Courriers

La réalisation de ce sujet va se faire par la réalisation de « versions » successives. Chaque version doit être réalisée complètement avant de passer à la suivante. Une bonne conception doit faciliter l'intégration des extensions apportées par les versions successives.

On se propose de modéliser la distribution du courrier : chaque jour, des habitants envoient des courriers à d'autres habitants de la même ville ou d'une autre ville.
Ces courriers doivent être déposés dans des boîtes aux lettres et ces courriers sont distribués le lendemain.

Villes et habitants

Dans cet exercice, les villes sont considérées comme des bureaux distributeurs : elles réunissent toutes les lettres postées le jour même dans la boîte aux lettre des la ville, et les distribuent le lendemain. Il convient donc, comme dans la réalité, de distinguer la boîte aux lettres, où sont postés les courriers, de la sacoche du facteur, qui contient les courriers à distribuer.
Une **ville** (*city*) possède un nom, des rues et des habitants (*inhabitant*).

- La méthode `addLetter` ajoute un courrier à l'ensemble des courriers à distribuer (dans la boîte aux lettres – *mailbox*).
- La méthode `distributeLetters()` s'occupe de la distribution du courrier.

Les **habitants** sont caractérisés par leur nom, la ville[1] où ils habitent et un compte en banque (*bank account*). On doit pouvoir créditer (*credit*) ou débiter (*debit*) ce compte d'un montant (*amount*) donné.

- la méthode `sendLetter` permet à un habitant d'envoyer un courrier à un autre habitant,
- la méthode `receiveLetter` gère la réception d'un courrier par un habitant.

Les courriers

Un **courrier** (*letter*) comporte un expéditeur (*sender*) et un destinataire *(receiver*) (deux habitants) et un contenu (*content* – a priori, n'importe quoi). **A tout courrier est associée une action, qui est accomplie lorsque ce courrier est reçu par son destinataire**. Enfin, tout courrier a un coût.

Version 1.0

Dans cette première version, un courrier peut être :

- Une **lettre simple** (*simple letter*) : son contenu est un texte. Le coût d'une lettre simple est fixe (0,5 par exemple).
- Une **lettre de change** (*bill of exchange*) : elle contient une certaine somme d'argent, qui, à la réception de la lettre, sera prélevée du compte de l'expéditeur et versée au compte du destinataire. Le destinataire, reconnaissant, envoie alors une lettre simple de remerciement à son bienfaiteur. Le coût d'une lettre de change est de 1 + 1% de la somme transférée.

Q 1 . Modélisez les différents éléments décrits ci-dessus sous la forme de diagramme de classes UML.

Q 2 . Comment garantir à la compilation que le contenu d'une lettre simple est de type texte ?

Q 3 . Proposez un test pour vérifier que « *A tout courrier est associée une action, qui est accomplie lorsque ce courrier est reçu par son destinataire* ».

Q 4 . Donnez en parallèle

- l'entête des types `Letter` et `SimpleLetter`
- le code des méthodes `distributeLetters` de `City` et `receiveLetter` et `sendLetter` de `Habitant`.

[1] Ce sera suffisant dans cet exercice comme gestion de l'adresse.

Version 2.0
Il doit maintenant être possible d'envoyer tout courrier **en recommandé** (*registered letter*). Dans ce cas lors de la réception du courrier, **en plus** de l'effet lié au courrier, un accusé de réception (*acknowledgement of receipt*) est alors envoyé à l'expéditeur en retour. Un recommmandé a un surcoût de 15% par rapport au courrier initial.

Q 5 . Modélisez les courriers recommandés, comment s'intègrent-ils à la version 1.0 ?

Q 6 . Donnez l'entête de la classe `RegisteredLetter`.

Q 7 . Proposez un test unitaire qui permet de tester la consigne « *un accusé de réception (...) est alors envoyé ...* » ?

Version 3.0
On ajoute maintenant la prise en compte de **courriers urgents** (*urgent letter*). Tout courrier peut être "transformé" en courrier urgent (y compris les recommandés). L'action est inchangée et le coût est doublé.

Q 8 . Comment prendre en compte cette modification ? Faites une proposition qui s'intègre à la version 2.0 .

