POO et UML-2

Introduction
Premiers codes objets en Java et Python : 1

Diagramme de classes : 1
Diagrammes de classes techniques

Bertrand LIAUDET

USAGE du document :

Les mots / phrases importants sont en bleu
Les exercices sont en vert
L'algo généralisant le Python ou le Java est mis en gris
Le Python est mis en vert foncé
Le Java est mis en jaune
Les chapitres dont le nom se termine par « (pur technique) » ne concernent pas l'UML

SOMMAIRE

SOMMAIRE 1

0 – INTRODUCTION GENERALE 4

0.1 : Introduction s1 4
Programmation objet et langage objet 4
0.2 : Un type parmi les 6 grands types de programmation s2 5
1 : La programmation procédurale : le C 5
2 : La programmation WEB côté client : le HTML 5
3 : La programmation WEB côté serveur : le Java et le PHP s3 6
4 : La programmation événementielle et asynchone : le Javascript 6
5 : La programmation objet : le Java s4 7
6 : La programmation de base de données : le SQL 7
0.3 : Les types de programmation se mélangent s5 8
0.4 : Historique du « marché » des langages s6 9
0.5 : Programmation objet et méthodologie s7 10

1 - OBJET VS PROCEDURAL - CONCEPTION UML 11

1.1 - Programmation objet versus programmation procédurale s8 11
Notions du chapitre 11
Programmation procédurale : analyser-décomposer le « main » par fonctions s9 12
Programmation objet : analyser par objet s12 15
Paradigmes et MVC s13 16
1.2 - Conception UML et POO - Exemple d'objet : le feu de circulation s15 18
Notions du chapitre 18
UML . 18
Exemple : un feu de circulation s16 19

Les attributs d'un objet : comment caractériser cet objet « feu tricolore » s17 20
Les méthodes d'un objet : Que peut-il arriver à notre feu ? s18 21
Diagramme de classes UML : conventions graphique et de nommage s19 22
Types de méthode s20 23

2 - CLASSE, MOULE, CONSTRUCTEUR, ENCAPSULATION – EXEMPLES DE CODE. 24

2.1 : La notion de classe : un moule pour créer des objets 24
Notions du chapitre s21 24
Principes s22 25
Notion de constructeur s24 27
Représentation graphique UML s25 28
Code Java de la classe Gaufre (syntaxe simplifiée, sans encapsulation) s27 30
Code Python de la classe Gaufre 32
2.2 : Utilisation d'une classe : instanciation d'objets, « main » s28 33
Notions du chapitre 33
Principes s29 34
« main » s30 34
Instanciation : new s31 35
Accès aux méthodes : envoi de message 35
Affichage : toString() 35
2.3 : Exemple de « main » objet : codes s32 36
Exemple du moule à gaufre 36
2.4 : Principe d'encapsulation – attributs « private » et méthodes « public » s33 39
Notions du chapitre 39
Objet = état + comportement s34 40
Encapsulation : principes et syntaxe s35 41
Encapsulation : exemples – Voiture et FeuTricolore s36 42
L'art du programmeur : pas facile ! s37 44
Codage de l'encapsulation s38 45
2.5 : Catégorie de méthodes s39 48
Notions du chapitre 48
Catégories de méthodes : 1ère classification = standard s40 49
Catégories de méthodes : 2ème classification = distinction fonctionnel / technique 49
Méthode de conception s41 50
2.7 : *(pur technique)* Code : utilisation d'une classe avec un main séparé 51
Principe 51
Exemple en Java 51
Exercices - Code 51
2.8 : *(pur technique)* Exercices – 2 – Classe – Constructeur – Main 52
Principes 52

3 – EXERCICES - 1 : CLASSE ET MAIN 53

1 : Concevez et codez la classe Gaufre avec son main d'utilisation 53
2 : Feu tricolore 53
3 : Télévision 54
4 : Voiture 54
5 : Afficheur de score 55
6 : Personnage de jeu vidéo 55
7 : Distributeur de boissons 56

8 : Bissextile **57**

4 – BILAN **59**

Synthèse 59
La suite : les relations entre les classes et diagramme des classes métier 59

Edition février 2025

0 – INTRODUCTION GENERALE

USAGE du document :
Les mots / phrases importants sont en bleu
Les exercices sont en vert
L'algo généralisant le Python ou le Java est mis en gris
Le Python est mis en vert foncé
Le Java est mis en jaune
Les chapitres dont le nom se termine par « (pur technique) » ne concernent pas l'UML

0.1 : Introduction s1

Cette introduction introduit beaucoup de notions, donc du vocabulaire.

Programmation objet et langage objet

- On dit « programmation objet » ou « programmation orientée objet » ou « P.O.O. »
- La POO est un type de programmation parmi d'autres.
- La programmation objet s'est industrialisée dans les années 90 (langages C++ et Ada) et généralisée dans les années 2000.
- La programmation objet aujourd'hui, c'est partout :
- **D'abord** : **le Java, le C#, le C++.**
- **Mais aussi** : le **PHP**, le **JavaScript**, le **Python** et bien d'autres langages.

- Dans ce document, on verra des **exemples en Java et en Python**. Mais le but est de comprendre les principes.
- Les exemples **Java** correspondent à de la **programmation objet « standard ».** C'est proche du **C#**, du **C++** ou du **PHP**.
- Les exemples en **Python** correspondent à de la **programmation objet non typée**. C'est proche du **JavaScript**.
- Les exercices seront essentiellement des **exercices « papier »**, codables accessoirement en Java ou en Python.

0.2 : Un type parmi les 6 grands types de programmation s2

- On peut distinguer 5 types de programmation et 5 types de langages associés :

1 : La programmation procédurale : le C

- C'est la programmation qu'on peut appeler « de base ».
- Ce qu'on apprend en **algorithmique de base** : Variable, Type, Test, Boucle, Fonction, Tableau, Structure, Tableau de structures.
- Fonction = Procédure. « fonction » et « procédure » veulent dire quasiment la même chose.
- On dit aussi « **langage impératif** » à la place de « langage procédural ».
- **Principaux langages procéduraux :** . Fortran, Pascal, Cobol, C, Python, etc.

    - A noter que la **programmation fonctionnelle**, c'est autre chose. La programmation fonctionnelle utilise des **lambda-expressions**. On ne s'y intéresse pas ici.

2 : La programmation WEB côté client : le HTML

- C'est le langage pour fabriquer les pages WEB.
- On peut le mettre à part car ses principes et sa syntaxe sont très spéciaux.
- Un langage unique : . le HTML-CSS-Javascript .
- Ce langage est associé à des langages de templating (template) qui permettent d'introduire des tests, des boucles et des variables. Ces langages sont associés aux langages PHP, Python, Node.js, etc.

3 : La programmation WEB côté serveur : le Java et le PHP s3

- C'est le langage pour fabriquer les pages WEB dynamiques.
- Ce sont des langages impératifs qui permettent de construire des pages HTML.
- **Principaux langages WEB côté serveur** : . Java, PHP, Javascript, Python, C#, etc. .

4 : La programmation événementielle et asynchone : le Javascript

- C'est une programmation qui gère la prise en compte d'événements pour réagir en conséquence.
- Ces événements sont essentiellement des **événement** du type « **clavier** », « **souris** », « **écran tactile** » en provenance d'un utilisateur. Mais il peut y en avoir d'autres : de **événements système**, des **événements http** (réseau).
- Les événements sont captés par des « écouteur » : « listener » en anglais.
- L'écoute est « asynchrone » : la réponse peut venir n'importe quand après la demande.
- Principal langage événementiel : JavaScript, Node.js
- La plupart des langages procéduraux et objet fournissent des bibliothèques leur permettant de faire de la programmation événementielle (**Tkinter en Python**, **Swing en Java**, etc.)

