IFT1169 – TRAVAIL PRATIQUE #3 – 1 décembre 2018

« Un voyage durant les fêtes »

Mohamed Lokbani

Équipes: Le travail est à faire en monôme ou en binôme, pas plus de deux. Vous ne remettez qu'un seul travail par équipe.
Remise : Une seule remise à effectuer par voie électronique le jeudi **20 décembre 2018, 23h59 au plus tard, sans possibilités de prorogation**.
Conseils: n'attendez pas le dernier jour avant la remise pour engager votre travail. Vous n'aurez pas le temps nécessaire pour le réaliser.

But : ce TP a pour but de vous faire pratiquer les différents paradigmes de la programmation orientée objet. Nous allons examiner en particulier les notions d'héritage. Ce travail va vous permettre aussi de vous familiariser avec la syntaxe de base d'un fichier « makefile » et vous introduire à des notions élémentaires du langage de modélisation objet unifié (UML).

Énoncé : pour certains, la période des fêtes est synonyme de voyages vers d'autres horizons. Parfois, pour voyager d'un endroit A vers un endroit B, il est nécessaire de prendre un ou plusieurs moyens de transport dépendant de la destination finale. Chaque voyage a un coût donné en fonction des moyens de transport utilisés. Dans la réalisation de ce travail pratique, vous allez prêter une attention particulière à :
- la distance à parcourir durant votre voyage,
- le temps nécessaire pour parcourir une des étapes de votre voyage,
- et finalement les coûts associés aux moyens de transport utilisés durant votre voyage.

Description : pour ce travail pratique, vous allez coder les 4 classes décrites sous la forme « UML » dans la figure -1. Ces classes vont avoir aussi la tâche de garder une trace de votre éventuel voyage. Ce voyage peut-être effectué à l'aide de plusieurs moyens de transport. Par exemple, vous pouvez décider de prendre votre propre voiture pour le trajet [Montréal → Québec], une voiture de location pour couvrir le trajet [Québec→ Halifax] et finalement l'avion pour le trajet [Halifax → Montréal].
Par ailleurs, nous allons supposer certaines hypothèses dont certaines sont loin d'être réelles!
- Le seul coût associé à la conduite d'une voiture est celui associé au prix total du carburant utilisé durant ce voyage.
- Le prix de l'essence reste le même durant tout le voyage. Il est donc indépendant de l'endroit où vous êtes.
- La vitesse moyenne d'une voiture (la vôtre ou celle de location) est de 80 km/h.
- Chaque voiture consomme en moyenne 5 litres au 100 km.
- Chaque compagnie de location a le même système de tarification. Elle va charger un tarif de base, plus un prix additionnel pour chaque kilomètre parcouru. Ainsi donc, pour un tarif de base de 200$ et pour un prix au kilomètre de l'ordre de 0.15 dollar, le coût de la location associé à 100 kilomètres parcourus sera donc : 200 + (0.15x100) = 215$.
- Durant votre séjour dans une ville donnée, vous n'allez pas tenir compte de la durée de ce séjour ni des coûts associés à la nourriture et l'hébergement dans un hôtel.

Description des classes :
Une instance de « TransBase » représente une étape de votre voyage: une conduite en voiture, un voyage en avion. Alors qu'une instance de « Voyage » ne représente que l'ensemble des étapes (une ou plusieurs) de votre voyage.

« **TransBase** » : Une instance de cette classe représente le moyen de transport de base utilisé durant cette étape du voyage. Ce moyen de transport de base est représenté par « votre propre voiture ».

Parmi ses membres, on peut citer « la ville de départ », « la ville d'arrivée », « la distance entre les deux villes en kilomètres ». Cette classe contient aussi la variable « dollarsParLitre » qui représente le prix courant du carburant (en dollars par litre). Par défaut, le prix est de « 1.25$ » par litre.
Par ailleurs, la classe contient une série de méthodes du type assesseur c.-à-d. « get » pour donner des informations relatives à certains champs de la classe et du type modificateur c.-à-d. « set » pour modifier les valeurs de certains champs de la classe.
Par ailleurs, la classe contient aussi d'autres méthodes :
« affiche » : retourne un « string » contenant les deux villes de cette étape et la distance qui les sépare : « Montréal à Québec (270.0 km) ». Cette méthode est appelée de manière indirecte par l'opérateur de sortie (la méthode operator<<).
« getTemps » : calcule la durée du voyage à partir de la distance à parcourir et la vitesse moyenne.
« getCout » : calcule le coût de cette étape qui n'est autre que le coût du carburant utilisé, calculé à partir de la distance parcourue, la consommation moyenne de litres par kilomètres et le prix du carburant.

