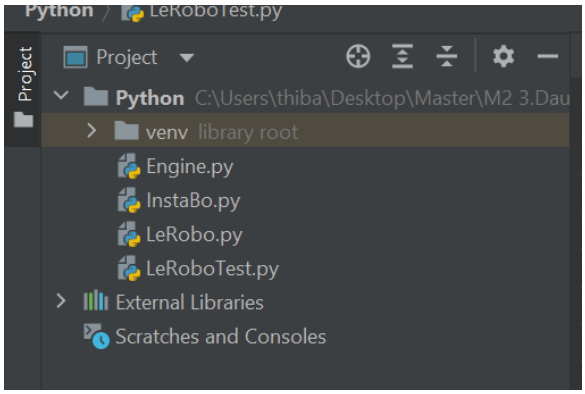
TD2 : Bobo et Coco sur InstaBo

Thibault Bougerol et Bastien Lesouef

Continuons notre histoire avec Bobo LeRobo mais cette fois-ci sur le langage python. On choisit comme IDE Pycharm community qui est téléchargeable ici : https://www.jetbrains.com/fr-fr/pycharm/download/#section=windows

On récupère les classes créées préalablement et on les organise dans un package :



Implémentation d'une association bidirectionnelle :

Pour cela, on va donner naissance au fabuleux réseau social InstaBo. En effet, ce réseau social va pourvoir permettre aux robots qui y sont connectés de discuter entre eux. Ainsi, un robot peut être associé à 0 ou 1 réseau social (parce qu'on veut bien être social mais pas trop non plus) et un réseau social peut être associé à 0 ou plusieurs robots (qui feront partie de sa communauté) en fonction de son succès.

```
class InstaBo :

    def __init__(self):
        self._community = list()

    def add_robo(self, robo):
        self._community.append(robo.get_name())

    def get_community(self):
        return self._community
```

Pour le test, on va connecter deux robots (Bobo et Coco) au réseau InstaBo et essayer de les faire discuter. Le test doit retourner True :

```
def setUp(self):
    self.insta = InstaBo()
    self.robo1 = LeRobo()
    self.robo1.set_name('Bobo')
    self.robo2 = LeRobo()
    self.robo2.set_name('Coco')
    self.robo1.subscribe_to_instabo(self.insta)
    self.robo2.subscribe_to_instabo(self.insta)

def test_talk(self):
    talk = self.robo1.talk_to_robo(self.robo2)
    self.assertEqual(True, talk)

def test_talk_2(self):
    talk = self.robo2.talk_to_robo(self.robo1)
    self.assertEqual(True, talk)
```

Et voici le message qui nous dit que le test a été passé avec succès :

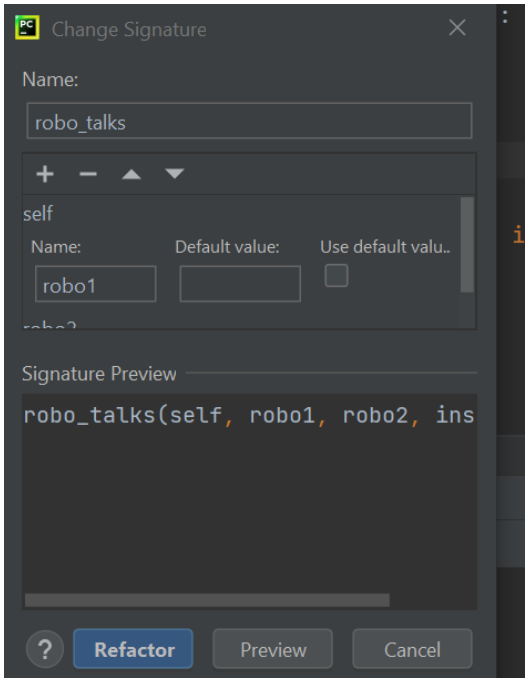
```
Ran 1 test in 0.006s

OK
```

Utilisation de deux techniques de refactoring :

Vous pouvez trouver ici le tutoriel pour l'utilisation du refactoring sur Pycharm :
https://www.jetbrains.com/help/pycharm/product-refactoring-tutorial.html#e35a158c

Par exemple, on peut se placer au niveau d'une fonction et appuyer sur ALT + F6 et changer les noms des arguments de la fonction ainsi que l'ordre :



On change le nom des robots en robox et roboy plutôt que robo1 et robo2 et on met le réseau social en premier dans les arguments puis on clique sur Refactor :

```
def robo_talks(self, insta, robox, roboy):
    #Check if both robots are in InstaBo
    if(robox.get_name() and roboy.get_name()) in insta.get_community():
        return True
    else:
        return False
```