5 : La programmation objet : le Java s4

- C'est une **programmation qui fabrique des « objets » informatiques** qui servent à simuler les objets de la réalité de la façon la plus intuitive possible.
- Une fois ces objets fabriqués, on va chercher les **fonctions qu'on peut leur appliquer**.
- Par exemple, j'ai un **objet voiture**. Je peux lui appliquer la **fonction démarrer()**, ou la **fonction changerVitesse( ).** Ces fonctions changent l'état de la voiture : le moteur tourne, la vitesse n'est plus la même.
- L'intérêt de la programmation objet est qu'on peut réutiliser les objets créés facilement dans d'autres programmes.
- Le Java est le langage de référence de la programmation objet.
- Principaux langages objets : le Java, le C++, le C# .

6 : La programmation de base de données : le SQL

- C'est une **programmation qui permet de consulter, traiter, ajouter, modifier, supprimer les données persistantes en base de données**.
- Principal langage de base de données : SQL
- D'autres langages existent autour de la BD : le NoSQL, le PL-SQL

0.3 : Les types de programmation se mélangent s5

- Aujourd'hui, les types de programmation se mélangent.
- La plupart des langages sont
  - des **langages objets**
  - qui permettent de faire **de la programmation événementielle et asynchrone**
  - et aussi de faire **de la programmation procédurale**
  - et qui intègre **des éléments de programmation fonctionnelle.**
- Le **Python** est un **langage plutôt procédural** qui permet de faire :

  - de la programmation procédurale,
  - de la programmation événementielle grâce à des bibliothèques (Tkinter),
  - de la programmation asynchrone grâce à des bibliothèques (Requests),
  - de la programmation objet
  - de la programmation fonctionnelle.

- Le **Java** est un langage **intrinsèquement objet** qui permet de faire la même chose.
- A noter toutefois que faire de la programmation procédurale en Python est une activité « normale ». On peut se limiter à un usage non-objet du Python.
- Faire de la programmation procédurale en Java n'est pas une activité normale ! Le Java est un langage intrinsèquement objet qui n'est pas fait pour faire du procédural.

0.4 : Historique du « marché » des langages s6

Vidéo du 7 octobre 2019

- https://www.youtube.com/watch?v=Og847HVwRSI

  ⇒ L'ère du Fortran : 1965-1979 (->56%)

  ⇒ L'ère du Pascal : 1980-1984 (->36%)

  ⇒ L'ère du C : 1985-2000 (-> 72% !)

  ⇒ L'ère du Java : 2001-2017 (->34%)

  ⇒ L'ère du JavaScript-Python-Java : depuis 2018 (->70% à eux 3)

Attention : ce n'est pas « la » vérité : c'est une vision des choses (cohérente avec mon expérience de « vieux » ayant vécu les évolutions industrielles et académiques des langages depuis les années 80).

0.5 : Programmation objet et méthodologie s7

<table>
<tr><th></th><th>Point de vue</th><th>Diagramme UML</th><th></th></tr>
<tr><td rowspan="4">ANALYSE FONCTIONNELLE</td><td rowspan="2">Statique – non objet (Analyse générale)</td><td>Cas d'utilisation</td><td rowspan="11">D O N N E E S</td></tr>
<tr><td>Séquence</td></tr>
<tr><td rowspan="2">Dynamique – non objet (Analyse détaillée)</td><td>Activités</td></tr>
<tr><td>Etats-transitions</td></tr>
<tr><td rowspan="2">ANALYSE DES DONNEES</td><td>Statique – non objet</td><td>MEA équivalent Classes</td></tr>
<tr><td>Statique – objet</td><td>Classes-métier</td></tr>
<tr><td rowspan="5">ANALYSE TECHNIQUE</td><td rowspan="2">Statique – objet</td><td>Classes techniques</td></tr>
<tr><td>Objets</td></tr>
<tr><td rowspan="2">Dynamique - objet</td><td>Séquence – objet
Séquence - MVC</td></tr>
<tr><td>Communication</td></tr>
<tr><td rowspan="2">Dynamique
Plutôt non objet</td><td>États-transitions</td></tr>
<tr><td>Activités</td></tr>
</table>

- La programmation objet se trouve dans les diagrammes de classes, d'objets, de séquence et de communication.
- Le diagramme de classes est un diagramme utile pour :
  - ⇒ L'analyse des données (qui relève du fonctionnel). On produit le Diagramme des Classes Métier (DCM).
  - ⇒ L'analyse technique. On produit des Diagrammes des Classes Techniques (DCT) selon les besoins.

1 - OBJET VS PROCEDURAL - CONCEPTION UML

1.1 - Programmation objet versus programmation procédurale s8

Notions du chapitre

- Programmation procédurale *versus* Programmation objet
- Objet, état d'un objet, méthodes d'un objet
- Paradigme procédural *versus* Paradigme objet
- Programme principal, main, contrôleur
- Framework
- MVC : Modèle, Vue, Contrôleur

Programmation procédurale : analyser-décomposer le « main » par fonctions s9

Principe

- Analyse le problème complet en le découpant en fonctions successives.

Exemple général

1) **Initialisation des données** : On commence par l'initialisation des données.

   ⇒ Une fonction fait ce travail.

2) **Saisie des données** : On continue par la saisie des informations par l'utilisateur.

   ⇒ Une fonction fait ce travail.

3) **Traitements des données** : On fait les traitements des données saisies.

   ⇒ Une fonction fait ce travail.

4) **Affichage des résultats** : On affiche les résultats des traitements.

   ⇒ Une fonction fait ce travail.

- Cet exemple, c'est déjà du MVC : 1 et 2 sont le M (le modèle, c'est-à-dire les données), 3 est le C (le contrôleur), 4 et le V (la vue).

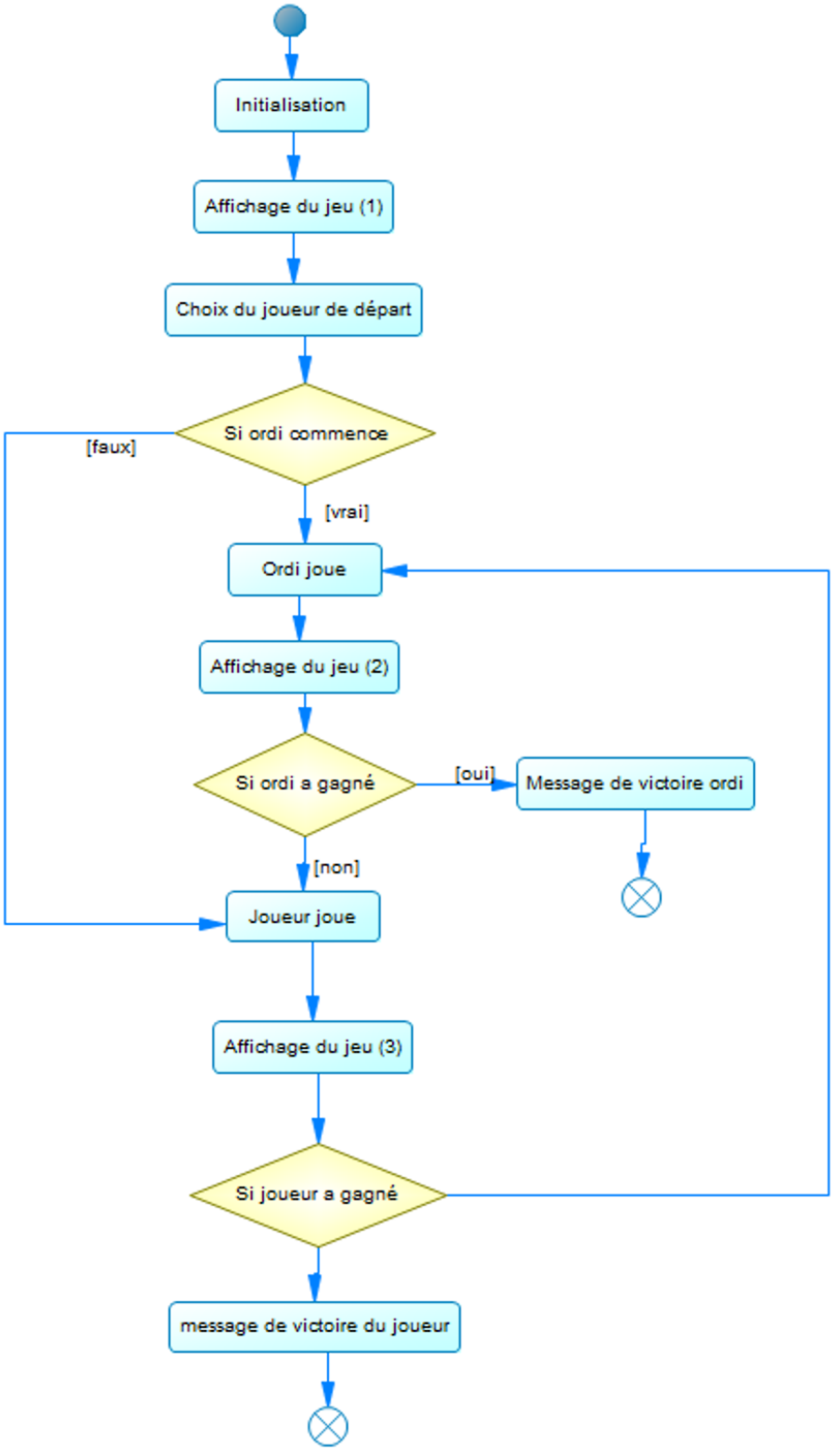
Exemple du jeu des allumettes (jeu de Marienbad), en algorithmique type Python **s10**

