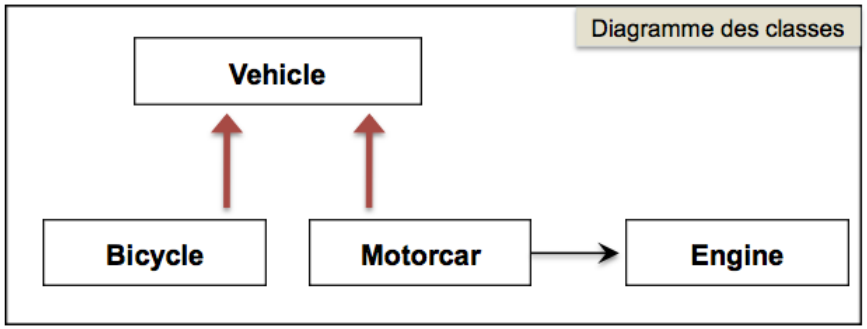
Exercice 3 : Modélisation de véhicule

Objectifs

Classe, héritage
Utilisation du logging
Tests unitaires

Enoncé

On veut modéliser un véhicule et ses variantes voiture et bicyclette
Pour cela nous allons créer quelques classes selon le diagramme suivantd'héritage



- Etape 1

  Définir une classe `Vehicle` (ce sera la classe de base), avec les méthodes `start()`, `stop()`, `brake()`
  Implémenter, par affichage d'un simple message, ces 3 méthodes

  Définir une classe `Bicycle` qui dérive de `Vehicle` et qui une méthode `treadle()` (=pédaler)

  - Définir `incspeed()` en terme `treadle()`
  - Définir `decspeed()` en terme de `brake()` de `Vehicle`

  - Ecrire un code d'utilisation de la classe `Bicycle`

Exemple de résultat d'exécution du code

```
16:29:45.121          Vehicle  INFO    Doing stuff before starting Vehicle
16:29:45.123          Bicycle  INFO    Increasing speed of Bicycle
16:29:45.124          Bicycle  INFO    Treadling with Bicycle
16:29:45.126          Bicycle  INFO    Decreasing speed of Bicycle
16:29:45.128          Vehicle  INFO    Braking action on Vehicle
16:29:45.129          Vehicle  INFO    Doing stuff after stopping Vehicle
```

- Etape 2

  Définir une classe `Motorcar`, qui dérive de `Vehicle` et qui a un object de classe `Engine` => définir une classe `Engine` avec les méthodes `incspeed()`, `decspeed()`, `set_engine_on(on|off)`, `is_engine_on()`

  Pour la classe `Motorcar`

  - Définir les méthodes `set_hand_brake(on|off)` et `is_hand_brake_on()`
  - Définir `incspeed()` en terme `incspeed()` de `Engine`
  - Définir `decspeed()` en terme de `decspeed()` de `Engine` et `brake()` de `Vehicle`
  - Re-définir `start()` en terme `set_engine_on(on)` de `Engine` et `set_hand_brake(off)` de `Motorcar`
  - Re-définir `stop()` de facon symétrique
  - Ecrire un code d'utilisation de la classe `Motorcar`

Exemple de résultat d'exécution du code

```
16:28:37.994          Vehicle  INFO    Starting Motorcar
16:28:37.997          Vehicle  INFO    Doing stuff before starting Vehicle
16:28:37.998           Engine  INFO    Starting Engine
16:28:38.000         Motorcar  INFO    Setting hand brake off
16:28:38.002         Motorcar  INFO    Increasing speed of Motorcar
16:28:38.003           Engine  INFO    Increasing speed of Engine
16:28:38.005         Motorcar  INFO    Driving a car on a sunny day ...
16:28:38.006         Motorcar  INFO    Decreasing speed of Motorcar
16:28:38.007           Engine  INFO    Decreasing speed of Engine
16:28:38.008          Vehicle  INFO    Braking action on Vehicle
16:28:38.009         Motorcar  INFO    Stopping  Motorcar
16:28:38.010         Motorcar  INFO    Setting hand brake on
16:28:38.011           Engine  INFO    Stopping Engine
16:28:38.012          Vehicle  INFO    Doing stuff after stopping
16:28:56.123         Motorcar  INFO    Doing stuff after stopping Vehicle
```

- Etape 3

  Cette exercice sera aussi l'occasion d'écrire un test pour valider l'implémentation de `decspeed()` de engine
  Par exemple en vérifiant que des appels `decspeed()` ne conduisent jamais à une vitesse négative !

  Exemple de résultat d'exécution du code de test (avec unittest)

```
15:11:07.957       EngineTest  INFO    Before running test class EngineTest
15:11:07.958       EngineTest  INFO    Before running test function
15:11:07.958       EngineTest  INFO    Running test on decspeed method
15:11:07.958           Engine  INFO    Starting Engine
15:11:07.958           Engine  INFO    Decreasing speed of Engine
15:11:07.958           Engine  INFO    Decreasing speed of Engine
15:11:07.958           Engine  INFO    Decreasing speed of Engine
15:11:07.958           Engine  INFO    Decreasing speed of Engine
15:11:07.958           Engine  INFO    Decreasing speed of Engine
15:11:07.958       EngineTest  INFO    For now current speed is 0
15:11:07.958           Engine  INFO    Decreasing speed of Engine
15:11:07.958       EngineTest  INFO    After running test function
15:11:07.959       EngineTest  INFO    After running test class EngineTest

Ran 1 test in 0.001s

OK

Process finished with exit code 0
```

Indications

- Pour avoir un bonne lisibilité, faire un fichier par classe et un fichier par code d'utilisation ou de test de classe
  Pour importer la classe 'Vehicle' dans le fichier bicycle.py, faire :
  `from .vehicle import Vehicle`

- Pour le logging
  On définira un logger par classe (avec un nom different pour chaque classe), et en parametre de getlogger, on mettra le nom de la classe via `__class__.__name__`

  ```
  self.my_logger = logging.getLogger(__class__.__name__)
  ```

  Dans le pattern de formattage du logging, on specifira ce nom par %(name)s, ce qui pourrait être
  (à faire une seule fois, par exemple dans la classe Vehicle)

  ```
  MY_FORMAT = "%(asctime)s.%(msecs)03d %(name)16s  %(levelname)-6s  %(message)s"
  logging.basicConfig(format=MY_FORMAT, datefmt='%H:%M:%S', level=logging.INFO)
  ```

- Pour démarrer, un aperçu des classes `Vehicle` et `Bicycle`

```
import logging

class Vehicle:
  # When specifying datefmt, (see below), ms are lost unless you add then with
 %(msecs) pattern
  MY_FORMAT = "%(asctime)s.%(msecs)03d %(name)16s  %(levelname)-6s  %(message)
s"
  logging.basicConfig(format=MY_FORMAT, datefmt='%H:%M:%S', level=logging.INFO
)

  def __init__(self):
    self.my_logger = logging.getLogger(__class__.__name__)
    return

  def start(self):
    self.my_logger.info("Doing stuff before starting Vehicle")

  def stop(self):
      ...

  def brake(self):
    ...

  # end of class Vehicle
  ...
```

```
from .vehicle import
import logging

class Bicycle(Vehicle)
  def __init__(self)
    self.my_logger2 = logging.getLogger(__class__.__name__)
    Vehicle.__init__(self)
    ..

...
```