TP N°1 — Pilotage d'un robot WifiBot

1. Objectifs

- Savoir déclarer et utiliser les énumérations en JAVA.
- Comprendre la notion de constructeur.
- Représentation sous forme de classes UML.
- Savoir utiliser les sockets

2. Documents fournis

- Le sujet du TP au format papier et pdf.
- La classe WifiBot (à compléter) permettant le pilotage du robot.
- Un document spécifiant le protocole de communication avec le WifiBot.
- Un document de présentation du WifiBot.
- L'exécutable du programme de simulation du robot.

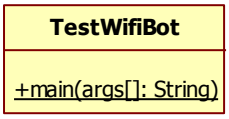
Tous ces documents sont situés sur le serveur dans le répertoire « _travail/TP1 ».

3. Cahier des charges

On souhaite réaliser une petite application en mode console permettant de piloter à distance le Robot « WifiBot » à partir d'un PC.
La liaison entre le PC et le robot peut être de type filaire ou via une connexion Wifi.

On va dans un premier temps, concevoir notre application en simulant les déplacements du WifiBot. Une fois la simulation au point, nous utiliserons les méthodes fournies (*connexion*, *déconnexion* et *envoyerCommande*) pour piloter le robot.

Q 1. Lisez le document de présentation du robot qui vous est fournie.
Q 2. Créez sous Netbeans une application java nommé « *WifiBot* ». On nommera la classe principale « *TestWifiBot* ».



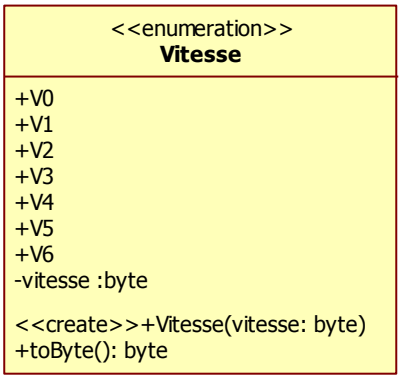
Représentation UML de la classe « TestWifiBot »

4. Enumération « Sens » et « Vitesse»

Les paramètres influant sur le comportement du robot sont les suivants :

- `sens`, le sens de déplacement (le robot peut aller en sens avant et arrière),
- `vitesse`, la vitesse de déplacement (de valeur entière comprise entre 0 et 60).

Q 1. Ajoutez au projet une énumération nommée « Vitesse » en vous appuyant sur la représentation UML suivante :

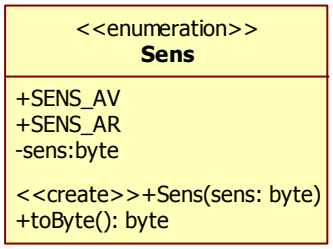


Représentation UML de l'énumération « Vitesse »

L'attribut vitesse pourra prendre plusieurs valeurs constantes définies dans l'énumération. On souhaite associer les valeurs suivantes à chacun des identifiants :
V0(0), V1(10), V2(20), V3(30), V4(37), V5(50), V6(60).
La méthode « *toByte()* » retourne une valeur entière correspondant à la vitesse du *WifiBot* .
Exemples :

- Vitesse.V0.toByte() renvoie 0 comme valeur de vitesse.
- Vitesse.V5.toByte() renvoie 50 comme valeur de vitesse.

Q 2. Ajoutez au projet une énumération nommée « Sens » en vous appuyant sur la représentation UML suivante :



Représentation UML de l'énumération « Sens

L'attribut sens pourra prendre deux valeurs constantes définies dans l'énumération. On souhaite associer les valeurs suivantes à chacun des identifiants :
SENS_AV(64), SENS_AR(0).
La méthode « *toByte()* » retourne une valeur entière correspondant au sens du *WifiBot* .
Exemples :