« **TransVdL** » : Une instance de cette classe correspond à une étape du voyage qui utilise comme moyen de transport la voiture de location. Cette classe dérive de la classe « TransBase ». Elle contient les membres supplémentaires suivants :
« PrixBase » : le prix de base fixé par la compagnie de location (en dollars).
« PrixParKm » : le prix par kilomètre (en dollars par kilomètre).
De même que la classe « TransBase », la classe « TransVdL » a ses propres assesseurs et modificateurs, en plus des méthodes suivantes :
« affiche » : retourne un « string » contenant tous les détails de cette étape du voyage : « Montréal à Québec (270.0 km, prix de base $50.0, prix par kilomètre $0.1) ». Cette méthode est appelée de manière indirecte par l'opérateur de sortie (la méthode operator<<).
« getCout » : retourne le coût total de cette étape en tenant compte du prix de base de la location, du coût associé aux kilomètres parcourus et le prix de l'essence.

« **TransAvion** » : Une instance de cette classe correspond à une étape du voyage en utilisant comme moyen de transport l'avion. Cette classe dérive elle aussi de la classe « TransBase ». Elle contient les membres supplémentaires suivants :
« Tarif » : le prix du billet d'avion (en dollars).
« Durée » : la durée du voyage (en heures).
De même que la classe « TransBase », la classe « TransAvion » a ses propres assesseurs et modificateurs, en plus des méthodes suivantes :
« affiche » : retourne un « string » contenant tous les détails de cette étape du voyage : « Montréal à Québec (270.0 km, Tarif $200.0, durée 1.2 heure) ». Cette méthode est appelée de manière indirecte par l'opérateur de sortie (la méthode operator<<).
« getCout » : retourne le coût total de cette étape qui n'est autre que le prix du billet d'avion.
« getTemps » : retourne le temps nécessaire pour effectuer cette étape qui n'est autre que la durée du vol.

« **Voyage** » : Un voyage a un nom. Il peut être parcouru sur une ou plusieurs étapes. À chaque étape, est associé un des moyens de transport précédemment définis : « TransBase » ou « TransVdL» ou « TransAvion ». Vous pouvez utiliser la structure de données de votre choix (autres que celles définies par la STL) pour stocker ces moyens de transport. Vous pouvez imposer un maximum de 20 étapes par voyage.
De même que les classes précédemment décrites, la classe « Voyage » a aussi ses propres assesseurs et modificateurs. Son constructeur permet de construire un voyage avec 0 étape (pour l'instant).
Parmi les autres méthodes :
« ajoutEtape » : ajouter une nouvelle étape au voyage.
« getNbreEtapes » : retourne le nombre d'étapes présentement dans ce voyage.
« getEtape » : retourne une étape donnée du voyage, désignée par un index.
« getCout » : retourne le coût total du voyage (la somme des coûts de toutes les étapes).

« getTemps » : retourne la durée totale du voyage (la somme de toutes les durées de toutes les étapes).
« getDistance » : retourne la distance totale parcourue (la somme de toutes les distances parcourues de toutes les étapes).
« estAllerRetour » : retourne vrai (« true ») si le voyage se termine sur la ville de départ.
« VerifiesNoms » : retourne vrai (« true ») si le nom de la ville pour cette étape est correct (la ville de départ d'une étape est la ville d'arrivée de l'étape qui a précédé).
« operator<< » : retourne un « string » qui inclut le nom du voyage en plus des villes visitées, par exemple « Vacances [Drummondville, Rivière-du-Loup, Rimouski, Gaspé] ». La chaîne affichée ne doit pas contenir plus de détails.

Description des fichiers fournis :
Le fichier « tp3A18.cpp » contient la fonction « main » permettant de tester les différentes fonctionnalités des classes que vous allez développer dans ce travail pratique. En aucun cas, vous ne devez modifier ce fichier.
Le fichier « sortietest.txt » est la sortie obtenue suite à l'exécution du programme « tp3A18.exe ». Vous devez obtenir exactement la même sortie que celle fournie dans le fichier « sortieA18tp3.txt ».

Les choses à faire :

Description	Fichiers associés	
Voyage	Voyage.h	Voyage.cpp
TransBase	TransBase.h	TransBase.cpp
TransVdL	TransVdL.h	TransVdL.cpp
TransAvion	TransAvion.h	TransAvion.cpp
Rapport	rapportA18.pdf	
Makefile	makefile	

Nous avons associé deux fichiers pour chaque classe. Le fichier d'en-tête ayant l'extension « .h » va contenir la définition de la classe, ses structures (s'il y en a), les déclarations de fonctions (s'il y en a), les « #define » ainsi que les constantes. Alors que le fichier « .cpp » va contenir l'implémentation de la classe (le code complet de la classe).
Le fait d'avoir découpé le programme en plusieurs fichiers ne va pas vous faciliter les opérations de compilation, d'édition de liens (linkage), d'exécution et de test, si vous êtes ramenés à écrire sur la ligne de commande les instructions permettant ces différentes opérations. Une façon de vous faciliter la vie, c'est de regrouper ces instructions dans un fichier de script. Ce fichier de script, peut-être appelé « makefile », s'il respecte une certaine syntaxe. Pour ce travail pratique, on vous demandera d'écrire un fichier « makefile » permettant de compiler l'ensemble des fichiers afin de produire un exécutable. Il doit permettre aussi d'effacer tous les fichiers objets ainsi que l'exécutable, afin de recommencer la compilation à zéro.

