Programmation Orientée Objet
IG2I, LE2 – 2017-2018

CTP – 25 Janvier 2018 – Partie 2

Sarah Ben Othman, Diego Cattaruzza, Philippe Kubiak, Maxime Ogier

Conditions du déroulement de la partie 2

- **durée** : 3h20 ;
- ordinateurs personnels autorisés ;
- documents et pages internet autorisés ;
- accès à Moodle autorisé ;
- les moyens de communication (mails, messageries instantanées, Facebook, Tweeter, téléphones portables, Google Drive, etc.) sont INTERDITS ; le fait qu'un moyen de communication ne soit pas listé ci-dessus ne vous donne pas le droit de l'utiliser ;
- si vous utilisez du code que vous n'avez pas développé par vous-même lors du CTP, vous devez citer vos sources (dans les commentaires) ; ne pas citer ses sources est une fraude.

Rendu

Vous déposez sur Moodle, dans la zone de dépôt **Dépôt du CTP 2017/2018** une archive compressée de votre répertoire de projet. Cette archive doit être nommée **CTP_Nom_Prenom.zip** avec votre nom et votre prénom.

Un nommage différent donnera lieu à deux points de pénalisation. Cette règle est également valable pour toute proposition de nommage de ce CTP : projet, paquetages, classes, méthodes, attributs, ...

Vérifiez bien que les fichiers sources que vous avez réalisés lors du CTP sont présents dans le dossier que vous déposez sur Moodle (oui, vérifiez bien cela : toutes les années il y a des répertoires vides ou contenant les mauvais fichiers !) Il ne sera pas possible de renvoyer une nouvelle version de votre projet après la date limite de dépôt.

Notation

Cette seconde partie de l'épreuve de CTP est évaluée sur 16 points. La dernière partie est un bonus.

Pour chaque question, la moitié des points est attribuée si le code fonctionne correctement. **L'autre moitié des points est attribuée en fonction de la qualité du code** (respect des principes de la programmation orientée objet, efficacité des algorithmes) et de la présentation du code (indentation correcte, respects des conventions Java dans les noms de classe, attributs, méthodes et variables, commentaires au format Javadoc quand c'est pertinent).

Consignes

Le sujet est composé de 4 grandes parties liées entre elles. Vous devez les traiter dans l'ordre.

Vous êtes bien sûr autorisés à rajouter les attributs ou méthodes que vous jugez nécessaires même si ce n'est pas dit explicitement dans le sujet.

Attestation sur l'honneur

Je soussigné(e) ..... certifie que le projet déposé sur Moodle est issu d'un travail purement personnel. Je certifie également avoir cité toutes les sources de code externe dans les commentaires de mon code.

Fait à .....

Le .....

Signature .....

Sujet

Description de l'application

Le centre ville de Lille est très souvent encombré lors des livraisons matinales car les camions et camionnettes se garent n'importe comment. La ville de Lille souhaiterait bien être le fer de lance en logistique urbaine, et ainsi réglementer les livraisons des commerçants. Pour cela, elle fait appel à la start-up DeliverIG2I pour gérer toutes les livraisons en centre ville. Sur chaque rue, la ville propose des emplacements de livraison, et pour chaque emplacement utilisé par DeliverIG2I il y a un coût à payer. Il ne faut pas utiliser trop d'emplacements. Les commerçants seront livrés à partir de ces emplacements. DeliverIG2I doit donc décider :

- quels sont les emplacements à utiliser parmi les emplacements disponibles ;
- pour chaque emplacement utilisé, quels sont les commerçants desservis depuis cet emplacement (il ne faut pas qu'il y ait trop de commerçants affectés à un même emplacement ou que le commerçant soit trop loin de l'emplacement).

L'objectif est de payer le moins cher possible les emplacements et les frais de livraison, afin que DeliverIG2I s'assure une rentabilité maximale. Cette problématique est vraiment très importante pour la start-up DeliverIG2I car le choix des emplacements ne pourra plus changer pendant les 5 ans à venir.

Face à la complexité de cette problématique, les meilleurs élèves informaticiens de la région (voire du pays) sont appelés en renfort afin de fournir une application qui permette de définir les emplacements de livraison.