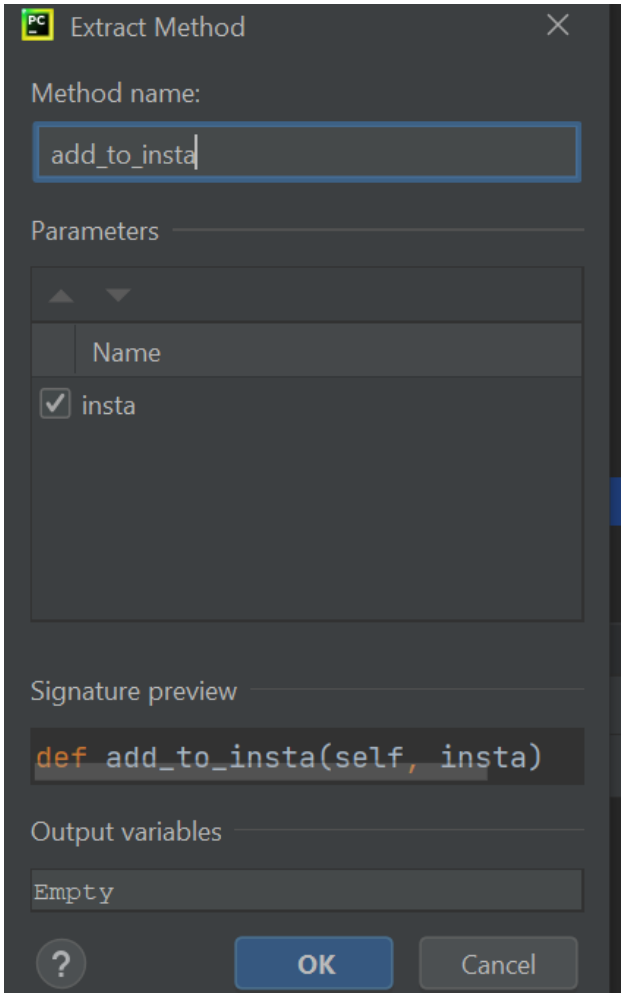
Une autre technique de refactoring intéressante est l'extraction de méthode. Imaginons que nous ayons mis trop d'actions dans notre constructeur :

```
def __init__(self, insta):
    self._name = "unknown"
    self._oil_level = 100
    self._is_alive = True
    self._ready_to_walk = False

    insta.add_robo(self)
    self.insta = insta
```

On voit que l'on peut aisément prendre les deux dernières lignes et les mettre dans une méthode à part pour des raisons de lisibilité. Pour cela, on sélectionne les deux lignes et appuient sur Ctrl + Alt + M. On choisit le nom de la méthode et elle est créée puis appelée

dans le constructeur.



Junit Test infected :

On va reprendre notre méthode talk et la modifier un petit peu.

```
def robo_talks(self, robo2):
    #Check if the Robo is in a social network
    if not hasattr(self, 'insta'):
        return

    #Check if both robots are in InstaBo
    if(self.get_name() and robo2.get_name()) in self.insta.get_community():
        return True
    else:
        return False
```

Dorénavant, il est évident que si un robot peut parler à un autre robot, alors cet autre robot doit pouvoir parler au premier robot. C'est ce qu'on va mettre dans notre test :

```
def test_talk(self):
    talk = self.robo1.robo_talks(self.robo2)
    talk2 = self.robo2.robo_talks(self.robo1)
    talk_union = talk and talk2
    self.assertEqual(True, talk_union)
```

Ce test fonctionne bien mais on va également le lancer de la ligne de commande :

```
Ran 1 test in 0.003s

OK

Process finished with exit code 0
```

Pour lancer un test de la ligne de commande, il suffit de suivre ce qui est écrit dans la documentation unittest :
https://docs.python.org/3/library/unittest.html

Test modules can be specified by file path as well:

```
python -m unittest tests/test_something.py
```

```
(base) C:\Users\thiba\Desktop\Master\M2 3.Dauphine\Cours\2. S2\Méthodes agiles\TD1\Python>python -m unittest LeRoboTest
py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK

(base) C:\Users\thiba\Desktop\Master\M2 3.Dauphine\Cours\2. S2\Méthodes agiles\TD1\Python>
```

Loi de Murphy : « Tout prend plus de temps que vous ne le pensez. »
En effet, on pensait se coucher beaucoup plus tôt…