent)**
   - Je crée des entités avec données et comportements
5. **Programmation Événementielle (Quand ça arrive)**
   - Je réagis à des événements
6. **Programmation Réactive (Flux de données)**
   - Je traite des streams de données en temps réel

Paradigme de Programmation	Instructions Caractéristiques
Programmation Impérative	Prendre_œufs Battre_œufs Mélanger_avec_farine
Programmation Déclarative	Gâteau = pâte_cuite(ingrédients_mélangés) Décoration = crème_fouettée + fruits Servir(gâteau_final)
Programmation Fonctionnelle	pâte = mélanger(œufs, farine) gâteau_cuit = cuire(pâte, 180, 30) gâteau_final = décorer(gâteau_cuit) **Donc** : gâteau_final = décorer(cuire(mélanger(œufs, farine) , 180, 30))
Programmation Orientée Objet	Four.cuire(gâteau) Gâteau.ajouter_décoration() Chef.servir(gâteau)
Programmation Événementielle	Quand(timer_fini) → sortir_du_four() Quand(bouton_préssé) → démarrer_mixeur() Quand(température_atteinte) → éteindre_four()
Programmation Réactive	flux_ingrédients → mélanger → cuire → décorer capteur_température → ajuster_chauffage commande_client → préparer → livrer

Programmation RÉACTIVE VS Programmation ÉVÉNEMENTIELLE

RÉACTIVE : Exemple : Un Thermomètre Intelligent

text

🌡️ Température_ambiante → [Si > 25°C] → 🌀 Allumer_ventilateur
→ [Si < 18°C] → 🔥 Allumer_chauffage

Fonctionnement :

Le thermomètre mesure **en continu** la température

Dès que la température change, le système **réagit automatiquement**

C'est un **flux permanent** de données → décisions

Programmation ÉVÉNEMENTIELLE

Exemple : Une Sonnette de Porte

text

🔔 QUAND(someone_appuie_sur_sonnette) → 🔊 Faire_sonner

Fonctionnement :

Rien ne se passe jusqu'à l'**événement précis** (appui sur sonnette)

Après l'action, le système **attend le prochain événement**

C'est **ponctuel** et **discret**

- Exercice :
- Donnez un exemple de classe et d'objet dans le contexte d'un système de gestion de bibliothèque.
- Solution :
- Classe : Livre
- Objet : livre1 (une instance de la classe Livre)

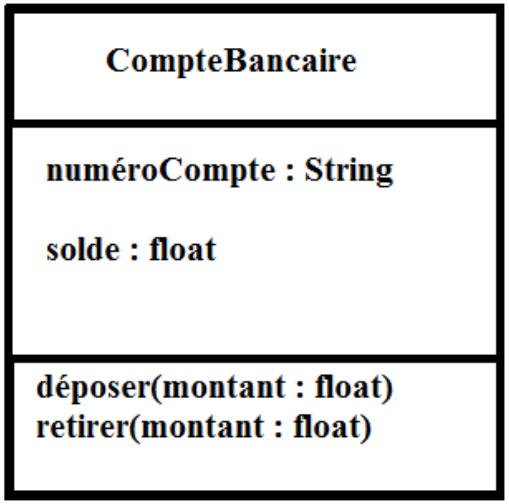
- Exercice :
- Identifiez les attributs et méthodes pour une classe Ordinateur.

- Solution :
- Attributs : marque, processeur, mémoire
- Méthodes : allumer(), éteindre(), redémarrer()

Groupe Sup MTI

Modélisation de classes simples

- Exercice :
- Modélisez (dessinez) une classe CompteBancaire avec les attributs numéroCompte, solde et les méthodes déposer() et retirer().


CompteBancaire
numéroCompte : String
solde : float
déposer(montant : float)
retirer(montant : float)

Exercice: Gestion des Notes Étudiantes

Implémenter une classe Note qui respecte les principes d'encapsulation et de validation des données.

Groupe Sup MTI

Consignes

Partie 1 : Conception de la Classe

- Dessinez le diagramme d'une classe Note avec :
- Un attribut **public** : nom_etudiant (chaîne de caractères)
- Un attribut **privé** : note (nombre réel)
- Deux méthodes **publiques** : setNote() et getNote()

Groupe Sup MTI

Consignes

Partie 2 : Algorithme du Setter

- Écrivez l'algorithme de la méthode setNote(nouvelle_note) qui :
- Vérifie que la note est comprise entre 0 et 20
- Stocke la note si elle est valide
- Affiche un message d'erreur si la note est invalide

Groupe Sup MTI

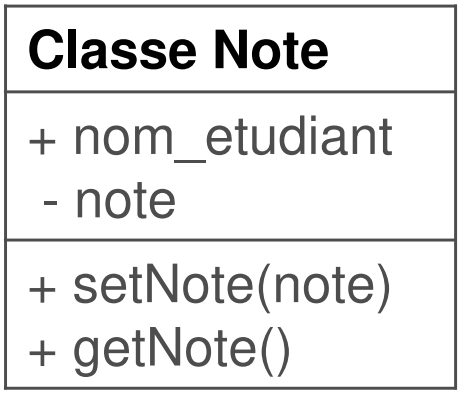
Consignes

Partie 3 : Algorithme du Getter

- Écrivez l'algorithme de la méthode getNote() qui :
- Retourne la valeur de l'attribut privé note

Groupe Sup MTI

solution



solution

```
Algorithme setNote(nouvelle_note)
Début
    Si (nouvelle_note >= 0 ET nouvelle_note <= 20) Alors
        note ← nouvelle_note
    Sinon
        Ecrire "Erreur : La note doit être entre 0 et 20"
    Fin Si
Fin
```

Groupe Sup MTI

solution

Algorithme getNote()

Début

    Retourner note

Fin

QCM

- Quelle est la différence entre programmation procédurale et POO ?
  a) La POO utilise des fonctions, la procédurale utilise des objets.
  b) La POO utilise des objets, la procédurale utilise des fonctions.
  c) Aucune différence.

- b

QCM

- Quel est un avantage de la POO ?
  a) Code moins organisé.
  b) Réutilisabilité.
  c) Difficulté de maintenance.

b

QCM

- Qu'est-ce qu'une classe en POO ?
  a) Une instance d'un objet.
  b) Un modèle pour créer des objets.
  c) Une fonction.

b

QCM

- Qu'est-ce qu'un attribut en POO ?
  a) Une fonction.
  b) Une variable qui décrit un objet.
  c) Une instance de classe.

- b

QCM

- Quel langage est souvent associé à la POO ?
  a) HTML.
  b) C++.
  c) C.

- b