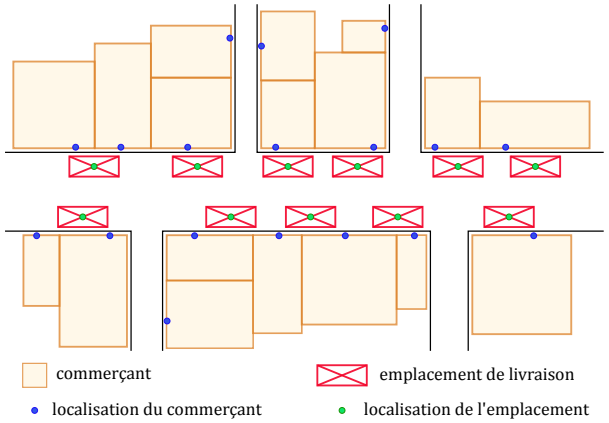

Figure 1 – Description du problème.

La Figure 1 présente une représentation graphique du problème à traiter. Autour d'une route, il y a un ensemble de commerçants. Sur cette route, plusieurs emplacements pourraient servir à la livraison. Chaque commerçant est représenté par une localisation (là où se situe la porte d'entrée). Chaque emplacement de livraison est également représenté par une localisation.

Modèle conceptuel de données

Après une première discussion avec DeliverIG2I, vous avez établi un premier modèle conceptuel de données présenté dans la Figure 2.

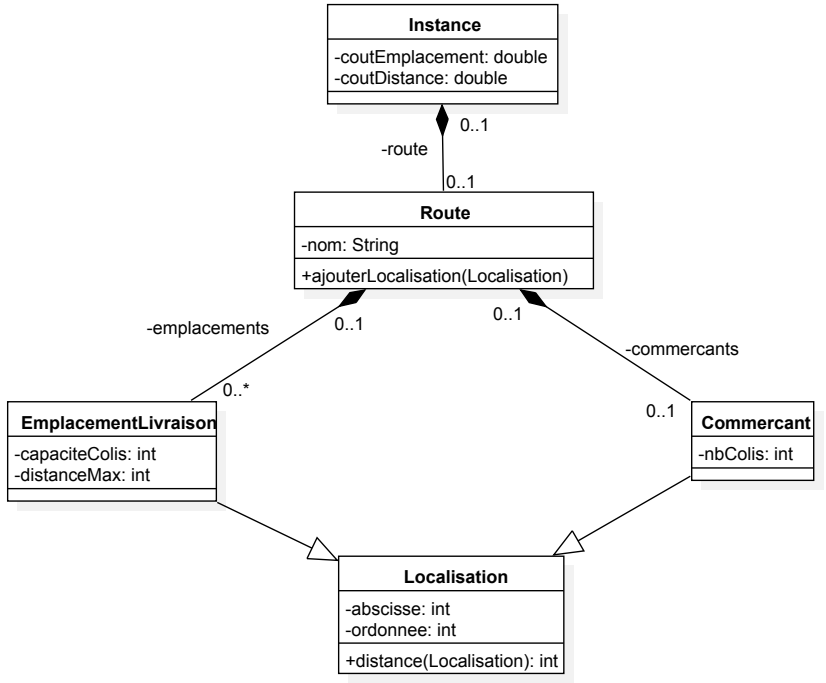

Figure 2 – Modèle conceptuel de données pour représenter une instance.

Une **Instance** représente les données d'entrée du problème. Elle est définie par un coût unitaire pour l'utilisation d'un emplacement (*coutEmplacement*) et un coût unitaire pour la distance à parcourir pour effectuer les livraisons (*coutDistance*). Elle possède une route.

Une **Route** est définie par un nom (*nom*). Elle est composée d'emplacements de livraison et de commerçants qu'il faut livrer.

Une **Localisation** est définie par une abscisse et une ordonnée (entières).

Un **EmplacementLivraison** est une localisation, et est défini par le nombre maximum de colis que l'on peut livrer depuis l'emplacement (*capaciteColis*) et la distance maximale que l'on peut parcourir pour livrer un commerçant (*distanceMax*).

Un **Commercant** est une localisation, et est défini par le nombre de colis qu'il faut lui livrer depuis un emplacement (*nbColis*).

1 Mise en place du modèle objet (5 points)

Dans un premier temps, nous nous occupons simplement de mettre en place le modèle objet qui correspond au diagramme de classe de la Figure 2.