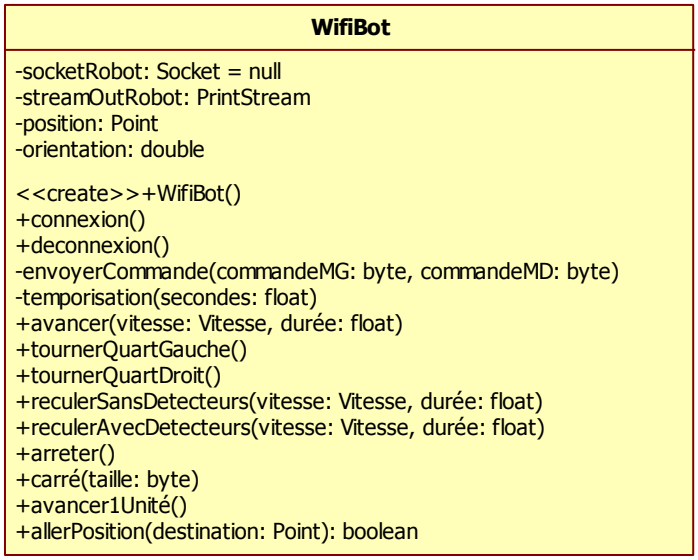
- Sens.SENS_AV.toByte() renvoie 64 comme valeur de sens.
- Sens.SENS_AR.toByte() renvoie 0 comme valeur de sens.

5. Réalisation de la classe WifiBot

On souhaite simuler dans un premier temps le déplacement du *WifiBot*. Le robot est équipé de deux moteurs (4 en réalité), un à gauche et l'autre à droite. La manière dont le robot se meut est liée au comportement des moteurs :

- Le robot avance (ou recule) lorsque les deux moteurs tournent dans le même sens à la même vitesse.

- Il tourne sur lui-même lorsque les deux moteurs tournent dans le sens contraire à la même vitesse,
- Il ne bouge pas lorsqu'aucune des conditions précédentes n'est réalisée.



Représentation UML de la classe « WifiBot »

Certaines méthodes comportent un paramètre « *durée* », correspondant à la durée de déplacement.

Une fois les paramètres réglés, le démarrage du robot peut être déclenché. Il en résulte alors une mise en rotation des deux moteurs dans le sens avant (*SENS_AV) ou arrière(SENS_AR)* pendant une durée de déplacement «*durée*» (exprimé en secondes) avec une vitesse variant de *V0(0) à V6(60).*

5.1 Réalisation des méthodes avancer, reculer, arreter et temporisation

Q 1. On souhaite dans un premiers temps que le robot puisse se déplacer sur une ligne à une vitesse donnée fixe pendant une période de déplacement donné.

Complétez la classe « `WifiBot` » fournie. Celle-ci contiendra une méthode *Avancer*, une méthode *temporisation*,et une méthode *reculer*. Pour connaitre la signature de la méthode, on se réfèrera à la représentation UML de la « `WifiBot` ».

Pour ceci, vous devrez dans un premier temps écrire les constructeurs adéquats, puis définir les méthodes liées au comportement du moteur

On testera la classe en réalisant un menu
Q 3. Testez la classe Moteur en utilisant une classe « *TestMoteur* » contenant une méthode main. On s'assurera que les paramètres du moteur fournis sont valides en les testant au niveau du constructeur de la classe Moteur.

Exemple de scénario de test : On souhaite faire un déplacement en sens avant à une vitesse de 30 (V3) sur une durée de 3s.

On appelle la méthode *avancer* de la classe robot. Elle affichera un message sur la console pour indiquer que le robot avance puis fera appel à une méthode *tempo(int seconde)* chargé de réaliser une tempo de 2 secondes.

Enfin, on arrêtera le robot en mettant la vitesse des moteurs à zéro.

On recommencera ce scénario de test en faisant cette fois-ci reculer le robot pendant le même laps de temps. Puis on l'arrêtera à nouveau.

Temporisation :

Pour bloquer l'exécution d'un programme pendant un temps fixé, utiliser l'instruction `Thread.sleep( T )` où `T` est exprimé en millisecondes. Cette instruction doit se faire dans un try…catch.

5.2 Réalisation des méthodes tourner un quart gauche et droite

On suppose que lorsque le robot tourne sur lui-même à une vitesse de V4(40) pendant une durée de 1,4s, il effectue un quart de tour.

5.3 Réalisation de la méthode reculer avec détecteurs

Q 2. On suppose également que lorsqu'il avance à une vitesse de 1 pendant une durée de 1s, il avance d'une unité d'espace sur le sol.

Modifiez la classe `Robot`.. La classe contiendra une méthode *avance1unité*, une méthode *recule1unité* et une méthode *tourne1_4*.

Testez la classe Robot en utilisant une classe Test contenant une méthode main.

