TD2 : Conduite de projets
Implémentation des tâches

Compétences

- Utiliser les concepts de classe et d'objet en POO
- Définir le constructeur et les méthodes d'une classe
- Implémenter des tâches

Préliminaire

- Créer un répertoire *VideoTracker* pour le projet.
- Puis créer l'arborescence suivante qui permettra d'organiser les fichiers pour le développement du logiciel *VideoTracker*

```
VideoTracker
   |
   |--- src
   |     |--- models
   |
   |--- tests
```

La Programmation Orientée Objet : POO

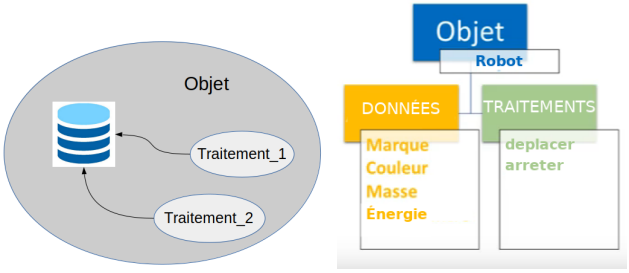
La programmation orientée objet (POO), ou programmation par objet, est un paradigme de programmation informatique. Il consiste en la définition et l'interaction de briques logicielles appelées objets ; un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou un livre. Il possède une structure interne et un comportement, et il sait interagir avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ; l'interaction entre les objets via leurs relations permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes. **Dès lors, l'étape de modélisation revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments du réel sous forme d'objets informatiques.**
Un cours de Programmation Orientée Objet par Xavier Blanc.

Un objet c'est quoi ?

Il est très important de bien définir les objets pour le bon déroulement de notre application, de préciser leur rôle dans l'application, leurs traitements et enfin de définir les données dont ils ont besoin avant de se lancer dans la programmation. Cette première étape essentielle est celle de la **conception / modélisation** des objets.

Caractéristiques d'un objet

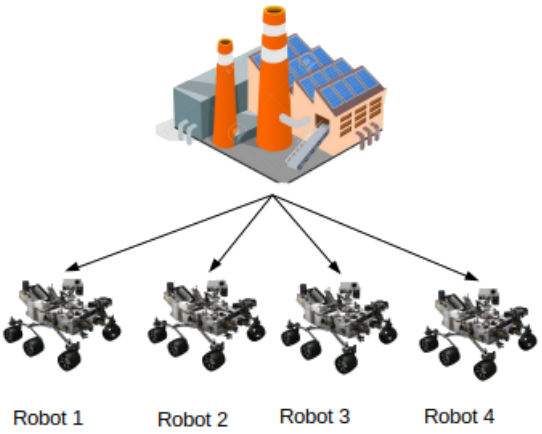
- Son identité
- Ses données
- Le(s) traitement(s) qu'il sait réaliser et qu'il propose aux autres objets



La classe : une usine à objets

Une fois les caractéristiques de notre objet `Robot` définies on peut fabriquer autant d'objets que nécessaires possédant

- ses propres données
- des traitements identiques



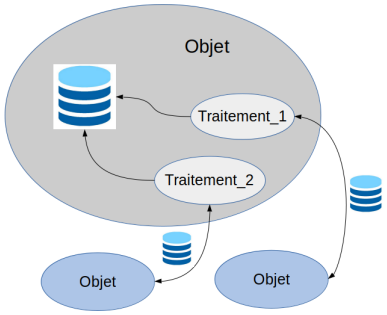
Dans cet exemple chaque objet `Robot` pourra avoir à titre personnel sa couleur, sa marque, sa masse et son énergie mais tous les objets `Robot` seront pourvus des mêmes traitements à savoir : `deplacer` et `arreter`. Cette usine est modélisée par la notion de **classe**.

Communication entre objets

Un objet *A* peut communiquer avec un objet *B* pour lui demander un traitement que *A* ne sait pas faire. Pour envoyer une demande de réalisation de traitement, un objet doit :

L'essentiel

- Connaître l'ID de l'objet qui va réaliser le traitement
- Lui envoyer un message avec le nom du traitement et les données qui lui sont nécessaires
- Recevoir la réponse



Encapsulation et responsabilité

- **Encapsulation** : L'objet protège ses données afin de préserver son intégrité. Seule l'objet peut modifier ses propres données et seul l'objet sait comment ses données son gérées.
- **Responsabilité** : L'objet est responsable des traitements qu'il sait faire et qu'il propose. Ce qui implique que l'objet doit impérativement avoir toutes les données nécessaires pour réaliser un traitement qu'il propose. Être responsable ne vaut pas dire que l'objet doit travailler seul, pour effectuer un traitement il peut avoir besoin de communiquer avec un ou plusieurs autres objets.

L'utilisateur de l'objet `Robot` n'a accès qu'aux traitements, il ne sait pas comment sont organisés les données utilisées par l'objet `Robot`. On pourrait avoir deux objets qui proposent les mêmes traitements mais avec des structures de données différentes.