1.1 Localisation (1,5 point)

Question 1. Créez un nouveau projet nommé **CTP_Nom_Prenom** avec votre nom et votre prénom. (Rappel : un nommage non conforme aux règles ⇒ 2 points de pénalisation). Ajoutez un paquetage **instance**. Dans ce paquetage, ajoutez une classe **Localisation**, avec un constructeur par données de l'abscisse et de l'ordonnée. Ajoutez également les accesseurs et mutateurs nécessaires. Deux localisations seront considérées égales si elles ont les mêmes coordonnées. Redéfinissez les méthodes **equals**, **hashCode** et **toString**. Faites un test pour vérifier que tout fonctionne correctement.

La distance entre deux localisations est définie comme la distance de Manhattan. C'est-à-dire que la distance $d(A, B)$ entre deux points $A(x_A, y_A)$ et $B(x_B, y_B)$ est : $d(A, B) = |x_A - x_B| + |y_A - y_B|$, où $|.|$ représente la valeur absolue. Un exemple est donné dans la Figure 3.

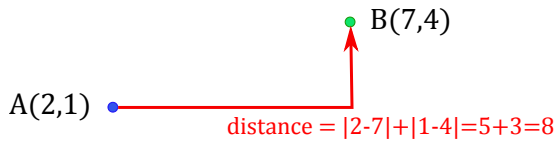

Figure 3 – Exemple de calcul de la distance de Manhattan entre deux points.

Question 2. Ajoutez dans la classe **Localisation** une méthode **public int distance(Localisation dest)** qui renvoie la distance de Manhattan entre les localisations **this** et **dest**. Vous pouvez utiliser la méthode statique **Math.abs()**. Testez que votre calcul de distance est correct.

1.2 Instance (3,5 points)

Question 3. Dans le paquetage **instance**, ajoutez les classes **EmplacementLivraison** et **Commercant** qui héritent de **Localisation**. Pour chacune de ces classes, ajoutez leurs attributs, implémentez un constructeur par données, et ajoutez les accesseurs et mutateurs nécessaires. Redéfinissez également la méthode **toString**. Faites un test pour vérifier que tout fonctionne correctement.

Question 4. Dans le paquetage **instance**, ajoutez une classe **Route**. Implémentez un constructeur par la donnée du nom de la route, et ajoutez les accesseurs et mutateurs nécessaires. De plus, ajoutez une méthode **public boolean ajouterLocalisation(Localisation loc)** qui permet d'ajouter une localisation à l'ensemble des emplacements ou l'ensemble des commerçants de la route, en fonction du type de localisation. Notez bien qu'il n'est pas possible qu'une route contienne deux emplacements ou deux commerçants situés aux mêmes coordonnées. Redéfinissez également la méthode **toString**. Faites un test pour vérifier que tout fonctionne correctement.

Question 5. Dans le paquetage **instance**, ajoutez une classe **Instance**. Ajoutez un constructeur par défaut (vous pourrez initialiser une route avec un nom quelconque), et les accesseurs et mutateurs nécessaires. Redéfinissez également la méthode **toString**. Ajoutez la méthode présente dans le fichier *methodesClasseInstance.txt* disponible sur Moodle. Si besoin, vous pouvez modifier ces méthodes. Faites un test pour vérifier que les données sont correctement chargées, par exemple :

```
Instance inst = new Instance();
inst.chargerData1();
System.out.println(inst);
```

2 Création d'une solution (10 points)

DeliverIG2I vous explique à présent qu'une solution est définie par :

- les emplacements de livraison utilisés ;
- les commerçants livrés depuis chaque emplacement utilisé.

Un exemple est donné dans la Figure 4.

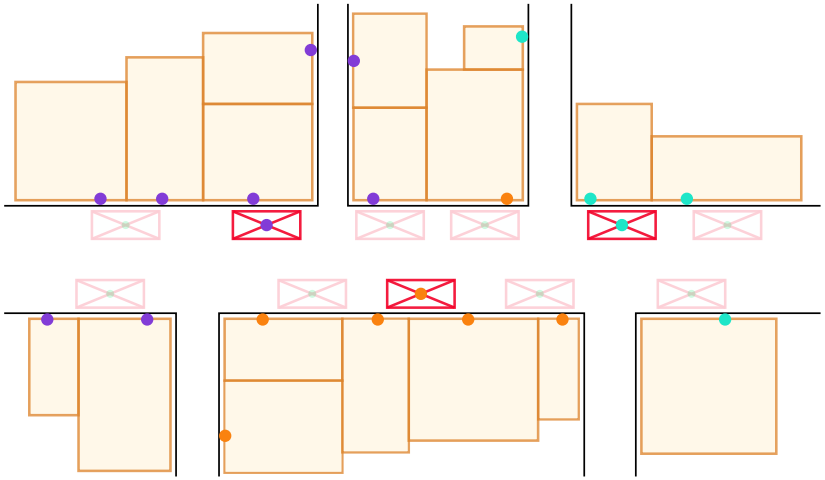

Figure 4 – Description d'une solution : on utilise 3 emplacements de livraison, et le code couleur permet de visualiser à quel emplacement de livraison est affecté chaque commerçant.