Exemple de scénario de test : Durée d'activité =12s.

On souhaite faire un déplacement en sens avant à une vitesse de 5 sur une durée de 5s. Puis faire reculer le robot à une vitesse de 3 sur une durée de 3s. Faire tourner le robot un quart de tour vers la gauche. Le faire avancer à une vitesse de 2 pendant 4s. Faire tourner le robot un quart de tour vers la droite. Le faire avancer à une vitesse de 1 pendant 3s.

La valeur de la durée d'activité sera décrémentée à chaque déplacement. On autorisera les déplacements tant que la durée d'activité sera supérieure à zéro.

5.4 Réalisation des méthodes « carré » et avancer d'une unité

Q 3. Ajoutez une méthode `carre` qui fait tracer un carré au robot à partir de sa position initiale. La taille du carré à tracer est passée en paramètre. On donnera la représentation UML de la classe Robot.
Testez la classe WifiBot en utilisant une classe TestWifiBot contenant une méthode main.

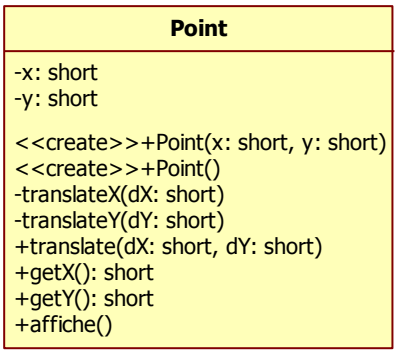
6. Déplacement du robot

On souhaite maintenant faire évoluer le robot sur un plan. Le sol sur lequel évolue le robot est représenté par un tableau d'entier de 2 dimensions (Point Plan2D [][]) où chaque case représente un emplacement. Une unité de sol représente un déplacement d'une case.

Chaque élément du tableau est un entier qui peut prendre trois valeurs :

- entier = 1 → Le robot est présent à cet emplacement
- entier = 0 → Emplacement vide
- entier = -1 → Un obstacle est présent à cet emplacement

6.1 Réalisation d'une classe Point



Représentation UML de la classe « Point »

Réaliser une classe *Point* permettant de représenter un point sur un axe. Chaque point sera caractérisé par un nom (de type S*tring)* et une abscisse (de type *int).* On prévoira :

- un constructeur recevant en arguments le nom et l'abscisse d'un point,
- un constructeur par défaut (sans parmètres).
- une méthode *void affiche()* imprimant (en fenêtre console) le nom du point et son abscisse,
- une méthode *void translateX (int dX)* effectuant une translation définie par la valeur de son argument.

Écrire un petit programme utilisant cette classe pour créer un point, en afficher les caractéristiques, le déplacer et en afficher à nouveau les caractéristiques.

Modifiez le programme afin que le point puisse se déplacer également en ordonnée. La classe aura au final une méthode affiche pour l'affichage du nom et des coordonnées, une méthode *void translateX (int dX)* pour les déplacements en x, une méthode *void translateY (int dY)* pour les déplacements en y et une méthode *void translate (int dX , int dY)* pour les déplacements en x et y.

6.2 Afficher les coordonnées du WifiBot pendant ses déplacements

Q 1. Rajoutez au robot une méthode `ballade` qui fait se *balader* le robot en évitant les obstacles.

