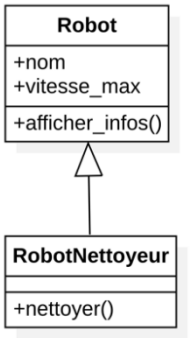
TD3 – Héritage

Objectifs du TD : héritage, surcharge, polymorphisme, classe abstraite (`pass`)

1. Héritage Simple

Description de l'exercice : héritage et ajout d'une méthode
Questions :



1. Écrire en python les classes `Robot` et `Robot_nettoyeur` (qui hérite de `Robot`), avec les méthodes `afficher_infos` qui affiche le nom et la `vitesse_max`. La méthode `nettoyer` affiche « Je nettoie ! »
2. Créer un programme principal qui instancie :
   - un robot qui s'appelle `robot1` et qui avance à une vitesse max de 5 m/s, et
   - un robot nettoyeur, `robot_nettoyeur1`, qui avance à une vitesse max de 2m/s.

   puis appelle les méthodes `afficher_infos` pour les 2 robots.
3. Appelez ensuite dans le programme principal la méthode `nettoyer` pour le bon robot.

2. Héritage et ajouts d'attributs

Description de l'exercice : extension de la classe avec de nouveaux attributs
Questions :

1. Ecrire le constructeur de la classe `Robot_nettoyeur` pour ajouter 3 nouveaux attributs : `pos_charge_x`, `pos_charge_y`, et `niveau_charge`, représentant la position en x, y de la station de charge, et le niveau de charge de la batterie. La position sera définie par l'utilisation et passé en paramètre, mais le niveau de charge sera initialisé à 0
2. Créer la méthode `afficher_infos` dans la classe `Robot_nettoyeur`, qui permet d'appeler la méthode `afficher_infos` de la classe mère, et d'afficher la position de la station de charge, ainsi que le niveau de charge.

3. Polymorphisme et classe abstraite

Questions :

1. Écrire une nouvelle classe `Robot_aspirateur` qui hérite de `Robot_nettoyeur`, et contenant un nouvel attribut `etat_reservoir` initié à « vide ».
2. Définir une nouvelle méthode `aspirer` qui affiche sur la console « j'aspire »
3. Redéfinir la méthode `nettoyer` qui appeler la méthode `aspirer` dans le cas d'un`Robot_aspirateur.`
4. Réaliser le diagramme de classes avec les classes `Robot`, `Robot_nettoyeur`, `Robot_aspirateur`
5. Si on écrit le programme principal suivant, qu'est-ce qu'il s'affiche à l'écran ?

```
robot_aspi1 = Robot_aspirateur("Roomba", 4, 12,30)

robot_aspi1.afficher_infos()

robot_aspi1.nettoyer()
```

6. Ajoutez dans la classe `Robot` une méthode `executer_tache` ne contenant pas de code (écrire simplement `pass`), et ajoutez-la à la classe `Robot_nettoyeur` qui appelle `nettoyer`. Pourquoi dit-on que la classe `Robot` est une classe abstraite ? Quel est l'intérêt de déclarer une méthode avec le mot-clé `pass` ?

4. Conception complète

Questions :

1. Donnez le diagramme de classes de la conception suivante :
   on souhaite ajouter une classe `Salle` ayant comme attributs un numéro et une surface, et qu'un robot Nettoyeur soit toujours associé à une salle (c'est_à_dire qu'il ait un attribut de type `Salle`). Un robot nettoyeur est donc affecté à une seule salle. Vous devez donc créer une méthode `affecter_salle` à la classe `Robot_nettoyeur`

   On souhaite également ajouter une nouvelle classe `Robot_rondier` qui hérite de la classe `Robot`, et qui possède un attribut alerte initialisé à `false`, et une méthode `executer_tache` qui affichera « je fais ma ronde ». Un robot rondier n'est pas affecté à une salle.

   On souhaite enfin ajouter une classe `Robot_serpillere` qui hérite de la classe `Robot_nettoyeur`, et possédant également une méthode `executer_tache`

2. Écrire le code du programme principal permettant d'initialiser une liste de dix robots aspirateurs (avec une initialisation des attributs de façon automatique et incrémentale), une liste de 10 salles (avec une incrémentation automatique des numéros), et d'affecter à chaque robot une salle.
3. Ajouter le code qui convient pour que le programme principal affiche les infos de tous les robots, y compris les salles qui leur sont associées.