http://nezumi.dumousseau.free.fr/im10/marienbad.html

```
1:   plateau = initialisationDuJeu() ;
2 :  afficherLeJeu(plateau)
3 :  joueurDeDepart = choixDunJoueurDeDepart()
4 :  SI joueurDeDepart  == « ordi » :
5 :          plateau = iaJoue(plateau) ; afficherLeJeu(plateau)
6 :  BOUCLE répéter :
7:           plateau = joueurJoue(plateau) ; afficherLeJeu(plateau)
8 :          SI  jeuGagnant(plateau) :
9 :                  afficher(“le joueur a gagné”) ; quitter boucle
10 :         plateau  = iaJoue(plateau) ; afficherLeJeu(plateau)
11 :         SI jeuGagnant(plateau) :
12 :                 afficher(“l’ia a gagné”) ; quitter la boucle
```

On se retrouve avec au moins 6 fonctions :

- initialisationDuJeu()
- afficherLeJeu()
- choixDunJoueurDeDepart()
- joueurJoue()
- iaJoue()
- jeuGagnant()


Initialisation
Affichage du jeu (1)
Choix du joueur de départ
Si ordi commence
[faux]
[vrai]
Ordi joue
Affichage du jeu (2)
Si ordi a gagné
[oui]
Message de victoire ordi
[non]
Joueur joue
Affichage du jeu (3)
Si joueur a gagné
message de victoire du joueur

Programmation objet : analyser par objet s12

Principes

- **On analyse les « objets »** qui entrent en jeu dans le problème.
- Les **objets** doivent simuler des objets de la réalité. Ils sont caractérisés par leur état (le nom, l'age, la date de naissance, etc.)
- Ensuite, on définit les fonctions associées aux objets (= les méthodes).
- Enfin, **on écrit le programme qui utilise ces objets** pour produire les résultats attendus (= le main = le ou les contrôleurs)

Exemples

- Un objet utilisateur et des méthodes associées :
  - changer le mot de passe d'un utilisateur.
  - afficher les informations d'un utilisateur.

- Un objet voiture et des méthodes associées :
  - démarrer une voiture.
  - avancer()
  - changer la vitesse d'une voiture.

- Un objet agenda et des méthodes associées :
  - ajouter une personne dans un agenda.
  - rechercher une personne dans un agenda.

- etc.

Paradigmes

- On parle de « paradigme objet » *versus* « paradigme procédural ».
- Un paradigme, c'est une **façon de voir les choses, de réfléchir, de faire**, qui est complètement différente.

Paradigme objet

- La façon de faire « objet » est complètement différente de la façon de faire « procédurale ».
- On ne s'intéresse pas d'abord au programme principal, au but ultime qu'on découpe en fonctions.
- **On s'intéresse d'abord aux objets** qui vont participer à la réalisation de ce but et aux fonctions (méthodes) qu'on peut attacher aux objets pour réaliser ce but => ce sera le diagramme de classes métier (DCM).

  ⇒ Autrement dit, avec le paradigme objet, on ne s'intéresse pas d'abord à la totalité (le main), mais à ses parties (ses composants) : les objets.

Le « main » = contrôleur principal s14

- Dans les deux cas de paradigme, objet ou procédural, il faudra un « main ».
- Le « main » relève de la programmation procédurale : il va organiser les traitements.
- Le « main » est un « contrôleur ». C'est le « contrôleur » principal (le programme principal) qui va dispatcher vers des contrôleurs secondaires (des fonctions par exemple).

Paradigme « pro » : les framework – architecture MVC

- Un framework c'est une architecture de dossiers et des fichiers qui sert de modèle (de «patron » dans le sens d'un patron de couture) pour la suite.
- En général, cette architecture organise le code avec :

  ⇒ des **classes métiers** qu'on va retrouver dans la BD. L'organisation de ces classes s'appelle le « **modèle** ».

  ⇒ des **contrôleurs** dont le contrôleur principal qu'est le « main ».

  ⇒ des **affichages** qu'on appelle des « **vues** ».
- L'ensemble forme une architecture **MVC** : Modèle, Vue, Contrôleur.

Paradigme du débutant : tout dans le main !

- A noter que le programmeur débutant n'applique ni le paradigme procédural, ni le paradigme objet, mais le « **paradigme du débutant** » (ou paradigme du main) : il met tout son code dans le « main » sans fonction, ou presque !

1.2 - Conception UML et POO - Exemple d'objet : le feu de circulation s15

Notions du chapitre

- UML
- propriété (= attribut) d'un objet
- getter, setter
- responsabilité
- classe
- diagramme de classe
- camelCase
- snake_case

UML .

- **L'UML, en POO** (programmation orientée objet), c'est **une façon de dessiner son code** !

  ⇒ Ça sert à réfléchir ! C'est utile pour la conception.

  ⇒ Attention : c'est du dessin technique ! On ne fait pas ce qu'on veut !

Exemple : un feu de circulation **s16**

Un **feu de circulation** (ou feu tricolore ou feu rouge) est un dispositif d'éclairage : la boite avec ses différentes ampoules. On veut simuler son fonctionnement.

- On considère que le feu marche tout le temps.

  ⇒ Il est forcément allumé et dans un certain état : vert, orange ou rouge.

- On simplifie donc les choses : il n'y a pas de orange clignotant et pas de feu éteint. Le feu passe forcément du vert au orange, du orange au rouge et du rouge au vert.

Les attributs d'un objet : comment caractériser cet objet « feu tricolore » s17

- Une propriété, c'est une information qui caractérise un objet.
- Une propriété peut changer de valeur ou pas au fur et à mesure de la vie de l'objet.
  - Par exemple :

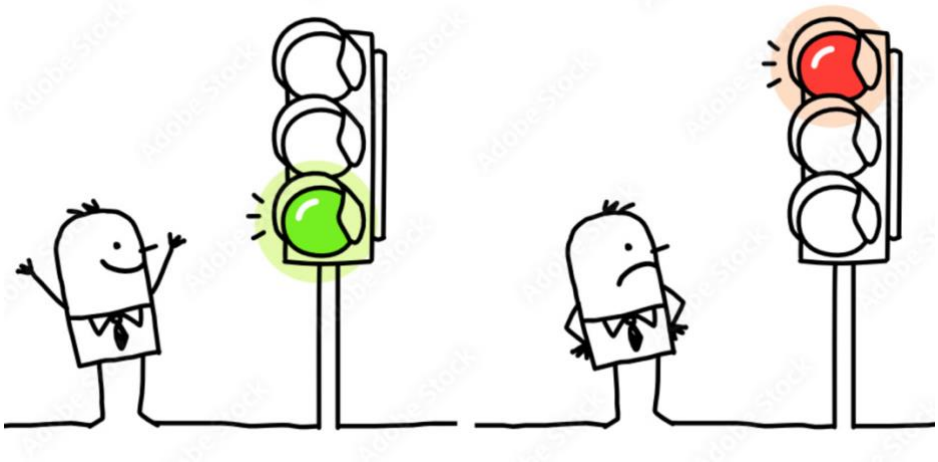
  ⇒ la **nom d'un joueur** (dans un jeu vidéo) est une propriété qui ne change pas (on peut le décider comme ça). Elle est définie à sa création.