Hypothèses et contraintes :
- Vous ne devez pas utiliser la directive globale « using namespace std ». Comme dans le cas du fichier fourni « tp3A18.cpp », nous n'avons pas fait appel « using namespace std ». Nous avons utilisé à la place un appel indirect « std :: ».
- Toutes les autres notions du C++ sont autorisées.
- Nous allons comparer de manière automatique (en utilisant des scriptes) la sortie obtenue par votre programme et la nôtre. De ce fait: vous devez suivre les spécifications à la lettre. Vous devez obtenir exactement les mêmes sorties que nous. Pour ce travail, le format d'affichage est très basic. Nous n'avons pas imposé un format fixe. Vous devez juste faire les opérations dans l'ordre et afficher les résultats simplement avec les séparateurs demandés.

Voir le TP#1 pour les commandes utilisées afin de comparer deux fichiers.

- Il faudra vous assurer de respecter les noms de fichiers. Vous devez respecter aussi le format de l'affichage en sortie.

Fichiers à remettre : vous devez remettre le fichier compressé « tp3A18.zip » par l'entremise du système de remise de Studium. Par ailleurs, vous devez inclure dans votre remise un rapport décrivant le travail effectué.

Remise : vérifiez que le code que vous avez produit (devant normalement fonctionner correctement chez vous) fonctionne aussi bien sur les ordinateurs de la DESI.

Barème : Ce TP3 est noté sur 8 points.

Conception des classes : 5 points

Classes	Nombre de points
Voyage.h/.cpp	2
TransBase.h/.cpp	1
TransVdL.h/.cpp	1
TransAvion.h/.cpp	1

Avis global : 2 points
Programmation, clarté du code, etc.

Rapport : 1 point
On ne vous demande pas de nous dire à quoi sert la variable « i », la boucle « for » ou la boucle « while ». Vos commentaires dans le code devraient nous expliquer leurs utilités. On vous demande de vous mettre à la place d'une personne qui connaît l'objectif de votre programme sans connaître votre code. En lisant votre rapport, elle devra pouvoir naviguer avec aisance dans votre programme. Elle saura par exemple comment la programmation orientée objet a été introduite et pour quelle raison; l'utilité des différentes fonctionnalités, etc.

Makefile : -1 point, s'il est manquant

Tests : -4 points, si les résultats obtenus ne coïncident pas avec le fichier fourni

En plus du précédent barème, vous risquez de perdre des points dans les cas suivants ….

- La non-remise électronique (volontaire ou par erreur) est sanctionnée par la note 0.
- Un programme qui ne compile pas : 0.
- Un programme qui compile, mais ne réalise pas les choses prévues dans la spécification : 0.
- Les avertissements (warnings) non corrigés : cela dépend de la quantité! À partir de -0.25 et plus.
- Aberration dans le codage : même si tous les chemins mènent à Rome, faites l'effort nécessaire pour éviter de prendre le plus long!

Documents fournis
- Les résultats obtenus
- Le fichier « tp03A18.cpp »
- Feuille de route.

Communications
- Par courriel : une seule adresse « dift1169@iro.umontreal.ca ».

