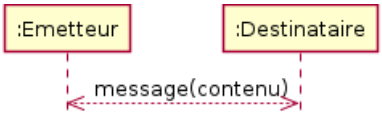
UML td5 : Diagrammes de séquences

Pierre Gérard, IUT de Villetaneuse, **DUT informatique, S2 2013**

Messages synchrones et asynchrones

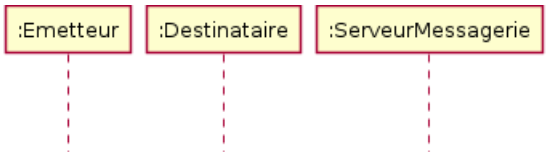
Un système de messagerie procède par envois de messages directement de l'émetteur vers le destinataire, sans intermédiaire.

1. Précisez le type de message transitant de l'émetteur au destinataire dans le diagramme ci-dessous.



2. Qu'est-il nécessaire de prévoir dans le diagramme de classes pour que cette solution soit correcte ?

On suppose maintenant que l'on dispose d'un serveur de messagerie faisant permettant de stocker temporairement les messages. Les lignes de vie sont donc les suivantes :



3. Représentez les messages modélisant l'envoi et la réception d'un message.

4. Qu'est-il nécessaire de prévoir dans le diagramme de classes pour que cette solution soit correcte ?

Séquences et activités

Considérons le pseudo-code suivant :

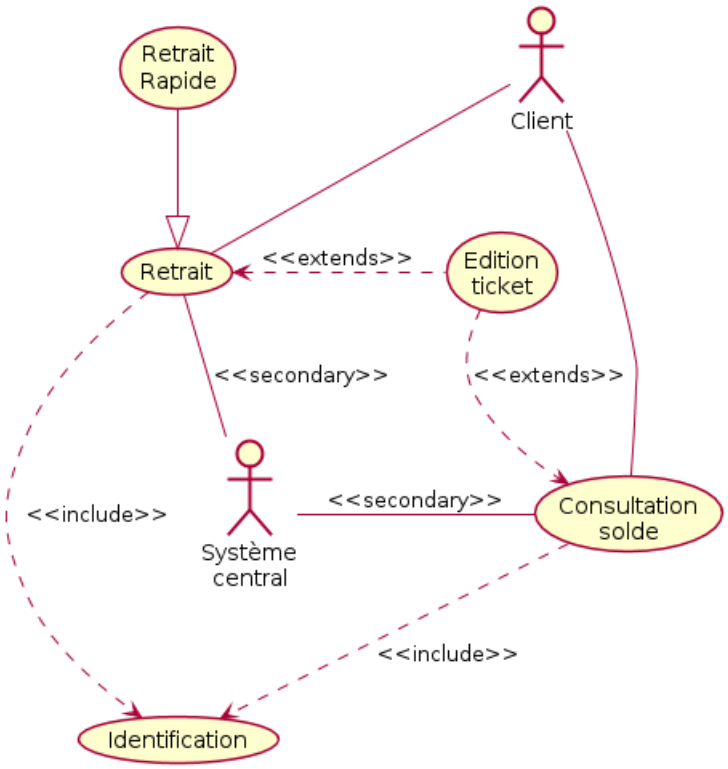
```
1 class Robot {
2 private :
```

```
    BrasArticule brasArticule ;
public :
    void chercherPiece() {
        brasArticule.deplier() ;
        brasArticule.replier() ;
    }
}

class BrasArticule {
private :
    Pince pince ;
public :
    void deplier() {
        ...
        pince.fermer() ;
    }
    void replier() {
        ...
        pince.ouvrir() ;
    }
}

class Pince {
private :
    ...
public :
    void fermer() { ... }
    void ouvrir() { ... }
}

Début programme principal
    Robot robot ;
    robot.chercherPiece() ;
Fin programme principal
```

1. Donnez un diagramme de classes correspondant au code ci-dessus.
2. Donnez un diagramme de séquences pour modéliser la séquence d'activités après l'appel de `robot.chercherPiece()` dans le programme principal. Représentez les barres d'activités des différentes lignes de vie.