  ⇒ Par contre la **force d'un joueur** peut changer.

- Ici, on en a **une seule propriété : la couleur du feu**.
- La couleur peut être : vert, orange ou rouge.
- En programmation objet, les **propriétés** des objets sont appelées « **attribut** ».
- « **couleur** » est un **attribut** de l'objet feu de circulation.

Les méthodes d'un objet : Que peut-il arriver à notre feu ? s18

Se demander ce qu'il peut arriver à notre feu, c'est aussi **se demander ce qu'il peut arriver à ses attributs.**



- La couleur peut changer de valeur.
- Ce changement est fonction de la couleur précédente : du vert on passe au orange, du orange au rouge et du rouge au vert.

  ⇒ On va définir une fonction **changerCouleur()** qui permet de gérer ce changement. Cette fonction n'a pas de paramètre (on ne passe pas d'informations entre les parenthèses).

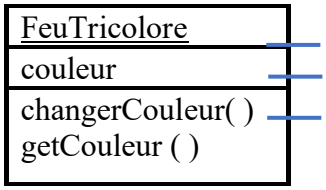
  ⇒ Les fonctions s'appliquent forcément à un objet : quand on applique **changerCouleur**() à un objet feu, la fonction change la couleur de l'objet, la couleur actuelle de l'objet.
- On veut aussi pouvoir **accéder à l'état du feu même sans le voir** ! Pour ça on définit une fonction **getCouleur()** : elle nous renvoie la couleur du feu.
- En programmation objet, les fonctions associées aux objets sont appelés « **méthode**».

  ⇒ **changerCouleur**() et **getCouleur**() sont deux méthodes de l'objet feu de circulation.

Diagramme de classes UML : conventions graphique et de nommage **s19**

Diagramme de classes

- On représente la synthèse de notre analyse graphiquement dans un rectangle à 3 compartiments :



un compartiment pour le nom de général de l'objet (sa **classe**)
un compartiment pour les **attributs**.
un compartiment pour les **méthodes**.

- Le nom général (ou le type) pour tous les objets qui seront des feux tricolores c'est ce qu'on appelle sa « **classe** ».
- Le schéma est standard : c'est un **diagramme de classes** UML.
- C'est très pratique : ça permet de synthétiser efficacement notre analyse.
- On l'utilise pour les diagrammes de classes techniques, DCT.

Convention de nommage

- La **première lettre** d'un nom de classe est toujours une **majuscule**.
- La première lettre du nom des attributs et des méthodes est en minuscule.
- On utilise ensuite le « **camel case** » : chaque mot suivant du nom complet commence par une majuscule. Le « **snake case** » (chaque partie est séparé par un _) est utilisé en Python.

- Cette convention de nommage est standard : on la retrouve dans tous les langages objets.
- C'est très pratique : avec l'habitude, ça permet de mieux s'y retrouver dans le code.

3 premiers types de méthode : getter, setter et responsabilité :

- Les fonctions pour récupérer un état sont des « **getter** » : on les appelle getCeci() ou getCela().
- Les fonctions pour donner directement une valeur à un attribut sont des « **setter** » : on les appelle setCeci(nouvelleValeur) ou setCela(nouvelleValeur).
- La fonction changerCouleur() n'est ni un getter, ni un setter. C'est une « **responsabilité** » au sens de : ce que l'objet doit pouvoir faire. Toutefois, c'est aussi un setter de couleur.

Attention :

- On aurait pu définir une fonction **setCouleur(nouvelleCouleur).**
  - ⇒ Cette fonction permet de donner une nouvelleCouleur à la place de la couleur actuelle. On passe la nouvelle couleur en paramètre. On dit « setCouleur » qui veut dire « donner une valeur à la l'attribut « couleur ». Cette valeur c'est le paramètre « nouvelleCouleur ».
  - ⇒ **On ne va pas faire ça !!!** Pourquoi ?
  - ⇒ Si on définit cette fonction, alors on pourra donner n'importe quelle couleur à n'importe quel moment, donc passer du vert au rouge directement : dans l'idéal, on veut éviter ça !

2 - CLASSE, MOULE, CONSTRUCTEUR, ENCAPSULATION – EXEMPLES DE CODE.

2.1 : La notion de classe : un moule pour créer des objets

Notions du chapitre **s21**

- Classe : un moule pour créer ses objets
- Type d'un objet
- Abstrait et concret
- Constructeur
- Instanciation
- Getter, setter, responsabilité
- this

Principes	**s22**

- La **programmation objet** consiste à **manipuler des objets** qui contiennent des attributs et des méthodes.

- **La classe est un « type » :**

  ⇒ Un objet est d' **une certaine sorte** : un objet Feu de circulation, Télévision, Voiture, etc.

  ⇒ La « sorte » de l'objet, c'est ce qu'on appelle le « **type** » en programmation procédurale et la « **classe** » en programmation objet.

- La classe est aussi un « moule » :

  ⇒ la classe, c'est ce qui permet de décrire ce que seront les objets concrets qu'on fabriquera à partir de cette classe.

  ⇒ C'est un peu comme **un moule pour fabriquer des objets concrets.**

  ⇒ Avec un moule, comme avec une classe, on pourra fabriquer plein d'objets, qui pourront être particularisé en fonction de ce qu'on met dans le moule et encore après les avoir fabriqués.

- **Exemple** : le **moule à gaufres**. On peut fabriquer des gaufres différentes selon la pâte qu'on utilise (standard, sans lactose, bio, allégé, etc.). On peut ensuite mettre de la garniture : du sucre ou de la purée de marron.

moule à gaufres = classe Gaufre

- La classe est une « **abstraction** » : s23

  ⇒ Les objets sont concrets.

  ⇒ La classe, c'est le modèle : c'est une abstraction (ou une généralisation).

  ⇒ C'est pour ça que c'est subtil !

Notion de constructeur s24

- La classe est un moule qui permet de fabriquer des objets.
- Le constructeur est une méthode particulière d'une classe : c'est une méthode pour fabriquer un objet.
- On dit « **instancier** » un objet (et pas fabriquer). On dit « **instanciation** » et pas fabrication.

- Le constructeur permet de créer un objet d'une certaine classe, comme on crée une variable d'un certain type :

  ⇒ objet = variable

  ⇒ classe = type

- Le constructeur a toujours le nom de sa classe.
- **Le constructeur peut avoir des paramètres** : ainsi, on pourra fabriquer des objets différents les uns des autres si on le souhaite. Par exemple :

  ⇒ une gaufre standard ou une gaufre bio.

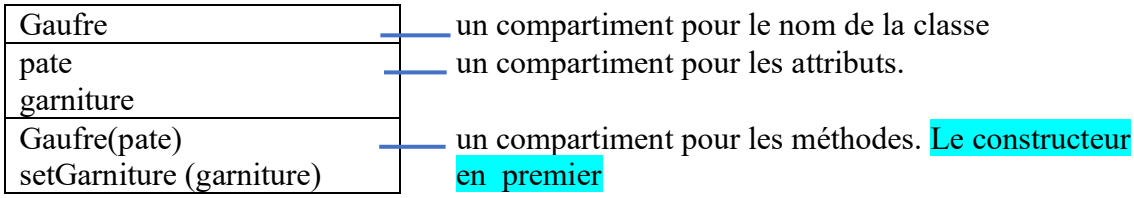
  ⇒ une voiture de telle marque et tel modèle ou d'une autre marque et d'un autre modèle.
- On peut avoir plusieurs constructeurs pour une même classe :

  ⇒ Un moule à petite gaufre.

