1 Enoncé

On souhaite réaliser un logiciel gérant le fonctionnement d'un réseau de lignes et de rames de métro. Le système doit permettre de connaître l'état du réseau : quelles sont les lignes de métro avec leurs arrêts et quelles sont les rames en circulation. Chaque rame circule sur une ligne unique. Une ligne peut comporter plusieurs rames en circulation. Les lignes sont définies par la liste de leurs arrêts. Les arrêts sont identifiés par leur position GPS. Un système de positionnement GPS fournit à chaque rame sa position en temps réél.

Scénario 1 L'utilisateur du système obtient l'ensemble des rames présentes sur le réseau.

Scénario 2 Pour une rame choisie, l'utilisateur du système peut obtenir la position GPS de cette rame.

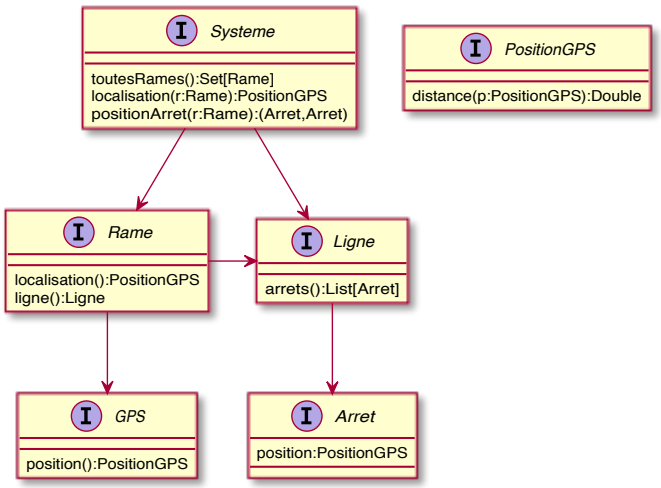
Scénario 3 Pour une rame donnée, l'utilisateur du système peut savoir entre quels arrêts se situe la rame.

1. Proposez un diagramme de classes UML découpant le développement du logiciel de gestion de la rame. Faites figurer les objets/classes/traits, les relations et les opérations/méthodes (avec leurs types) que vous jugez nécessaires. Proposez un découpage (en objets/classes/traits) suffisamment fin pour faire travailler **au moins** 6 équipes de développeurs sur des objets/classes séparés.
2. Expliquez comment les scénarios proposés au dessus s'exécutent sur votre diagramme. En particulier, en déroulant le scénario, expliquez quelles informations circulent entre vos objets/classes/traits et **en utilisant quelles opérations/méthodes**.

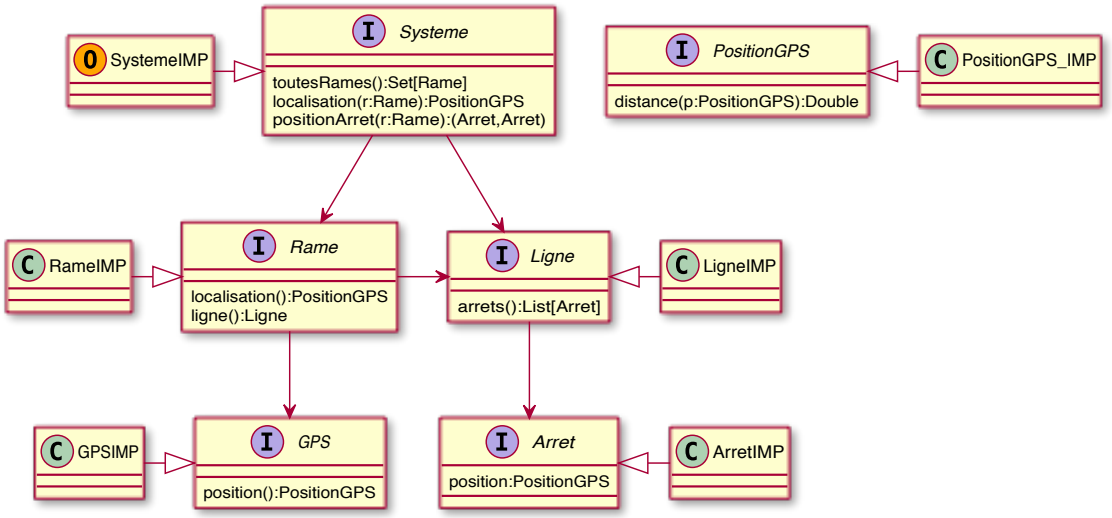
2 Correction

2.1 Diagrammes de classe

Voici quelques éléments de correction. Les explications sont données dans la vidéo. Concernant les diagrammes de classes UML, les voici. Voici le premier diagramme faisant apparaître uniquement les 6 interfaces/traits et donc les 6 équipes correspondantes.



Maintenant, voici le diagramme complété avec les objets/classes implantant les différents traits. Seul élément intéressant ici : pour le système on n'a pas besoin d'une classe car il n'y aura qu'un seul objet implantant le système.

