Projet noté

(encadré pendant 1 TD et 3 TP, travail personnel nécessaire)

Règles strictes :

- Projet à réaliser en **binôme** (1 binôme = 2 personnes) ;
- Rapport et programmes Java à rendre au plus tard le **21 décembre 2018** sous la forme d'un (et un seul) fichier compressé au format **zip** ou **tar.gz** ;
- Ce fichier doit être déposé sur madoc par **un** (et un seul) membre du binôme dans l'espace devoir correspondant à son groupe de TP.

Sujet

Une petite entreprise de distribution de repas en milieu urbain demande à réaliser un logiciel pour répondre à ses besoins. D'une part, elle souhaite acquérir des véhicules, et dans un premier temps des vélos et des scooters.

- Un véhicule est caractérisé par les informations suivantes : un nom, un prix d'achat en euros, une vitesse moyenne en kilomètres par heure, un taux d'émission de CO2 en grammes par kilomètre, une charge utile en kilogrammes et un coût d'utilisation sur un kilomètre en euros par kilomètre.
- Un vélo a une vitesse moyenne de 15 kilomètres par heure (en milieu urbain). Son taux d'émission de CO2 est nul. Sa charge utile est nulle. Son coût d'utilisation sur un kilomètre est égal au prix d'achat divisé par 30000.
- Un scooter a une charge utile en kilogrammes, une cylindrée en centimètres cube et une consommation moyenne d'essence en litres par 100 kilomètres. Sa vitesse moyenne est de 30 kilomètres par heure plus sa cylindrée divisée par 50. Son taux d'émission de CO2 est égal à sa cylindrée divisée par 4. Son coût d'utilisation sur un kilomètre est égal au prix d'achat divisé par 20000 plus le coût de la consommation d'essence (en fonction du prix d'un litre d'essence qui sera représenté par une variable dont la valeur est fixée, par exemple $1,45$ euros).

D'autre part, l'entreprise souhaite embaucher des salariés. Un salarié possède un nom, un poids en kilogrammes, un salaire en euros par heure. Sa charge utile est égale à son poids divisé par 8.

Cette entreprise a vocation à réaliser des courses. Une course consiste à faire transporter un poids sur une distance (entre le local de l'entreprise et le lieu de livraison) par un salarié conduisant un véhicule. La charge utile d'une course est le maximum entre celle du véhicule et celle du conducteur. Une course est possible si la charge utile est supérieure ou égale au poids à transporter et si le temps de parcours est d'au plus une heure. Sa quantité de CO2 émis est calculée en fonction du taux d'émission de CO2 du véhicule et de la distance à parcourir. Le prix d'une course est calculée comme une somme entre le salaire du conducteur et le coût d'utilisation du véhicule sur la distance à parcourir. Il faudra penser à compter l'aller et le retour dans les différents calculs effectués.

On peut comparer les courses entre elles comme suit. Soit une course (c_1, p_1) avec une quantité de CO2 c_1 et un prix p_1 et soit une autre course (c_2, p_2). On dit que (c_1, p_1) est pire que (c_2, p_2) si la condition suivante est vraie :

$$((c_1 > c_2) \wedge (p_1 > p_2)) \ \vee \ ((c_1 > c_2) \wedge (p_1 = p_2)) \vee \ ((c_1 = c_2) \wedge (p_1 > p_2))$$

Maintenant, considérons un repas à transporter donné par son poids et la distance à parcourir. Le gérant de l'entreprise souhaite connaître les meilleures courses pour pouvoir choisir le véhicule et le conducteur : d'une part, il veut connaître toutes les courses possibles (si aucune course n'est possible, il peut donc refuser de livrer le repas) ; ensuite, parmi ces courses possibles, il veut toutes les courses qui ne sont pas pires qu'une autre course, appelons-les les courses optimales (il veut prendre la meilleure décision) ; puis, parmi les courses optimales, il veut celles qui ont une quantité minimale de CO2 émis (il a la fibre écologique).

Pour tester les programmes, vous pourrez considérer les données (salariés et véhicules) ci-dessous et le transport d'un repas de 5kg sur 3km.

Nom	Poids (kg)	Salaire (EUR/h)
Louise	55	7.4
Alfred	80	7.2
Camille	67	7.5
Pierre	62	7.2

Nom	Prix (EUR)
Velov	450
Bicloo	500

Nom	Prix (EUR)	Cylindrée (cm^3)	Charge utile (kg)	Conso. Essence (l/100km)
Yamama	3500	300	50	7.5
Vespo	2500	125	30	5.5
Piagi	2000	150	35	6

Le travail consiste donc à programmer toutes les classes nécessaires à la réalisation de ces tâches.