  ⇒ Un moule à grande gaufre.

La classe Gaufre

➢ ***représentation UML***



➢ ***constructeur : Gaufre (pate)***

- Le constructeur permet de fabriquer un nouvel objet gaufre. On lui passe en paramètre la pâte utilisée.

➢ ***setGarniture (garniture)***

- setGarniture est une méthode qui permet de donner une garniture à une gaufre. On lui passe en paramètre la garniture utilisée.

Convention de nommage

- Le constructeur est une méthode. Il a le nom de la classe et commence par une majuscule.
- On le met en premier dans le compartiment des méthodes.

Convention de nommage s26

- On donne **des noms** de classe, d'attribut et de méthode **les plus explicites que possibles**.
- Le nom de la **classe** a une **majuscule en premier** puis continue en minuscules
- Le nom des **attributs** et des méthodes commence par une **minuscule** et continue en minuscules.
- Quand un nom de classe, d'attribut ou de méthode est composé de **plusieurs parties**, chaque partie commence par une majuscule : c'est le style **camelCase**. En Python, on utilise le **snake_case**.
- Quand un paramètre de méthode correspond à un attribut, on lui donne le nom de l'attribut.
- Les méthodes qui servent à **instancier** un objet sont appelées des « **constructeurs** ».
- Le nom du **constructeur** reprend celui de la **classe**. Il commence donc par une majuscule.
- Les méthodes qui servent à donner une valeur à un attribut s'appellent des « **setter** ».

  ⇒ La signature (= l'en-tête) de ces méthodes est toujours : « **setAttribut (attribut)** ».
- Les méthodes qui servent à récupérer la valeur d'un attribut s'appellent des « **getter** ».

  ⇒ La signature (= l'en-tête) de ces méthodes est toujours : « **getAttribut ()** ».
- Les méthodes qui gèrent une responsabilité spécifique à une classe s'appellent des « **responsabilités** ».

- Pour accéder à un attribut dans une méthode, on le préfixe toujours par « **this.** ».

Code Java de la classe Gaufre (syntaxe simplifiée, sans encapsulation) s27

- Tous les exemples de code de base sont là :
  http://bliaudet.free.fr/IMG/zip/Codes-POO-Exemples-de-bases-multilangages.zip

```
/**
 * Déclaration de la classe Gaufre en Java
 */
class Gaufre {
    String pate;        // standard, bio, sansGluten, allégée, etc.
    String garniture;  // sucre, marron, etc.

    Gaufre(String pate) {
        this.pate = pate;
        this.garniture = null; // par défaut, la garniture est à null
    }

    void setGarniture(String garniture) {
        if (this.garniture == null) {    // on ne set la garniture que si elle valait
    null
            this.garniture = garniture;
        }
    }

    string toString() {
        return "gaufre : " + this.pate + " " + this.garniture;
    }
}
```

Explications du code Java

- On retrouve la **structure du schéma UML** : 1) le nom de la **classe** ; 2) les **attributs** ; 3) les **méthodes**.
- Le **Java** est un **langage typé**. Le type String est utilisé pour les chaines de caractères. La « pate » et la « garniture » sont des chaines de caractères.
- **this**: c'est un paramètre qui permet d'accéder aux attributs de la classe. Le « pate » de « this.pate » c'est attribut « pate » de la classe. Le « pate » de « = pate », c'est le paramètre « pate » du constructeur « Gaufre ( ).
- **null :** le permet d'initialiser une chaine de caractère à rien.
- constructeur : ici, « **pate** » et « **garniture** ». Seule l'attribut « pate » est initialisé avec une valeur par le constructeur. L'attribut garniture est initialisé à « vide » : "". On pourra ensuite lui donner la valeur qu'on veut, de n'importe quel type.
- **La méthode afficher ( )** : souvent, on met une méthode afficher( ) dans les classes pour pouvoir afficher la valeur des attributs d'un objet.
- **System.out.println** ( ) : c'est la fonction Java qui permet d'afficher du texte à l'écran.
- **void** : signifie en Java que la fonction ne retourne rien.

Code Python de la classe Gaufre

```
# Déclaration de la classe Gaufre en Python
class Gaufre:
    def __init__(self, pate):
        self.pate = pate
        self.garniture = ""

    def setGarniture(self, garniture):
        if self.garniture == "" : # on met la garniture s'il n'y en a pas déjà
            self.garniture = garniture

    def toString(self):
        print("gaufre : " + self.pate + " " + self.garniture)
```

Explications du code Python

- **__init__** : c'est le constructeur. En Python, il s'appelle toujours comme ça.
- **self** : c'est un paramètre qui permet d'accéder aux attributs de la classe.
- **Les attributs** sont définis dans le constructeur : ici, « **pate** » et « **garniture** ». Seul l'attribut « pate » est initialisé avec une valeur par le constructeur. L'attribut garniture est initialisé à « vide » : "". On pourra ensuite lui donner la valeur qu'on veut, de n'importe quel type.
- **La méthode afficher(self)** : souvent, on met une méthode afficher( ) dans les classes pour pouvoir afficher la valeur des attributs d'un objet.

2.2 : Utilisation d'une classe : instanciation d'objets, « main » s28

Notions du chapitre

- Objet
- Instanciation d'objet : « new »
- Méthode d'objet et utilisation d'une méthode d'un objet.
- Méthode toString()
- Main d'utilisation d'une classe
- Tests unitaires

Principes s29

- La classe est un moule. La classe (le moule) va nous permettre de créer des objets (avec le constructeur) puis d'utiliser les objets avec leurs méthodes.
- Il y donc **deux actions principales** que permet une classe :

  - **La fabrication des objets** : on parle **d'instanciation.**
  - **L'utilisation de méthode à partir d'un objet** : en théorie on dit qu'on « envoie un message » à un objet quand on utilise une de ses méthodes (la métaphore n'est pas très claire et a été abandonnée).

« main » s30

- La fabrication et l'utilisation peuvent se faire dans le **programme principal** (le « **main** » en référence au langage C) ou dans le code de n'importe quelle méthode d'une autre classe le plus souvent.
- Une fois la ou les classes écrites, on peut écrire un **main pour tester les méthodes de la classe** : c'est ce qu'on appelle un **test unitaire** (un main par classe à tester).
- On peut aussi écrire un programme complet !

Instanciation : new s31

- Quand on fabrique un objet à partir d'une classe, on dit qu'on **instancie** un nouvel objet. C'est une **instanciation** d'objet.
- On utilise en général le mot clé « **new** ». On écrit :

```
objet=new Constructeur( … );
```

- par exemple :

```
gaufre=new Gaufre("standard");
```

Accès aux méthodes : envoi de message

- Quand on a un objet, on peut accéder aux méthodes de sa classe.
- **Quand on fait appel à une méthode d'un objet pour l'objet, on dit qu'on envoie un message à l'objet.**
- par exemple :

```
gaufre.setGarniture("marron");
```

⇒ cette instruction envoie un message à l'objet gaufre.

Affichage : toString()

- La méthode toString() permet de retourner l'état de l'objet sous la forme d'une String
- Ensuite, on peut l'afficher dans le main.
- par exemple :

```
print(gaufre.toString());
```

2.3 : Exemple de « main » objet : codes s32

Exemple du moule à gaufre

Objectifs du « main »

- On veut fabriquer une gaufre standard sucrée et afficher les résultats.
- On veut ensuite mettre du marron sur cette gaufre et vérifier que ça ne fait rien.
- On veut ensuite fabriquer une deuxième gaufre, bio cette fois et aux marrons, puis afficher les résultats.