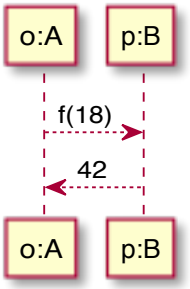


2.2 Exécution des scénarios

Dans cette partie, l'idée est d'expliquer pour chaque scénarios quels sont les objets qui sont utilisés et de quelle façon. En particulier, quelles sont les méthodes appelées et avec quels paramètres. Pour cette partie, je donne une version textuelle et également une version sous la forme d'un diagramme de séquence UML qui est, généralement, plus lisible et plus précise. Ces diagrammes décrivent les intéractions entre des objets. Par exemple, supposons que :

- l'on dispose d'un objet `o` de classe `A` et un objet `p` de classe `B` ;
- `o` détient une référence sur `p`, autrement dit `o` "connaît" `p` ;
- `o` appelle la méthode `f` sur l'objet `p` avec comme paramètre 18, c'est à dire que `o` fait `p.f(18)` ;
- le retour de `p.f(18)` est 42.

Le scénario précédent peut être résumé par le diagramme suivant :



2.2.1 Scénario 1

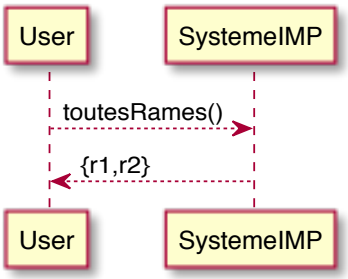
L'utilisateur du système obtient l'ensemble des rames présentes sur le réseau.

Version textuelle de l'exécution Sur l'objet SystèmeIMP l'utilisateur appelle

```
SystemeIMP.toutesRames()
```

qui rend l'ensemble des rames présentes sur le réseau. L'objet SystemeIMP connaît toutes les rames présentes sur le réseau.

Diagramme de séquence



2.2.2 Scénario 2

Pour une rame choisie, l'utilisateur du système peut obtenir la position GPS de cette rame.

Version textuelle de l'exécution Sur l'objet SystèmeIMP, pour localiser la rame r1, l'utilisateur appelle

```
SystemeIMP.localisation(r1)
```

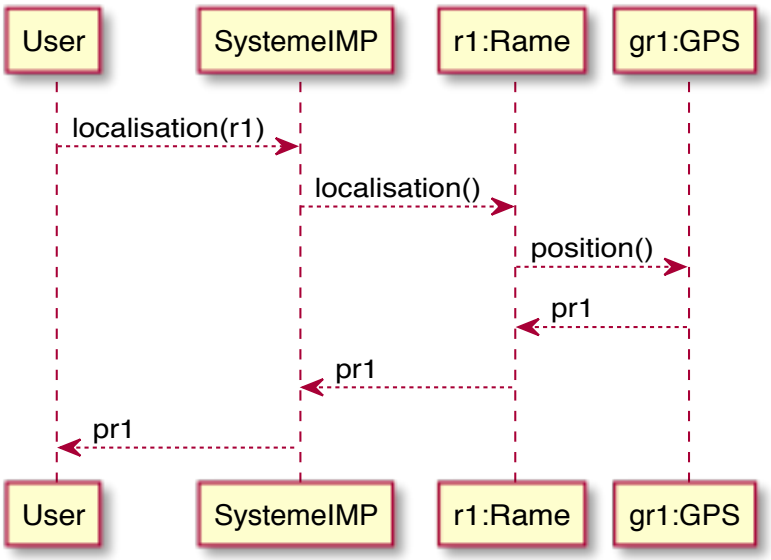
SystemeIMP connaît toutes les rames (y compris r1) mais pas leur localisation. SystemeIMP appelle

```
r1.localisation()
```

pour connaître la localisation de la rame r1. La rame r1 ne connaît pas sa position mais dispose d'un objet de classe GPS, appelons le gr1, auquel elle peut la demander :

```
gr1.position()
```

Diagramme de séquence



2.2.3 Scénario 3

Pour une rame donnée, l'utilisateur du système peut savoir entre quels arrêts se situe la rame.

Version textuelle de l'exécution SystemeIMP procède comme en 2) pour connaître la position GPS de la rame r1. Reste à connaître la position de tous les arrêts de la ligne de la rame r1 pour trouver les deux plus proches. D'après le diagramme, chaque rame connait sa ligne. Appelons l1 la ligne de la rame r1.

```
l1.arrets()
```

donne la liste des arrêts de la ligne l1. Soit List(a1,a2,a3) la liste des arrêts. Pour chaque arrêt on peut connaître sa position :

```
pa1= a1.position
pa2= a2.position
pa3= a3.position
```

Ensuite on calcule toutes les distances entre pa1, pa2, pa3 et pr1 et on retourne les deux arrêts ayant une distance minimale par rapport à pr1.

```
d1= pr1.distance(pa1)
d2= pr1.distance(pa2)
d3= pr1.distance(pa3)
```

Diagramme de séquence