Le coût d'une solution est calculé de la manière suivante :

- c'est le coût des emplacements plus le coût de la distance totale de livraison ;
- le coût des emplacements est le nombre d'emplacements utilisés multiplié par le coût d'un emplacement (*coutEmplacement*) ;
- le coût de la distance totale de livraison est la distance totale multipliée par le coût unitaire (*coutDistance*) ;
- la distance totale est la somme des distances entre chaque commerçant et son emplacement.

Pour qu'une solution soit valide, il faut que :

- chaque commerçant soit affecté à un seul emplacement de livraison ;
- la distance entre un emplacement et le commerçant livré ne dépasse pas la distance maximale pour l'emplacement (*distanceMax*) ;
- le nombre total de colis livrés aux commerçants affectés à l'emplacement de livraison ne dépasse pas le nombre maximal de colis qu'on peut livrer depuis l'emplacement (*capaciteColis*).

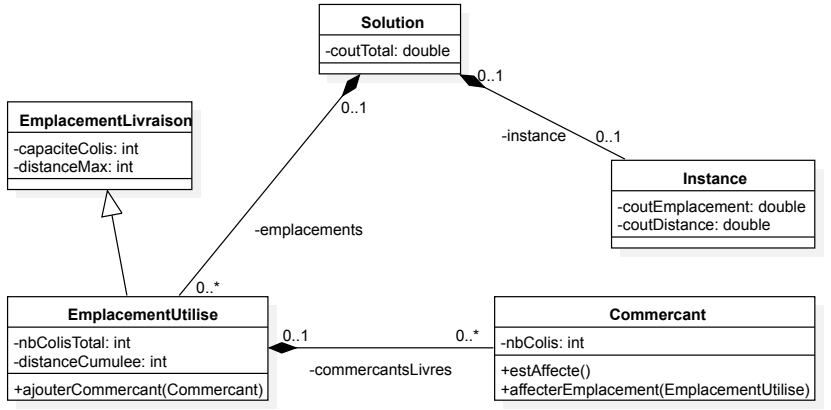

Figure 5 – Modèle conceptuel de données pour représenter une solution.

Après une deuxième discussion avec DeliverIG2I, vous avez établi un modèle conceptuel de données pour représenter une solution. Ce modèle est présenté dans la Figure 5.

Une **Solution** est définie par un coût total (comme expliqué précédemment), une instance et un ensemble d'emplacements utilisés. Un **EmplacementUtilise** est un emplacement de livraison qui est utilisé. Il est donc défini par le nombre total de colis livrés aux commerçants, la distance cumulée pour livrer les commerçants, et l'ensemble des commerçants livrés depuis l'emplacement.

2.1 Emplacements utilisés (5 points)

Question 6. Ajoutez un paquetage **solution**, et ajoutez-y une classe **EmplacementUtilise** qui hérite de **EmplacementLivraison**. Ajoutez un constructeur par copie d'un objet de type **EmplacementLivraison**, et les accesseurs et mutateurs nécessaires. Redéfinissez la méthode **toString**.

Question 7. Dans la classe **EmplacementUtilise**, ajoutez une méthode **public boolean ajouterCommercant(Commercant comm)** qui essaye d'ajouter un commerçant à l'emplacement. Si l'ajout a bien lieu, on renvoie **true**, et on met à jour les attributs. Sinon, on renvoie **false**. Testez que tout fonctionne correctement en vous aidant, par exemple, du code disponible dans le fichier *testEmplacementUtilise.txt* sur Moodle.

Question 8. On souhaite à présent que chaque commerçant connaisse l'emplacement auquel il est affecté. Modifiez la classe **Commercant** de la manière suivante :

- ajoutez un attribut de type **EmplacementUtilise** ;
- ajoutez une méthode **public boolean estAffecte()** qui renvoie **true** si le commerçant est déjà affecté à un emplacement de livraison, **false** sinon ;
- ajoutez une méthode **public boolean affecterEmplacement(EmplacementUtilise empl)** qui essaye d'affecter l'emplacement au commerçant ; cette méthode renvoie **true** si l'affectation a eu lieu, **false** sinon (par exemple si le commerçant était déjà affecté à un emplacement). Remarque : on ne vérifiera pas que l'emplacement contient bien le commerçant.