Code algorithmique

```
Main :
    // fabrication d'une gaufre standard et affichage
    gaufre1=new Gaufre("standard");
    gaufre1.setGarniture("sucre");
    print(gaufre1.toString();

    // ajout d'une autre garniture et affichage
    gaufre1.setGarniture("marron");
    gaufre1.afficher();    // gaufre1 n'a pas changé: on sette s'il n'y en a pas déjà

    // fabrication d'une gaufre bio aux marrons et affichage
    gaufre2=new Gaufre("bio");
    gaufre2.setGarniture("marron");
    print(gaufre2.toString();
```

- **new** : **pour créer un objet**, on utilise le mot clé « new » devant le constructeur. On considère que le constructeur retourne un objet qui est mis ici dans « gaufre1 ». Ensuite un deuxième objet est créé et mis dans gaufre2.
- Accès aux méthodes : pour accéder aux méthodes, on écrit : objet.méthode(…). Ici par exemple : gaufre1.afficher() et gaufre1.setGarniture(« sucre »).

Code Java du « main » utilisant la classe Gaufre

```
public static void main (String args[]) {
        // fabrication d'une gaufre standard sucré et affichage
        Gaufre gaufre1=new Gaufre("standard");
        gaufre1.setGarniture("sucre");
        gaufre1.afficher();

        // ajout d'une autre garniture et affichage
        gaufre1.setGarniture("marron");
        System.out.println(gaufre1.toSring());        // gaufre1 n'a pas changé

        // fabrication d'une gaufre bio aux marrons et affichage
        Gaufre gaufre2=new Gaufre("bio");
        gaufre2.setGarniture("marron");
        System.out.println(gaufre2. toSring());
}
```

- Le code du main est placé dans la fonction « public static void **main(** String args[] **)** »
- « **public static void** » sont à écrire obligatoirement devant le « main » avec **String args[]** en paramètre.
- La fonction est mise dans une classe dédiée au main : on peut mettre le main dans la classe Gauffre.
- Le reste, c'est comme l'algorithme, sauf que les objets gaufre1 et gaufre2 sont typés. On écrit Gaufre gaufre1 = …

Code Python du « main » utilisant la classe Gaufre

```
# "main" python : utilisation de la classe gaufre
#
# fabrication d'une gaufre standard et affichage
gaufre1=Gaufre("standard");
gaufre1.afficher();

# ajout d'une garniture affichage
gaufre1.setGarniture("sucre");
gaufre1.afficher();

# ajout d'une autre garniture et affichage
gaufre1.setGarniture("marron");
gaufre1.afficher();  # gaufre1 n'a pas changé

# fabrication d'une gaufre bio aux marrons et affichage
gaufre2=Gaufre("bio");
gaufre2.setGarniture("marron");
gaufre2.afficher();
```

- C'est comme l'algorithme :
  - Le code du main est placé en dehors de toute fonction.
  - On n'utilise pas le mot clé « new »
  - On n'utilise pas de type.
  - On peut placer ce code dans le fichier de déclaration de la classe. On commence par déclarer la classe, ensuite on écrit le main.

2.4 : Principe d'encapsulation – attributs « private » et méthodes « public » s33

Notions du chapitre

- Objet = état + comportement
- Etat = valeur des attributs
- Comportement = ensemble des méthodes = responsabilité = rôle
- Encapsulation, public, private
- Attributs private : -
- Méthodes public : +

Objet = état + comportement **s34**

Objet = **données** (état) + **méthodes** (comportement, rôles, responsabilités)

- Un objet informatique c'est :

  ⇒ une variable

  ⇒ avec une ou plusieurs valeurs (les attributs) qui seront manipulées (en lecture ou en écriture)

  ⇒ par **les fonctions associées** à l'objet (les méthodes).

- Un « bon objet » en P.O.O. doit représenter un objet du monde réel.

Etat d'un objet : la valeur de ses attributs

- **L'état** d'un objet à un moment c'est **la valeur** de ses attributs à ce moment.
- Certaines valeurs peuvent **changer au cours du temps**. D'autres peuvent **être constantes**.

Comportement d'un objet : ses méthodes (rôle, responsabilités, fonctionnalités)

- Le **comportement** d'un objet c'est l'ensemble de ses **méthodes**. On parle aussi du **rôle** d'un objet ou des **responsabilités** d'un objet. On peut aussi dire : les **fonctionnalités** d'un objet.

Utilisation d'un objet

- Pour utiliser un objet, il faut **commencer par le créer avec le constructeur** (l'instancier). Cette première étape définira un **premier état de l'objet**.
- Ensuite, on peut **lui appliquer toutes ses méthodes** qui permettront, entre autres, de **consulter** ou de **modifier** l'état de l'objet.

Encapsulation : principes et syntaxe s35

Principes

- **L'encapsulation** consiste à **cacher l'état de l'objet** (ses attributs et leurs valeurs).
- Les cacher ? C'est-à-dire ?

  ⇒ On ne pourra accéder à l'état de l'objet, que ce soit pour le consulter ou pour le modifier, qu'en utilisant les méthodes de l'objet.

  ⇒ On dit que les données sont encapsulées par les méthodes.
- Techniquement, pour **encapsuler les attributs**, on les définira en précisant le mot clé « **private** ». Ce mot clé fait qu'on ne pourra plus accéder à cet attribut en dehors de sa classe.
- Puis on définira les **méthodes** avec le mot clé « **public** ». Ce mot clé fait qu'on pourra utiliser les méthodes en dehors de la classe (dans un « main » par exemple). Les méthodes se chargeront de consulter ou de modifier les attributs.

Syntaxe : 2 niveaux de visibilité principaux : private et public

Symbole	Mot-clé	Signification
-	private	Visible uniquement **dans la classe**
+	public	Visible **partout**

Syntaxe : 2 autres niveaux de visibilité : protected et « rien »

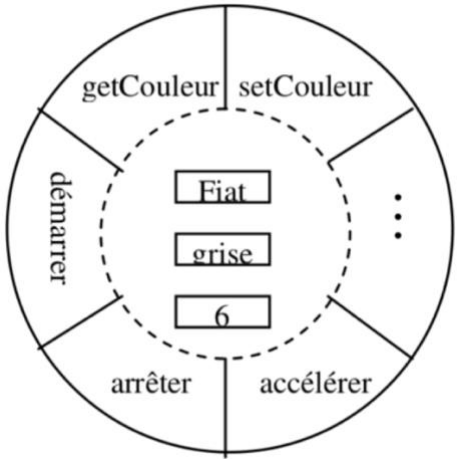
#	protected	Visible uniquement dans la classe et ses enfants
		Visible par toutes les classes du même package (dossier)

Exemple 1 : la classe Voiture

- La classe Voiture :

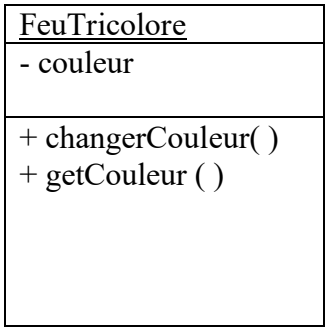
Voiture
- marque - couleur - chevaux
+ démarrer ( ) + arrêter ( ) + accélérer + getCouleur ( ) + setCouleur (couleur) + getMarque() + getChevaux

- un objet voiture :



- Ici, le marque de la voiture (Fiat) et le nombre de chevaux (6) sont des attributs qui ne seront pas modifiés. Il n'y aura donc **pas de** méthodes **setMarque**(marque) ni **setChevaux**(chevaux). Par contre, il y a **une méthode setCouleur**(couleur) pour le cas où on repeint la voiture !

Exemple 2 : le feu de circulation



- On peut appliquer la méthode **changerCouleur()** qui modifie la couleur en fonction de son état (passe du vert au orange, du orange au rouge, du rouge au vert).
- **On évite d'avoir une méthode setCouleur(couleur)** qui permettrait de donner n'importe quelle couleur quel que soit l'état initial et donc permettrait de faire passer du vert au rouge directement.

- Il consiste à :

  - **Éviter tout ce qui peut rendre l'état de l'objet incohérent** et tout ce qui peut correspondre à une évolution incohérente de l'objet.