Documentation de cas d'utilisation

Les diagrammes de séquence sont souvent utilisés pour documenter des cas d'utilisation. Dans un TD précédent, nous avons produit le diagramme suivant pour modéliser les fonctionnalités attendues d'un DAB (distributeur automatique de banque) :



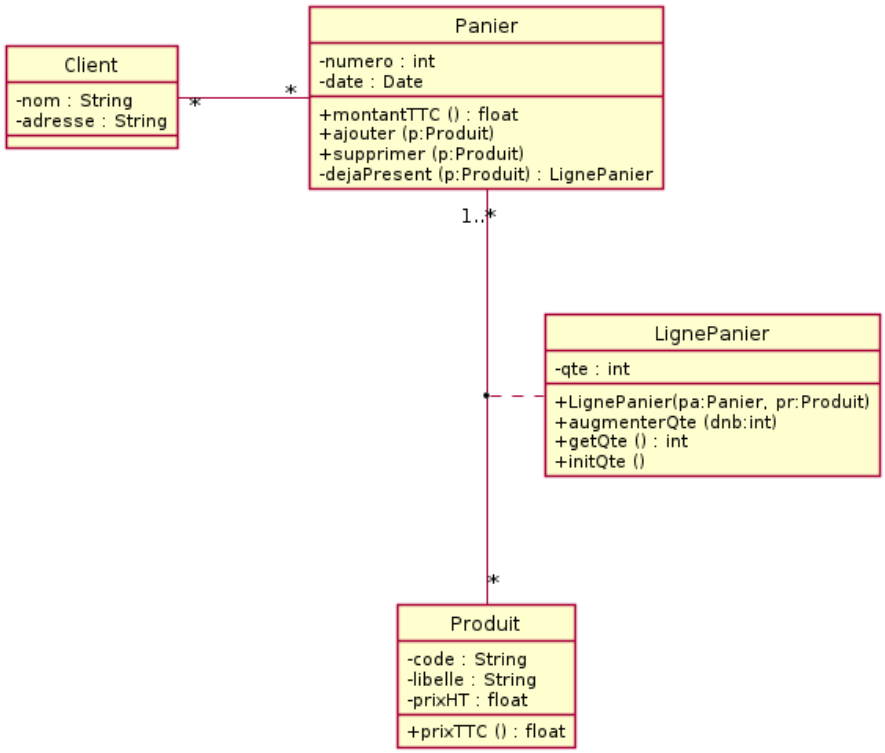
La description textuelle du cas d'utilisation `Retrait` inclut les éléments suivants : 1. Le client demande un retrait 2. On procède à l'identification du client 3. Le DAB demande au client quel compte débiter 4. A l'invitation du DAB, le client donne le montant à débiter 5. Le DAB procède aux vérifications nécessaires auprès du système central 6. Le client est informé des suites données à sa demande 7. Le DAB demande au client s'il souhaite un ticket 8. Si le retrait a été autorisé, le DAB restitue la carte bancaire 9. Immédiatemement après que le client ait pris la carte, le DAB met les billets à disposition 10. Retrait autorisé ou non, si le client a demandé un ticket, il est édité

On suppose qu'il existe déjà des diagrammes de cas d'utilisation intitulés `Identification` et `EditionTicket` documentant les cas d'utilisation correspondants.

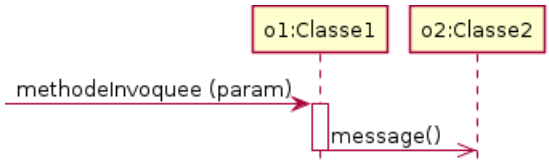
- Donnez un diagramme de séquences formalisant en UML le scénario ci-dessus.

Interactions internes à un système

Considérons le diagramme de classes suivant :



On cherche à décrire la manière dont le système se comporte lorsque la méthode `Panier::ajouter (p:Produit)` est invoquée. On peut pour celà employer un diagramme de séquence plus rigoureux que dans le cas précédent, avec tous les éléments syntaxiques correctement définis et les activités correctement représentées. Le premier message peut partir du côté du diagramme, sans source définie, à la manière du début de diagramme suivant où `o1.methodeInvoquee(param)` est exécutée :



Dans le cas considéré, lorsqu'on ajoute un produit à un panier, on vérifie qu'il n'y est pas déjà présent. S'il n'existe pas on le crée mais dans le cas contraire, on ne fait qu'en augmenter la quantité dans la ligne corrrespondante. La méthode `Panier::dejaPresent(p:Produit)` retourne `null` quand le produit est absent, et une référence vers la ligne correspondant dans le cas contraire.

1. Proposez un diagramme de séquences pour modéliser les interactions internes au système lorsque a méthode `Panier::ajouter (p:Produit)` est invoquée.

2. Ecrivez en Java les éléments en rapport avec ce que vous venez de modéliser.

Module d’UML