Les traitements et les données d'un objet doivent former un tout cohérent.

En résumé

Penser une application objet consiste à :

- Identifier les objets nécessaires au bon déroulement de l'application.
- Préciser les traitements de chaque objet dans l'application.
- Définir les données dont ils ont besoin pour réaliser les différents traitements dont ils sont responsables

- Établir les communications
- Faire des objets cohérents

Remarque : Suivant le contexte les objets de l'application vont être plus ou moins facile à déterminer. Pour illustrer ce propos prenons deux exemples très simples :

1. Un bateau de pirate qui vient attaquer une île sur laquelle se trouve un fort défendu par des canons.
2. Un jeu de pendu.

Dans le premier exemple les objets sont presque naturels. Il va y avoir les objets : Bateau, Pirate, Canon... Dans le deuxième exemple les objets sont plus abstraits.

L'application

Une application objet c'est quoi ?

Définition

Une application objet est un ensemble d'objets qui communiquent pour rendre un service global à son utilisateur

Deux questions importantes :

1. Comment l'utilisateur interagit avec l'application donc avec les objets ?
   Tous les langages de programmation propose la possibilité de capter les intéractions de l'utilisateur via échange de messages entre un être humain et un objet informatique.
2. Comment sont construits les objets ? Comment l'application démarre-t-elle ?
   À l'aide d'un objet `Main` qui est un objet un peu particulier permettant de construire tous les objets nécessaires au démarrage de l'application.

Deux règles de bases pour une application objet

CHALLENGE

Les objectifs importants d'une application orientée objet :

- **maximiser la cohérence**, c'est à dire si possible avoir de petits objets. L'idée est de fabriquer des objets dédiés à des traitements bien spécifiques. **Plus les objets sont petits plus la cohérence est forte**. Pour comprendre cette notion prenons comme exemple notre robot. Pour l'instant il sait se déplacer d'un point à un autre. Notre objet est cohérent. Si maintenant on lui ajoute la possibilité de modifier sa vitesse notre objet reste cohérent puisqu'il a besoin d'ajuster celle-ci pour se déplacer et éventuellement faire des économies de carburant. Par contre si on donne à notre robot la possibilité d'effectuer une analyse de roche alors nous avons un objet robot qui sait faire deux choses complètement indépendantes et l'objet devient incohérent. Les analyses doivent être déléguées à un autre objet.
- **minimiser le couplage**, c'est à dire avoir peu (voir pas du tout) de communication entre les objets. Moins il y a de communication plus le couplage est faible. La question à se poser est : **est-ce qu'un objet peut-être trop ou mal couplé ?** Ce qui est sûr c'est qu'un objet qui ne possède aucun couplage est un objet isolé qui présente pas ou peu d'intérêt pour notre application. À l'opposé un objet trop couplé est un objet qui sera difficile à tester car possédant trop de dépendances. Pour finir on pourra également regarder comment les objets sont couplés, par exemple on peut imaginer un couplage sous forme de cycle entre plusieurs objets. Et bien sûr plus le cycle est grand moins le couplage est bon.

  Pour approfondir : Cohérence et couplage

Comment construire une classe en Python ?

Le constructeur d'une classe

Définition

Le constructeur de la classe est la méthode qui sera appelée lors de la création de l'objet

En Python :

- Commence par `def __init__(self):`
- Peut avoir des paramètres transmis lors de la création de l'objet.
- Permet d'initialiser des variables liées à un objet

Les traitements

Chaque traitement d'un objet possède un nom. Ces traitement sont encore appelés **`méthodes`**. Une méthode se définit comme une fonction.

En Python :

- Commence par `def nomMethode(self):`
- Peut avoir des paramètres transmis lors de l'utilisation de la méthode.
- Réalise un un traitement
- Retourner le résultat du traitement avec le mot clé `return`

```
1  class Humain:
2
3      def __init__(self, genre:str):
4
5          self.__genre = genre
6          self.__vie = 100
7
8      # Les méthodes de ma classe
9      def manger(self):
10
11         if self.__vie >= 95:
12             self.__vie = 100
13         else:
14             self.__vie += 5
15
16     def getVie(self):
17
18         return self.__vie
```

Implémentater les tâches du projet

Dans un premier temps nous ne nous occuperons pas de la partie IHM, c'est à dire de l'interface graphique qui permet à l'utilisateur d'intéragir avec le logiciel. Nous allons commencer le code de notre application en définissant les classes :

- Point
- FileRepo

1. Associer à chacune des classes un fichier python rangé dans le bon dossier.
2. Implémenter chaque classe en s'appuyant sur les tâches à réaliser.
3. Dans un fichier `test.py` écrire une méthode `randomPoints(n:int)->list` qui prend en argument un entier `n` et renvoyant un tableau contenant `n` points tirés aléatoirement.
4. À l'aide de ce tableau de `Points` tester la fonction `export2CSV` de la classe `FileRepo`