Question 9. Mettez à jour la méthode **ajouterCommercant** de la classe **EmplacementUtilise** de telle sorte que lorsqu'un commerçant est ajouté à l'emplacement, alors l'emplacement est également ajouté au commerçant. Et si l'emplacement ne peut pas être ajouté au commerçant, alors le commerçant n'est pas ajouté à l'emplacement. Testez que tout fonctionne correctement : un commerçant ne peut pas être affecté dans deux emplacements.

2.2 Solution et calcul de coût (1 point)

Question 10. Dans le paquetage **solution**, ajoutez une classe **Solution** définie par son coût total, une instance et une collection d'emplacements utilisés. Ajoutez-y un constructeur par la donnée d'une instance, les accesseurs et mutateurs nécessaires et redéfinissez la méthode **toString**.

Question 11. Dans la classe **Solution**, ajoutez une méthode **public void calculerCout()** qui fait le calcul du coût total de la solution, et met à jour l'attribut correspondant.

2.3 Première solution (4 points)

Nous souhaitons à présent implémenter un algorithme simple afin d'obtenir de premières solutions. L'idée est la suivante : tant qu'il reste des commerçants non affectés, on prend un nouvel emplacement de livraison, et on affecte des commerçants (pas déjà affectés) à cet emplacement. Pour affecter les commerçants à un emplacement de livraison, on parcoure les commerçants, et on y ajoute les commerçants non affectés si possible. À la fin, on calcule le coût total de la solution.

Question 12. Dans un premier temps, il nous faut récupérer certaines données de l'instance : les commerçants et les emplacements de livraison. Ainsi, ajoutez dans la classe **Instance** deux méthodes :

- **public Collection<Commercant> getTousCommercants()** ;
- **public Collection<EmplacementLivraison> getTousEmplacements()**.

Indications : (1) il est préférable de ne pas retourner les références mais de faire des copies, (2) n'oubliez pas que vous pouvez rajouter les méthodes que vous souhaitez dans d'autres classes (la classe **Route** par exemple).

Question 13. À présent, ajoutez dans la classe **Solution** une méthode **public void solutionSimple()** qui permette de construire une solution simple (l'algorithme présenté précédemment par exemple), et de mettre à jour l'attribut correspondant au coût de cette solution. Testez pour vérifier que tout fonctionne correctement.
Remarque : rappelez-vous que vous avez le droit d'ajouter toutes les méthodes que vous souhaitez, et qu'un code propre devrait comporter des méthodes courtes (en nombre de lignes).

3 Écriture dans un fichier (1 point)

À présent, nous souhaitons pouvoir écrire la solution dans un fichier, selon le format suivant :

- la première ligne contient le coût total ;
- la deuxième ligne contient le nombre d'emplacements utilisés ;
- les lignes suivantes contiennent une description des emplacements utilisés ;
- pour chaque emplacement utilisé, on affiche sur une seule ligne ses coordonnées, son nombre de colis livrées ; puis sur les lignes suivantes on affiche les coordonnées des commerçants livrés depuis l'emplacement.

Voici un exemple de fichier de sortie :

```
Cout total = 3026.0
Nb emplacements utilises = 3
Emplacement : (6 ; 0) — nb colis total = 25
    Commercant : (12 ; 5)
    Commercant : (5 ; 2)
Emplacement : (4 ; 0) — nb colis total = 20
    Commercant : (7 ; 3)
Emplacement : (9 ; 0) — nb colis total = 15
    Commercant : (8 ; 5)
```

Question 14. Dans la classe **Solution**, ajoutez une méthode **public void ecrireSolution(String nomFichier)** qui écrit la solution dans le format proposé dans un fichier (dont le nom est passé en paramètre).

4 Amélioration de la solution (Points bonus)

Question 15. Dans la classe **Solution**, ajoutez une méthode **public void solutionOptimisee()** qui propose une meilleure solution que celle de l'algorithme simple. Pour cela, voici quelques pistes :

- ajouter un peu d'aléatoire et prendre la meilleure solution ;
- faire un tri sur les emplacements de livraison avant de choisir celui qui va être utilisé ;
- trier les commerçants avant de les affecter à un emplacement utilisé.