6.3 Déplacement du WifiBot vers un emplacement donné

Analyse UML

Diagramme de deploiement

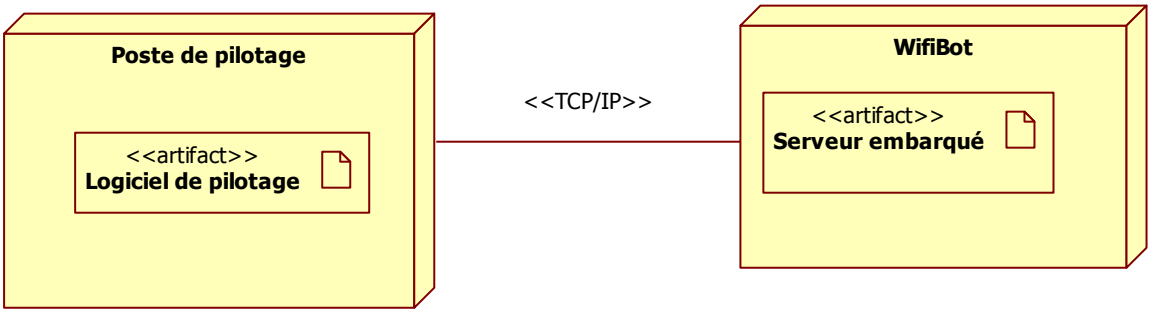


Diagramme des cas d'utilisation

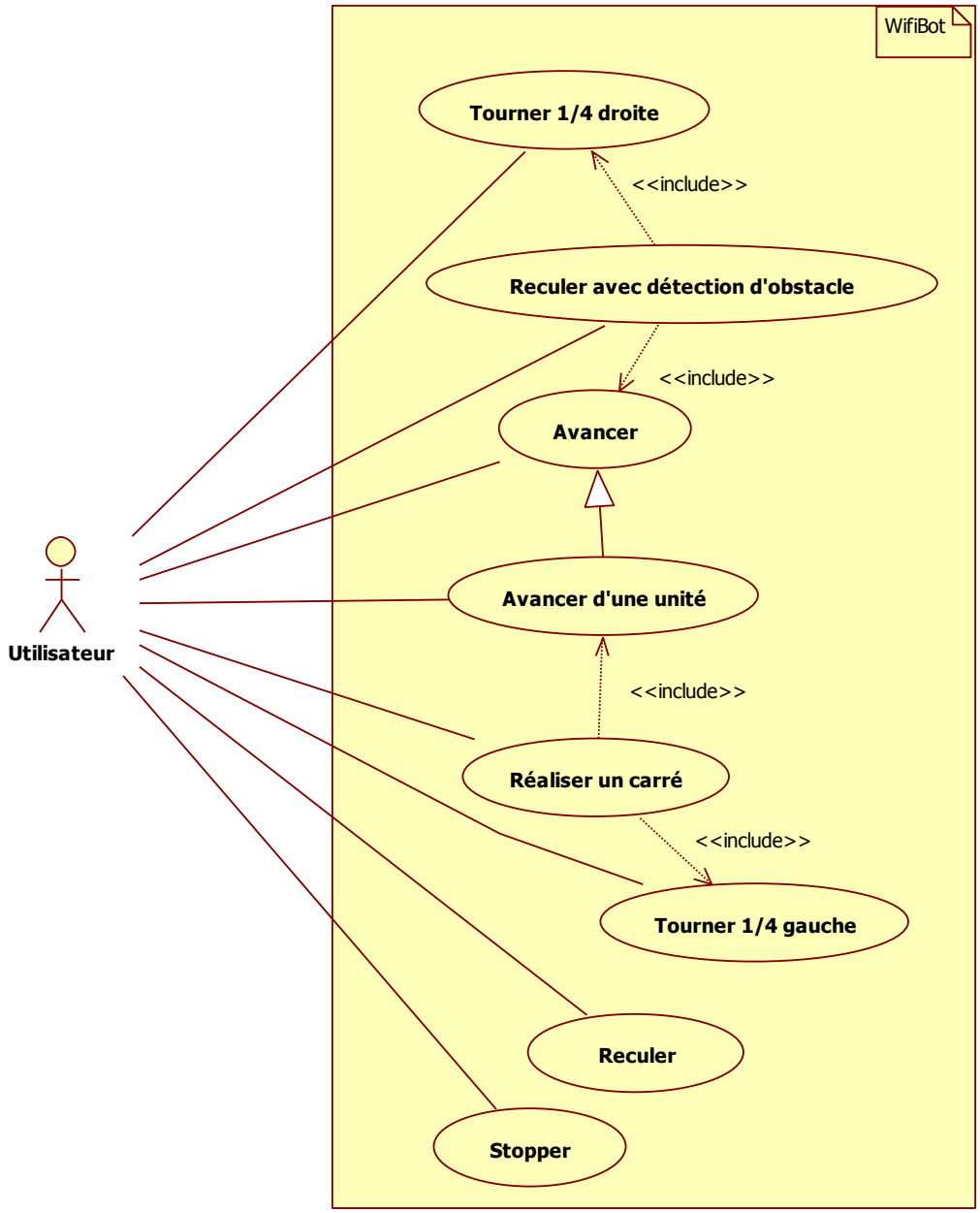


Diagramme des classes