Version 4.0 – « Chaîne des naïfs » Chacun connaît les chaînes de courrier censées vous rendre millionnaire. Le fonctionnement est le suivant : vous recevez une lettre contenant une liste composée de quatre habitants et la lettre vous demande de :

- envoyer 5 euros à chacun des noms de la liste, sous la forme d'une « lettre de change » ;
- supprimer le premier nom de la liste et mettre votre nom en dernière position ;
- envoyer la lettre (contenant la nouvelle liste de noms) à 10 habitants ;

En principe, vous devez tôt ou tard devenir riche...

Q 9 . Faites une proposition de conception pour prendre en compte ces nouveaux courriers (*fool letter*). Intégrez la à la version 3.0.

NB : on peut supposer l'existence d'une méthode de `City` qui fournit un de ses habitants aléatoirement. On peut également supposer qu'à chaque réception d'un tel courrier, il y a 10% de chance que le destinataire y réponde.

Simulations (Projet)

Ce projet doit être l'occasion pour vous d'appliquer les principes vus en cours (KISS, DRY, SOLID, etc.) ainsi que d'expérimenter les méthodologies présentées. Inspirez-vous des principes de l'*extreme programming*, par exemple en pratiquant avec votre binôme le pair programming, en expérimentant le TDD, etc.
Exploitez les possibilités offertes par votre IDE pour améliorer en continu la conception de votre code (par exemple, dans Eclipse, étudiez le menu Refactor (Alt+Shift=T) et les facilités d'*extract* qu'il propose).
Inspirez-vous dans votre pratique du travail présenté dans les vidéos suggérées en cours.
Lors de votre travail appliquez-vous également à adopter une meilleure pratique d'utilisation de git et à soigner vos messages de commits.

Q 10 . Créez une ville V et ses habitants (pour simplifier on considèrera que tous les habitants de la simulation habitent la ville V).

Chaque jour, pendant k jours, les courriers de la ville sont distribués puis n habitants envoient un courrier à un autre habitant de la ville, choisi aléatoirement. Vous ferez varier les types des courriers envoyés. Eventuellement, ces courriers peuvent engendrer des réponses (accusés de réception, lettres de remerciement, etc.).

Simulez la collecte et la distribution du courrier pendant les k jours puis tant qu'il reste des courriers à distribuer.

NB : Le caractère « urgent » des courriers n'est pas pris en compte dans cette simulation, urgent ou non un courrier arrive à destination le lendemain du jour où il est posté, simplement un courrier urgent coûte le double...

Q 11 . Créez une seconde simulation avec une « chaîne à naïfs » initiée par un seul courrier.

Dans cette simulation,

- il y a une certaine probabilité qu'un habitant donné réponde (15% par exemple),
- un habitant dont le solde du compte est négatif ne répond pas.

La simulation commence avec l'envoi d'un seul courrier et elle se poursuit tant qu'il y a des courriers à distribuer (c'est-à-dire tant que la chaine est active).

A la fin de la simulation le solde de l'habitant le plus riche est affiché. Faites plusieurs simulations, produisent-elles des millionnaires ?