1. Dans un premier temps, le binôme doit chercher à identifier les classes, définir les relations entre ces classes et représenter les objets en mémoire, bien avant de commencer à programmer. Ce travail sera abordé dans la séance de TD encadrée avec chaque binôme (former les binômes avant la séance de TD et se mettre à côté pendant la séance).
2. Dans l'activité de programmation, vous pourrez commencer à développer les classes les plus élémentaires. Dans chaque classe, il est conseillé d'implémenter et de tester les méthodes une par une. Ne pas attendre de programmer toute une classe avant de tester les méthodes.
3. Vous constaterez que votre solution est extensible si l'ajout d'un nouveau type de véhicule se fait sans modifier le code existant. Il faudra avoir un œil critique sur ce point dans le rapport.

Rapport et programmes Java à rendre

Les programmes Java seront dûment commentés. Le rapport (se voulant complet et succinct) comprendra les parties suivantes :

1. une page de titre : noms, prénoms, groupe de TP, formation, logo de l'université, année, titre ou sujet ;
2. une introduction rappelant le sujet ;
3. la modélisation présentée sous la forme d'un diagramme de classe en justifiant clairement les relations entre les classes (par exemple : il existe une hiérarchie de [*blablabla*] avec une super-classe [*blablabla*] et des sous-classes [*blablabla*] sachant que [*blablabla*] ; un objet B se comporte comme un objet A avec en plus [*blablabla*] donc B hérite de A ; un objet B se compose de plusieurs objets A car [*blablabla*]) ;
4. la spécification de chaque classe avec son rôle et un tableau à deux colonnes donnant pour chaque méthode publique sa signature et une description brève, par exemple :

   Classe : `UneClasse`

   Rôle : la classe `UneClasse` permet de créer des objets [*blablabla*] et de réaliser les actions [*blablabla*]

   Méthodes :

Signature	Description
`UneClasse()`	construit une instance telle que [*blablabla*]
`void action(int n)`	réalise `n` fois [*blablabla*]
`int f()`	calcule [*blablabla*] et retourne [*blablabla*]

5. les résultats obtenus sur les jeux de données introduits ci-avant pour aider le gérant à prendre une décision ;
6. une conclusion avec un regard critique sur votre projet en montrant que votre solution est extensible (ou pas), en identifiant clairement les réussites et les échecs (par exemple : l'algorithme permettant de [*blablabla*] est performant car [*blablabla*] ; l'implémentation de la classe [*blablabla*] n'a pas été faite car [*blablabla*]) et en indiquant si le principe d'encapsulation est respecté.

Enfin, il est fortement recommandé de ne pas écrire à la première personne (par exemple : j'ai fait [*blablabla*] ou nous avons créé [*blablabla*]) et de ne pas décrire la réalisation du projet de manière chronologique (par exemple : dans un premier temps nous avons fait [*blablabla*] puis comme [*blablabla*] nous avons fait autrement [*blablabla*] et enfin [*blablabla*]). Soyez clairs, précis et concis dans vos explications. Utilisez un correcteur orthographique. Faites relire votre rapport par des camarades ou des proches.

Compléments de programmation

Soit la déclaration suivante :

```
final int var = 5;
```

Le mot-clé `final` indique que la valeur de `var` ne pourra plus changer par la suite. C'est donc un moyen de définir une constante.

Une classe peut être également `final`. Cela indique qu'on ne pourra pas en faire des sous-classes.

Dans une classe, il est possible de définir une variable qui n'est pas un attribut des objets ; on parle de variable de classe. Pour cela, il faut utiliser le mot-clé `static`, par exemple :

```
class Param {
   static int PrixEssence = 1.45;
```

La variable `Param.PrixEssence` est une variable déclarée dans la portée de la classe `Param`. Dans le projet, on pourra représenter ainsi les valeurs constantes apparaissant dans l'énoncé.

Il est possible de combiner les mot-clés `final` et `static`. Par exemple, dans la clase standard `Math` on trouve la valeur approchée de π :

```
public class Math {
   public static final double PI = 3,14159265358979323846;
```

Dans une classe, on peut définir également une fonction qui n'est pas une méthode des objets ; on parle de fonction de classe. Par exemple, dans la clase standard `Math` on trouve la fonction de calcul de la racine carrée d'un réel :

```
public class Math {
   public static double sqrt(double x) { ... }
```

Remarquons que nous avons déjà utilisé `Math.sqrt`.

Dans le projet, il peut être utile de décomposer la fonction `main` en plusieurs fonctions, par exemple :

```
public class Projet {
   public static void main(String[] args) {
      PremiereAction();
      DeuxiemeAction();
      ...
   }

  public static void PremiereAction() { ... }
  public static void DeuxiemeAction() { ... }
}
```

On comprend enfin ce que signifie le mot-clé `static` associé à la fonction `main`. Ce n'est pas une méthode, simplement une fonction dans la portée de la classe `Projet`. En `Java`, toutes les définitions se font au sein d'une classe.