RAPPORT

Ce rapport contient une explication générale de la méthodologie employée pour répondre aux exercices des différentes séances.

ANGELI Nicolas

TP1

Une fois la clase Robot créée pour pouvoir ajouter la méthode robotCount() on créée une variable « public static int n=0 ; » static afin qu'elle ne soit pas dupliquée à chaque création d'objet, public pour que la variable soit accessible de l'extérieur de la classe (pas utile ici car on pourra juste appeler la méthode robotCount() qui est « public static robotCount() » static est nécessaire pour que la méthode puisse être appelée dans le main.
Dans la suite du tp on utilise un thread qui au démarrage affiche les phrases « Vous avez 10 secondes ! » et « Entrez le mot de passe » et attends 10 secondes avant de mettre fin au programme. En parallèle dans le main une boucle while compare l'entrée clavier au mot de passe prédéfinit, affiche si le code est correct ou non à chaque tentative et arrête le programme s'il est correct.

- On veut réaliser un programme qui affiche un message toutes les 3 secondes. La solution que vous avez proposée fonctionne t elle si il y a plusieurs taches de différentes périodes ? Comment mettre une échéance ? Comment réagir en cas de dépassement ?

On peut créer un thread qui afficherait un message toutes les 3 secondes en mettant dans la méthode run une boucle infinie

```
while(true){
  try{
    Thread.sleep(3000);
    System.out.println("message");
  }
  catch(Exception e){}
}
```

Cependant on remarque que la méthode sleep(...) met certes le thread en pause pendant un certain temps mais que celui-ci s'ajoute au temps d'exécution du programme de ce fait chaque tache de période différente induira une marge d'erreur proportionnelle au temps d'exécution. Il existe des méthodes de la classe thread pouvant mettre fin à un thread mais celles-ci comportent des risques. Pour mettre une échéance on peut utiliser une condition dans la boucle while de sorte à ce que le thread s'arrête lorsque la condition n'est plus vérifiée.

TP2

EX1

« Button.waitForPress (); » l5 permet d'attendre la pression d'un bouton avant de continuer (et dans notre cas terminer) le programme.

EX2

Pour cet exercice on stocke dans une variable le résultat de la méthode getVoltage() de la classe Battery. Variable que l'on pourra diviser par 10 (batterie complètement chargée à 10V) puis multiplier par 100 pour obtenir un le résultat en pourcentage (revient finalement à multiplier par 10).
D'où la commande « System.out.println("Battery : " + f*10 + "%"); »

EX3

Dans cet exercice on crée une classe constituée d'une procédure main(…) qui teste si la batterie est rechargeable ou non par un appel de la méthode isRechargeable() de la classe Battery, puis affiche un message sur la console déportée en conséquence, tout cela après avoir demandé à la brique d'attendre la connexion (étape que l'on exécutera avant d'afficher la première fois avec RConsole.println() dans nos classes (nécessite l'import de RConsole)).

EX4

La classe Ex4 étend la classe RealtimeThread et redéfinit sa méthode run().
La méthode run() est appelée toutes les secondes par le biais des PeriodicParameters passés au constructeur de la classe.
La méthode run() affiche le temps écoulé (ms) depuis le début du programme sur la console déportée.
La méthode main(…) crée une nouvelle instance de la classe et la lance en tant que thread.

EX5

Le moteur peut être contrôlé grâce aux méthodes forward(), backward() et stop() de l'objet motor. L'exécution du programme est suspendue à chaque fois que l'on attend l'appui sur un bouton grâce à la méthode waitForPress() de la classe Button.

EX6

La classe Ex6 est constituée d'une procédure main qui crée et lance deux RealtimeThreads par le biais des constructeurs des clases TM et TA.
Les classes TM et TA prolongent la classe RealtimeThread.
Dans la classe TM le constructeur met le moteur en marche et crée un RealtimeThread dont la procédure run() teste à chaque période si le moteur tourne ou non. Si le moteur est arrêté le programme se finit au test suivant.
Dans la classe TA le constructeur crée un RealtimeThread dont la procédure run() arrête le moteur et le programme lorsqu'un bouton est pressé.

TP3

La classe RTLineLeader étend la classe RealtimeThread.
Son constructeur récupère par le biais de ses paramètres la priorité et la période du thread temps réel qu'il va ensuite créer via un appel au constructeur de la classe parente.
Thread dont la procédure run() exécute la procédure masque à chaque période.

La méthode "run" récupère la valeur du capteur de ligne et la fournit à la fonction "masque" qui va déterminer la vitesse et le sens de rotation des moteurs B et C en fonction de la valeur du capteur. Si aucune ligne n'est détectée, les moteurs s'arrêtent. Si la ligne est détectée, les moteurs tournent dans une direction ou l'autre en fonction de l'emplacement de la ligne par rapport au capteur. Si la ligne est au milieu du capteur, les deux moteurs tournent à la même vitesse et dans la même direction.

Comme la classe RTLineLeader, la classe RTPullEvent étend la classe RealtimeThread et son constructeur récupère par le biais de ses paramètres, la priorité et la période du thread temps réel qu'il va ensuite créer via un appel au constructeur de la classe parente.
Dans sa méthode run, une instance de la classe NXTLineLeader est créée avec le capteur S2. Si le bouton du capteur est enfoncé, alors les moteurs B et C sont arrêtés. Ensuite, la méthode waitForNextPeriod est appelée pour mettre le thread en attente jusqu'au prochain intervalle de temps.

Dans la classe RTLauncher, la méthode main crée un objet de la classe RTLineLeader ainsi qu'un objet de la classe RTPullEvent. Les threads de ces objets sont ensuite lancés par le biais des méthodes start().
Dans le bloc :

```
try{
  rt_ll.join();
  rt_pe.join();
}catch(InterruptedException e){
 // RConsole.println("ie in launcher");
}
```

La méthode join() est appelée sur les objets rt_ll et rt_pe et permet de bloquer l'exécution du thread principal (ici, celui de la procédure main()) jusqu'à ce que leurs threads se terminent. Si les threads sont interrompus, l'exception est gérée dans le bloc catch.

- Quelles priorités relatives donner aux deux tâches RTLineLeader et RTPullEvent pour empecher RTLineLeader de redémarrer les moteurs ?

Dans la méthode main, une instance de RTPullEvent est créée avec une priorité de 16 tandis qu'une instance de RTLineLeader est créée avec une priorité 15.

TP4

EX1

Avec ce programme après avoir affiché « Drive Circle » et avoir attendu l'appui d'un bouton le robot entame une trajectoire circulaire, le programme s'arrête au prochain appui de bouton.

EX2

Dans cet exercice nous avons utilisé les équations données dans l'énoncé ainsi que la méthode getTachoCount() de la classe Motor pour calculer les coordonnées et les afficher à chaque période.

Pour RTOdometry, on remplacera le while(true){…} de l'Ex2 par :
while(a<pi/2 || Math.abs(x-R)>0.1 || Math.abs(y)>0.1 || Math.abs(a)>0.1){…} afin que le robot s'arrête une fois revenu assez proche de sa position initiale. (on pourra remplacer 0.1 par une constante adaptée à la précision de l'odométrie).

ANGELI Nicolas