  - **L'encapsulation au sens restreint** consiste à **éviter un accès direct aux attributs** pour éviter que le programmeur puisse leur donner des valeurs sans faire de vérification de cohérence.

  - **L'encapsulation au sens large** consiste à **éviter toute possibilité d'action incohérente** (par exemple : change la pâte d'une gaufre, ou changer la garniture d'une gaufre). Il faut donc que les actions réalisées par les méthodes le soient de façon cohérente, c'est-à-dire en prenant en compte l'état de l'objet et les règles de fonctionnement de l'objet (par exemple : on ne passe pas d'un feu vert à un feu rouge directement).

  - L'art du programmeur, c'est de gérer proprement l'encapsulation pour **offrir un jeu de méthodes judicieux qui protège contre une utilisation non conforme**.

code Java de la classe Gaufre - syntaxe standard (avec encapsulation : private et public)

```
/**
 * Déclaration de la classe Gaufre en Java
 */
public class Gaufre {
    private String pate;
    private String garniture;

    public Gaufre(String pate){
        this.pate = pate;
        this.garniture = null;
    }

    public void setGarniture(String garniture){
        if (this.garniture == null){
            this.garniture = garniture;
        }
    }

    public void afficher(){
        System.out.println("gaufre : "+this.pate+" "+this.garniture);
    }
}
```

Le mot clé « **public** » s'applique aussi aux classes. Cela veut dire qu'on peut les utiliser.

code Java de la classe GaufreMain – syntaxe standard

```
public class GaufreMain {
    public static void main (String args[]) {
        // fabrication d'une gaufre standard sucré et affichage
        Gaufre gaufre1=new Gaufre("standard");
        gaufre1.setGarniture("sucre");
        gaufre1.afficher();

        // ajout d'une autre garniture et affichage
        gaufre1.setGarniture("marron");
        gaufre1.afficher();     // gaufre1 n'a pas changé

        // fabrication d'une gaufre bio aux marrons et affichage
        Gaufre gaufre2=new Gaufre("bio");
        gaufre2.setGarniture("marron");
        gaufre2.afficher();
    }
}
```

Le mot clé « public » est obligatoire pour le « main ». On le met aussi sur la classe.

codage Python de l'encapsulation

- Le python de gère pas l'encapsulation. Il n'y a pas de mot clé « private » et « public » ou équivalent.
- On retrouve donc le même code que celui déjà présenté mais on crée un fichier avec la classe et un fichier avec le main : le main importe le fichier avec la classe :

```
from Gaufre import *   # on importe le fichier de la classe Gaufre
```

L'art du programmeur objet en Python

- Il consiste à « faire comme si » : faire comme si les attributs étaient encapsulés par des méthodes !
  - On évite de manipuler directement les attributs.
  - On évite les méthodes setAttribut (attribut) inutiles.
- Il consiste aussi à profiter judicieusement des facilités offertes par le Python ! Autrement dit, à ne pas appliquer à la lettre le principe d'encapsulation.

2.5 : Catégorie de méthodes s39

Notions du chapitre

- Méthodes fonctionnelles : constructeurs et responsabilités
  -> ce sont les méthodes « subtiles ».
- Méthodes techniques : getter, setter
  -> ce sont les méthodes « évidentes ».

Catégories de méthodes : 1ère classification = standard **s40**

- **Les constructeurs** : pour créer les objets (création).
- **Les getter** : (ou accesseurs) pour renvoyer l'état d'un objet (consultation).
- **Les setter** : (ou mutateurs) pour modifier l'état d'un objet (modification).
- **Les responsabilités ou rôles** : ces méthodes correspondent aux fonctions de haut niveau permettant la réalisation des fonctionnalités de l'objet.

Catégories de méthodes : 2ème classification = distinction fonctionnel / technique

- **Les méthodes fonctionnelles** : les constructeurs et les responsabilités
- **Les méthodes techniques** : getter et setter.

Méthode de conception s41

- En phase de **conception**, on s'intéresse surtout aux **responsabilités** de la classe. C'est ce qui caractérise le mieux la classe : la voiture peut démarrer, s'arrêter, accélérer, etc. Ça correspond aux usages qu'on a de l'objet.
- On n'est pas obligé d'être précis : on donne déjà une première idée de ce que seront les responsabilités. On affinera au fur et à mesure de la conception jusqu'au MVC.
- Les paramètres des constructeurs sont utiles à préciser pendant la conception.
- Les getter et setter sont moins importants pendant la conception.

  ⇒ On retrouve la programmation procédurale : il s'agit trouver les bonnes fonctions-méthodes.

  ⇒ Mais on commence par définir les objets, puis les fonctionnalités (Use Case) pour ensuite trouver les bonnes fonctions-méthodes qu'on va appliquer aux objets.

2.7 : *(pur technique)* Code : utilisation d'une classe avec un main séparé

Principe

Pour une classe donnée, on peut écrire une autre classe qui contient un main de test.

Exemple en Java

Code Java du « main » utilisant la classe Gaufre dans un fichier séparé : GaufreMain

```
class GaufreMain {
    public static void main (String args[]) {
        // fabrication d'une gaufre standard sucré et affichage
        Gaufre gaufre1=new Gaufre("standard");
        gaufre1.setGarniture("sucre");
        gaufre1.afficher();

        // ajout d'une autre garniture et affichage
        gaufre1.setGarniture("marron");
        gaufre1.afficher();       // gaufre1 n'a pas changé

        // fabrication d'une gaufre bio aux marrons et affichage
        Gaufre gaufre2=new Gaufre("bio");
        gaufre2.setGarniture("marron");
        gaufre2.afficher();
    }
}
```

- On crée une classe pour le main() : GaufreMain
- Cette classe ne contient que le main()
- Elle sert à tester la classe Gaufre : on l'appelle donc GaufreMain
- Le fichier GaufreMain.java est dans le même dossier que Gaufre.java => le compilateur trouvera tout seule la classe Gaufre utilisée par GaufreMain : pas besoin de faire d'importation.

Exercices - Code

Reprendre les exercices 1, 2 et 3 (Gaufre, Television, Voiture) et coder un main() dans une classe séparée.

2.8 : *(pur technique)* Exercices – 2 – Classe – Constructeur – Main

Principes

- Exercice de code : on reprend 2.6 qui n'était que de la conception, et on code tout.
- On écrit les codes des méthodes.
- On écrit le code du main.
- Encapsulation : précisez si les attributs et les méthodes sont private (-) ou public (+).
- Bien réfléchir à la logique d'encapsulation « **au sens large** » : avoir les bonnes méthodes pour réaliser les fonctionnalités de la classe.

3 – EXERCICES - 1 : CLASSE ET MAIN

On va faire les exercices avec un modeleur UML.

1 : Concevez et codez la classe Gaufre avec son main d'utilisation

- Faites le diagramme de classe UML vu dans le cours.
- Ajoutez des attributs de votre choix.
- Ajoutez des méthodes de votre choix.
- Codez la classe et son main d'utilisation.

2 : Feu tricolore

- On reprend la description déjà vue :
  - ⇒ Un feu de circulation (ou feu tricolore ou feu rouge) est un dispositif d'éclairage : la boite avec ses différentes ampoules. On veut simuler son fonctionnement.
  - ⇒ On considère que le feu marche tout le temps. Il est forcément allumé et dans un certain état : vert, orange ou rouge.
  - ⇒ On simplifie donc les choses : il n'y a pas de orange clignotant et pas de feu éteint. Le feu passe forcément du vert au orange, du orange au rouge et du rouge au vert.
- On ajoute la précision suivante :
- On considère qu'à la création, le feu est rouge, ce qui est plus prudent !
- Dessinez la classe UML avec ses attributs et ses méthodes.
- Écrivez un main qui permette de :
  - ➢ Créer un feu.
  - ➢ Afficher la couleur du feu.
  - ➢ Changer la couleur et l'afficher.
  - ➢ Répéter l'opération jusqu'au feu rouge.
  - ➢ Faire ensuite une boucle qui répète le changement et l'affichage du feu rouge au feu rouge (on ne gère pas la durée d'affichage d'une couleur).