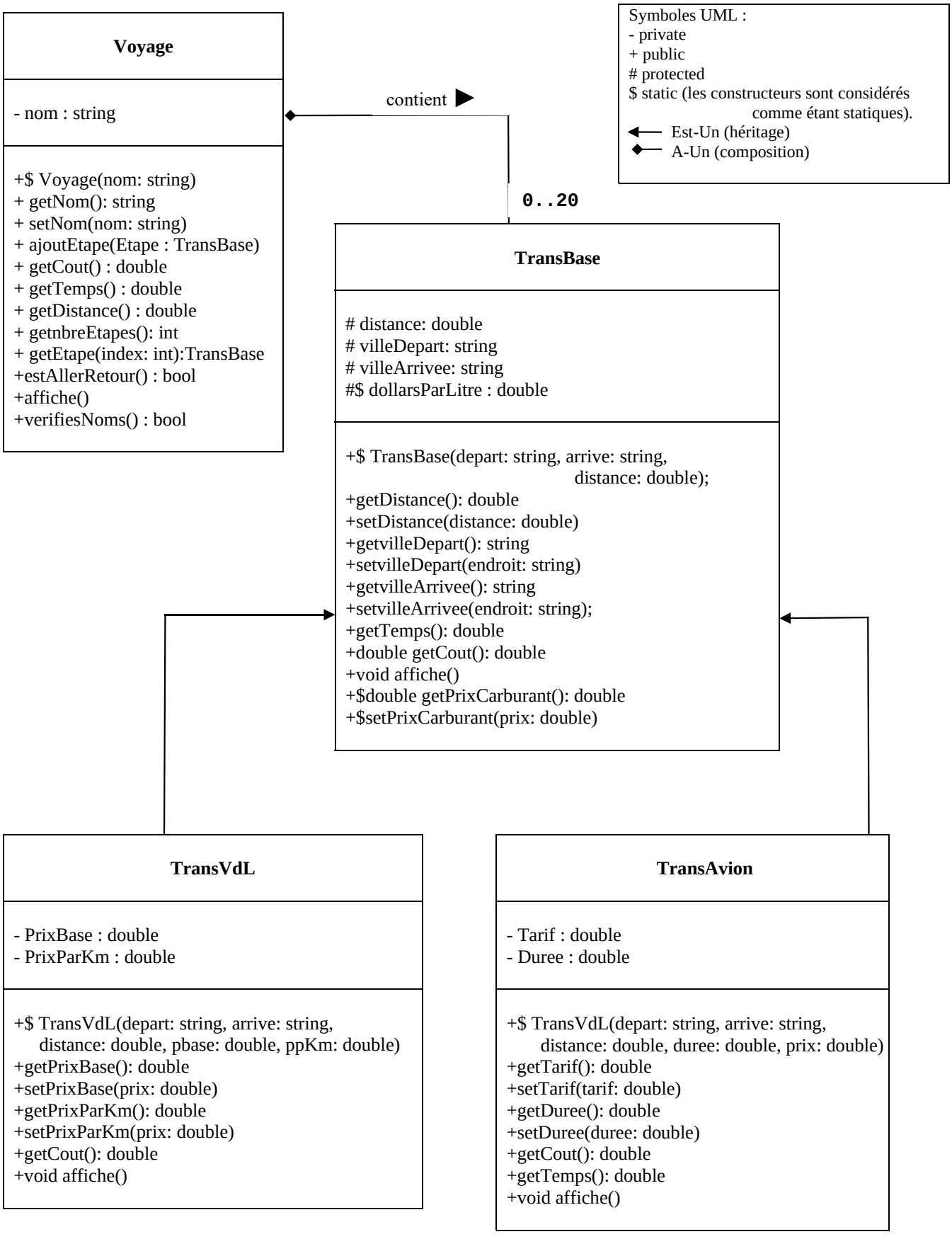
Pour faciliter le traitement de votre requête, inclure dans le sujet de votre courriel, au moins la chaîne [IFT1169] et une référence au tp03.

- À travers Studium : forum TP#3.

- En personne : à la démo ou sur rendez-vous.

Mise à jour

1-12-2018 diffusion de l'énoncé.


Voyage
- nom : string
+$ Voyage(nom: string)
+ getNom(): string
+ setNom(nom: string)
+ ajoutEtape(Etape : TransBase)
+ getCout() : double
+ getTemps() : double
+ getDistance() : double
+ getnbreEtapes(): int
+ getEtape(index: int):TransBase
+estAllerRetour() : bool
+affiche()
+verifiesNoms() : bool
contient
0..20
Symboles UML :
- private
+ public
# protected
$ static (les constructeurs sont considérés comme étant statiques).
Est-Un (héritage)
A-Un (composition)
TransBase
distance: double
villeDepart: string
villeArrivee: string
#$ dollarsParLitre : double
+$ TransBase(depart: string, arrive: string, distance: double);
+getDistance(): double
+setDistance(distance: double)
+getvilleDepart(): string
+setvilleDepart(endroit: string)
+getvilleArrivee(): string
+setvilleArrivee(endroit: string);
+getTemps(): double
+double getCout(): double
+void affiche()
+$double getPrixCarburant(): double
+$setPrixCarburant(prix: double)
TransVdL
- PrixBase : double
- PrixParKm : double
+$ TransVdL(depart: string, arrive: string, distance: double, pbase: double, ppKm: double)
+getPrixBase(): double
+setPrixBase(prix: double)
+getPrixParKm(): double
+setPrixParKm(prix: double)
+getCout(): double
+void affiche()
TransAvion
- Tarif : double
- Duree : double
+$ TransVdL(depart: string, arrive: string, distance: double, duree: double, prix: double)
+getTarif(): double
+setTarif(tarif: double)
+getDuree(): double
+setDuree(duree: double)
+getCout(): double
+getTemps(): double
+void affiche()