Exemple de trace d'exécution d'une simulation

Avec les classes mises en place on devrait être en mesure de réaliser une simulation d'envoi de courriers dont une trace pourrait être (avec ici $k = 4$ + deux jours nécessaires pour distribuer les courriers induits) :

```
Jour 1
>>> hab96 envoie courrier0 de change recommande (cout:1.84) a hab33
>>> hab13 envoie courrier2 (cout:1.0) a hab78
>>> hab56 envoie courrier3 recommande (cout:1.15) a hab2
>>> hab43 envoie courrier5 URGENT (cout:2.0) a hab2
```

```
Jour 2
courrier0 de change (cout:1.6) de [valeur = 10.0 euros] envoye par hab96 reçu par hab33
>>> hab33 envoie courrier7 de remerciement (cout:1.0) a hab96
>>> hab33 envoie courrier8 accuse de reception (cout:1.0) a hab96
courrier2 (cout:1.0)[bla bla] envoye par hab13 reçu par hab78
courrier3 (cout:1.0)[bla bla] envoye par hab56 reçu par hab2
>>> hab2 envoie courrier9 accuse de reception (cout:1.0) a hab56
courrier5 (cout:1.0)[bla bla] envoye par hab43 reçu par hab2
>>> hab1 envoie courrier10 URGENT (cout:3.2) a hab13
>>> hab35 envoie courrier12 (cout:1.0) a hab69
>>> hab85 envoie courrier13 de change recommande URGENT (cout:3.68) a hab83
>>> hab3 envoie courrier15 de change (cout:1.6) a hab2
```

```
Jour 3
courrier7 de remerciement (cout:1.0)[merci pour courrier0 de change] envoye par hab33 reçu par hab96
courrier8 accuse de reception (cout:1.0)[accuse du courrier0 de change recommande] envoye par hab33 reçu par hab96
courrier9 accuse de reception (cout:1.0)[accuse du courrier3 recommande] envoye par hab2 reçu par hab56
courrier10 de change (cout:1.6) de [valeur = 10.0 euros] envoye par hab1 reçu par hab13
>>> hab13 envoie courrier16 de remerciement (cout:1.0) a hab1
courrier12 (cout:1.0)[bla bla] envoye par hab35 reçu par hab69
courrier13 de change (cout:1.6) de [valeur = 10.0 euros] envoye par hab85 reçu par hab83
>>> hab83 envoie courrier17 de remerciement (cout:1.0) a hab85
>>> hab83 envoie courrier18 accuse de reception (cout:1.0) a hab85
courrier15 de change (cout:1.6) de [valeur = 10.0 euros] envoye par hab3 reçu par hab2
>>> hab2 envoie courrier19 de remerciement (cout:1.0) a hab3
>>> hab95 envoie courrier20 de change (cout:1.6) a hab6
>>> hab52 envoie courrier23 URGENT (cout:2.0) a hab1
>>> hab46 envoie courrier25 URGENT (cout:2.0) a hab91
>>> hab60 envoie courrier26 (cout:1.0) a hab24
```

```
Jour 4
courrier16 de remerciement (cout:1.0)[merci pour courrier10 de change] envoye par hab13 reçu par hab1
courrier17 de remerciement (cout:1.0)[merci pour courrier13 de change] envoye par hab83 reçu par hab85
courrier18 accuse de reception (cout:1.0)[accuse du courrier13 de change recommande] envoye par hab83 reçu par hab85
courrier19 de remerciement (cout:1.0)[merci pour courrier15 de change] envoye par hab2 reçu par hab3
courrier20 de change (cout:1.6) de [valeur = 10.0 euros] envoye par hab95 reçu par hab6
>>> hab6 envoie courrier27 de remerciement (cout:1.0) a hab95
courrier21 (cout:1.0)[bla bla] envoye par hab52 reçu par hab1
>>> hab1 envoie courrier28 accuse de reception (cout:1.0) a hab52
courrier23 (cout:1.0)[bla bla] envoye par hab46 reçu par hab91
...  courrier25 (cout:1.0)[bla bla] envoye par hab60 reçu par hab24
>>> hab23 envoie courrier29 (cout:1.0) a hab25
>>> hab55 envoie courrier30 de change recommande (cout:1.84) a hab44
>>> hab98 envoie courrier32 (cout:1.0) a hab33
>>> hab14 envoie courrier33 URGENT recommande (cout:2.30) a hab53
```

```
Jour 5
courrier27 de remerciement (cout:1.0)[merci pour courrier20 de change] envoye par hab6 reçu par hab95
courrier28 accuse de reception (cout:1.0)[accuse du courrier21 recommande] envoye par hab1 reçu par hab52
courrier29 (cout:1.0)[bla bla] envoye par hab23 reçu par hab25
courrier30 de change (cout:1.6) de [valeur = 10.0 euros] envoye par hab55 reçu par hab44
>>> hab44 envoie courrier35 de remerciement (cout:1.0) a hab55
>>> hab44 envoie courrier36 accuse de reception (cout:1.0) a hab55
courrier32 (cout:1.0)[bla bla] envoye par hab98 reçu par hab33
courrier33 URGENT (cout:2.0)[bla bla] envoye par hab14 reçu par hab53
>>> hab53 envoie courrier37 accuse de reception (cout:1.0) a hab14
```

```
Jour 6
courrier35 de remerciement (cout:1.0)[merci pour courrier30 de change] envoye par hab44 reçu par hab55
courrier36 accuse de reception (cout:1.0)[accuse du courrier30 de change recommande] envoye par hab44 reçu par hab55
courrier37 accuse de reception (cout:1.0)[accuse du courrier33 recommande] envoye par hab53 reçu par hab14
```