3 : Télévision

- Une télévision a une marque et une diagonale. Elle peut être allumée ou éteinte. On peut changer de chaine en donnant le numéro de la chaine ou en montant ou descendant le numéro en cours. On peut monter ou baisser le son. On peut aussi couper le son ou le remettre, avec le même bouton. Quand on remet le son, il revient à son niveau avant d'avoir coupé le son. On peut connaître le niveau du son ainsi que la chaine actuellement sélectionnée.
- A l'allumage, le son est à 3 par défaut et la chaine à 0 par défaut.
- Dessinez la classe UML avec ses attributs et ses méthodes.
- Écrivez un main qui permette de :

  ⇒ Créer une télévision, l'allumer et afficher la chaine au démarrage et le niveau du son au démarrage. Afficher aussi toutes les informations.

  ⇒ Mettre la 9ème chaine et monter le son 3 fois. Afficher le numéro de la chaine et le niveau du son. Afficher aussi toutes les informations.

  ⇒ Couper le son. Afficher le numéro de la chaine et le fait que le son est coupé. Afficher aussi toutes les informations.

  ⇒ Remettre le son. Descendre de 2 chaines. Baisser le son 2 fois. Afficher le numéro de la chaine et le niveau du son. Afficher aussi toutes les informations.

4 : Voiture

- Une voiture a une marque, un modèle, une immatriculation. Elle peut avoir 2 portes, 4 portes ou 5 portes.
- On peut mettre le moteur de la voiture en route et passer des vitesses. Le passage de vitesse se fait forcément de 1 en 1, en montant ou en descendant. On peut aussi passer au point mort à tout moment. On peut aussi freiner ou accélérer.
- Dessinez la classe UML avec ses attributs et ses méthodes.
- Écrivez un main qui permette de :

  ⇒ Créer une voiture. Afficher ses caractéristiques.

  ⇒ Démarrer. Passer en 1ère. Accélérer. Passer en 2ème. Accélérer. Passer en 3ème. Afficher la vitesse. Afficher toutes les caractéristiques de la voiture.

  ⇒ Passer en 1ere. Afficher la vitesse. Afficher toutes les caractéristiques de la voiture.

  ⇒ Freiner. Passer au point mort. Afficher la vitesse. Afficher toutes les caractéristiques de la voiture.

5 : Afficheur de score

- Afficheur de score est un objet qui affiche des scores de matchs de football.
- Par exemple :
  - Pour les matchs à venir, on affiche : Paris – Marseille : à venir
  - Pour les matchs en cours, on affiche : Paris – Marseille : 1-0 - en cours
  - Pour les maths finis, on affiche : Paris – Marseille : 3-2 - terminé
- L'affichage commence toujours par le nom de l'équipe qui reçoit, suivi de l'équipe qui se déplace.
- Il s'agit d'un match sans prolongation ni tirs au but pour nous simplifier la tâche.
- L'afficheur permet de faire évoluer le score au fur et à mesure de la partie.
- Le score ne peut changer que pendant le match, ni avant, ni après.
- On initialise l'objet en lui donnant le nom des deux équipes. Automatiquement, le match est « à venir ».
- Dessinez la classe UML avec ses attributs et ses méthodes.
- Écrivez un main qui permette de :
  - ⇒ S'occuper d'un match Paris – Marseille.
  - ⇒ Créez l'afficheur correspondant et affichez-le.
  - ⇒ Faites démarrez le match et affichez la situation.
  - ⇒ Paris marque. Marseille égalise. Paris marque à nouveau. Codez et affichez la situation au fur et à mesure.
  - ⇒ Le math est fini. Codez la situation et affichez la situation.

6 : Personnage de jeu vidéo

- Un personnage de jeu vidéo a un nom, une force, une localisation, une expérience et des dégâts.
- Le personnage peut se déplacer et combattre un autre personnage. Le déplacement modifie la localisation. Le combat modifie l'expérience et les dégâts.
- Dessinez la classe UML avec ses attributs et ses méthodes.

7 : Distributeur de boissons

- Un distributeur automatique permet de choisir une boisson au choix.
- On peut préciser le niveau de sucre qu'on souhaite avoir.
- Un fois la boisson versée, le distributeur rend la monnaie.
- Le client peut demander la restitution de l'argent qu'il a déjà mis.
- Le fonctionnement est le suivant : quand on clique sur un bouton de boisson, la boisson se préparer si montant fourni est suffisant.
- Quand on a choisi sa boisson et qu'elle se prépare, le distributeur affiche les informations de boisson (son nom et le niveau du sucre) ainsi que le prix, le montant payé et la monnaie prévue.
- On simulera uniquement deux boissons : la boisson numéro 1 : un café à 40 centimes et la boisson numéro 2 : un chocolat à 50 centimes.
- Dessinez la classe UML avec ses attributs et ses méthodes.
- Écrivez un main qui permette de :
- Commander un café. C'est la boisson n°1. Elle coûte 40 centimes. On veut un café sans sucre.
- On met 20 centimes, on choisit le niveau de sucre puis on commande le café.
- On rajoute 50 centimes puis on commande le café.
- Vous voulez un autre café normalement sucré. Après avoir entré 30 centimes (10 puis 20) vous réalisez qu'il vous en manque dix. Vous récupérer votre argent. Affichez la situation.
- Le but est d'afficher :

```
20 centimes encaissé
niveau de sucre baissé : 1
niveau de sucre baissé : 0
La commande demandée est la boisson n°1
Le montant encaissé est trop faible : il manque 20 centimes
70 centimes encaissé.
Le distributeur prépare la boisson n°1
café sucre=0 en cours de préparation
Prix:40 - payé:70 - rendu:30

20 centimes encaissé
30 centimes encaissé
30 centimes rendus
0 centimes encaissé
```

8 : Bissextile

Le code python ci-dessous permet de saisir une année et un mois et de donner le nombre de jours du mois.

```
def estBissextile(annee):
    if annee % 400 == 0: return True
    if annee % 100 == 0: return False
    if annee % 4 == 0: return True
    return False

def nb_jours_mois_annne(mois, annee): # return nombre de jours du
mois
    if mois in [1, 3, 5, 7, 8, 10, 12]: return 31
    elif mois in [4, 6, 9, 11]: return 30
    elif mois == 2:
        if estBissextile(annee): return 29
        else: return 28
    else: return 0 # mauvaise données

# ------------------------------------------------------
# le "main", sans vérification
# ------------------------------------------------------
mois = int(input("mois de 1 à 12: "))
annee = int(input("annee: "))

nb_jours_du_mois = nb_jours_mois_annne(mois, annee)
bissextile = estBissextile(annee)

if bissextile :
    print(f"{annee} est bissextile")
else:
    print(f"{annee} n'est pas bissextile")
print(f"{mois}/{annee} : {nb_jours_du_mois} jours")
```

Reprenez ce code en concevant une classe de POO et un main de test.

Le but sera d'afficher :

```
L'année 2021 n'est pas bissextile
11/2021 : 30 jours

L'année 2020 est bissextile
2/2020 : 29 jours

L'année 2000 est bissextile
2/2000 : 29 jours

L'année 1900 n'est pas bissextile
2/1900 : 28 jours
```

4 – BILAN

Synthèse

- On sait coder une classe (en Java et en Python) et on sait faire le diagramme de classe UML correspondant.

La suite : les relations entre les classes et diagramme des classes métier

- La suite va consister à avancer sur la création du diagramme de classes métier, DCM.
- Pour cela, il faudra avancer sur les associations (ou liaisons ou relations) entre les classes.
- Ce sera le thème du document suivant.
- On verra la notion de classe métier et de diagramme des classes métier.
- On verra la correspondance entre classe métier et table de la BD.
- On sait faire le modèle de données du d'une classe métier.
- On commencera donc la Conception de Système d'Information en UML