IFT1169 – TRAVAIL PRATIQUE #2 – 14 mai 2006

« Périple estival au menu »

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 est à effectuer : électronique le jeudi **03 juin 2006, 23h59 au plus tard, sans possibilités de retard**.
Conseils: N'attendez pas le dernier jour avant la remise pour commencer, vous n'aurez pas le temps nécessaire pour le faire.

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, l'été est synonyme de voyages vers des horizons meilleurs. 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 transports. 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 votre 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 à100 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$ » par litre.

Par ailleurs, la classe contient une série de méthodes du type assesseur i.e. « get » pour donner des informations relatives à certains champs de la classe et du type modificateur i.e. « 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) ».
« getTemps » : calcule la durée du voyage à partir de la distance à parcourir et la vitesse moyenne.
« getCoût » : 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 km $0.1) ».
« getCoût » : 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 heures) ».
« getCoût » : 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 transports. 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.
« getCoût » : 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).
« affiche » : 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 « tp2E06.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 « tp2E06.exe ». Vous devez obtenir exactement la même sortie que celle fournie dans le fichier « sortietest.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	rapportE06.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 » à base de règles très simplistes : une seule cible par défaut, une cible locale associée à chaque fichier objet potentiel de votre projet et une cible globale associée à votre programme exécutable. Des exemples de création d'un tel fichier sont disponibles à partir de ce lien « http://gl.developpez.com/tutoriel/outil/makefile/ »

Hypothèses et contraintes :

- Pour commencer, vous devez déjà respecter les différentes contraintes de programmation précédemment décrites.
- C'est un travail à faire à **deux** ! Et deux ne signifie pas quatre ni dix. Plagiat équivaut à un 0 pour commencer.
- Se mettre à deux, signifie aussi relever le niveau de la réflexion dans le cadre de l'enrichissement de vos connaissances.
- IFT1169 est un cours avancé en C++, donc votre programme doit être écrit en C++ et pas en C ni en Java! Nous n'autoriserons aucune référence au langage C. Par exemple l'instruction [#include <stdio.h>] fait référence au langage C, donc elle n'est pas autorisée. Utiliser plutôt [#include <iostream>].
- Ce travail n'est pas un exercice d'algorithmique, donc pas besoin de compliquer le travail pour rien!
- Il faudra vous assurer de respecter les noms de fichiers. Vous devez respecter aussi le format de l'affichage en sortie.

Remise : Il est important de noter que votre TP sera compilé avec gcc3.2.4. Si par choix vous décidez d'utiliser un autre compilateur, vérifiez que le code que vous avez produit (qui normalement fonctionne correctement chez vous) fonctionne bien sur les ordinateurs de la DESI. Pour avoir la version du compilateur, utilisez la commande "**gcc –v**", qui devra donner le numéro de version "**3.2.4**".
Par ailleurs, assurez-vous de la présence de l'option « pedantic » sur la ligne de compilation. (Cette option n'est pas activée par défaut dans l'utilitaire « devcpp ». Il faudra donc penser à l'activer).
Vous devez remettre un seul fichier : « tp2E06.zip ». Le fichier « tp2E06.zip » va contenir l'ensemble de vos fichiers « C++ », un exemplaire du fichier de sortie, et une version « PDF » de votre rapport.
Comment générer le fichier « tp2E06.zip » : sous le système Linux, vous pouvez utiliser la commande
[zip tp2E06.zip Voyage.h Voyage.cpp TransAvion.h TransAvion.cpp TransVdL.h TransVdL.cpp TransBase.h TransBase.cpp monficsortie.txt tp2E06rapport.pdf]. Sous le système Windows, vous créez d'abord un nouveau fichier compressé. Vous renommez par la suite ce fichier en « tp2E06.zip ». Finalement, vous faites glisser les fichiers un à un dans « tp2E06.zip ».
Comment produire un rapport dans le format « pdf » : voir le site web du cours, rubrique « Foire aux questions ».
Commencer d'abord par vous connecter sur la machine « remise » comme suit : « ssh remise ». Par la suite :

1. Envoyer votre fichier « tp2E06.zip » par la procédure de remise électronique habituelle (Pour obtenir de l'aide sur cette commande, tapez dans un Xterm : man remise). Respecter les noms des fichiers.

   remise ift1169 tp2 tp2E06.zip

2. Vérifier que la remise s'est effectuée correctement.

   remise –v ift1169 tp2

Barème : Ce TP2 est noté sur 13 points.

Compilation et respect des spécifications	**1**
Codage, commentaires etc.	**5**
Rapport	**4**
Tests (fournis et non fournis)	**3**

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.
- La non remise papier vous pénalise de 1 point.
- Les programmes ne contenant pas d'en-tête, -1 point.
- Un programme qui ne compile pas : 0.
- Un programme qui compile mais ne fait 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.
- Le non respect du nom du fichier entraîne une erreur de compilation donc un des points de la spécification n'a pas été respecté : 0.
- Aberration dans le codage : Même si tous les chemins mènent à Rome, faites l'effort nécessaire pour éviter de prendre le plus long!

Automatisation de la correction : Nous allons utiliser des scriptes pour compiler, exécuter et corriger votre programme. Assurez-vous de respecter les noms des fichiers.
Pour comparer votre sortie avec celle fournie en exemple, rediriger votre sortie dans un fichier avec le symbole « > » comme suit : « tp2.exe > masortie.txt ». Par la suite, comparez la au caractère près avec la sortie fournie en exemple, en vous aidant pour cela de la commande « diff » (sous « MinGW/MSys » sinon la commande « fc » sous « DOS », ajustez juste les options) comme suit : **diff –b –w –i –B sortietest.txt votresortie.txt**
Si les fichiers sont identiques, vous n'obtiendrez rien en sortie. Sinon, vous obtiendrez les différences entre les deux fichiers. Pour plus de détails sur cette commande « unix », sur votre console « xterm », exécutez la commande suivante: « man diff ».
Si la différence est provoquée par des caractères non imprimables (« bizarroïdes »), il faudra vous assurer que le fichier texte est dans le bon format d'encodage. Pour cela, utilisez par exemple :

« Conversion dos à unix » : **dos2unix < entree > sortie**
« Conversion unix à dos » : **unix2dos < entree > sortie**

Par ailleurs, nous utiliserons d'autres jeux de test pour la correction.

Ainsi donc au moment de la remise de votre travail, vous avez déjà une idée approximative de la note que vous allez obtenir. Si la commande « diff » ne signale pas de différences entre les deux fichiers, vous êtes sur la bonne voie pour obtenir une note élevée.

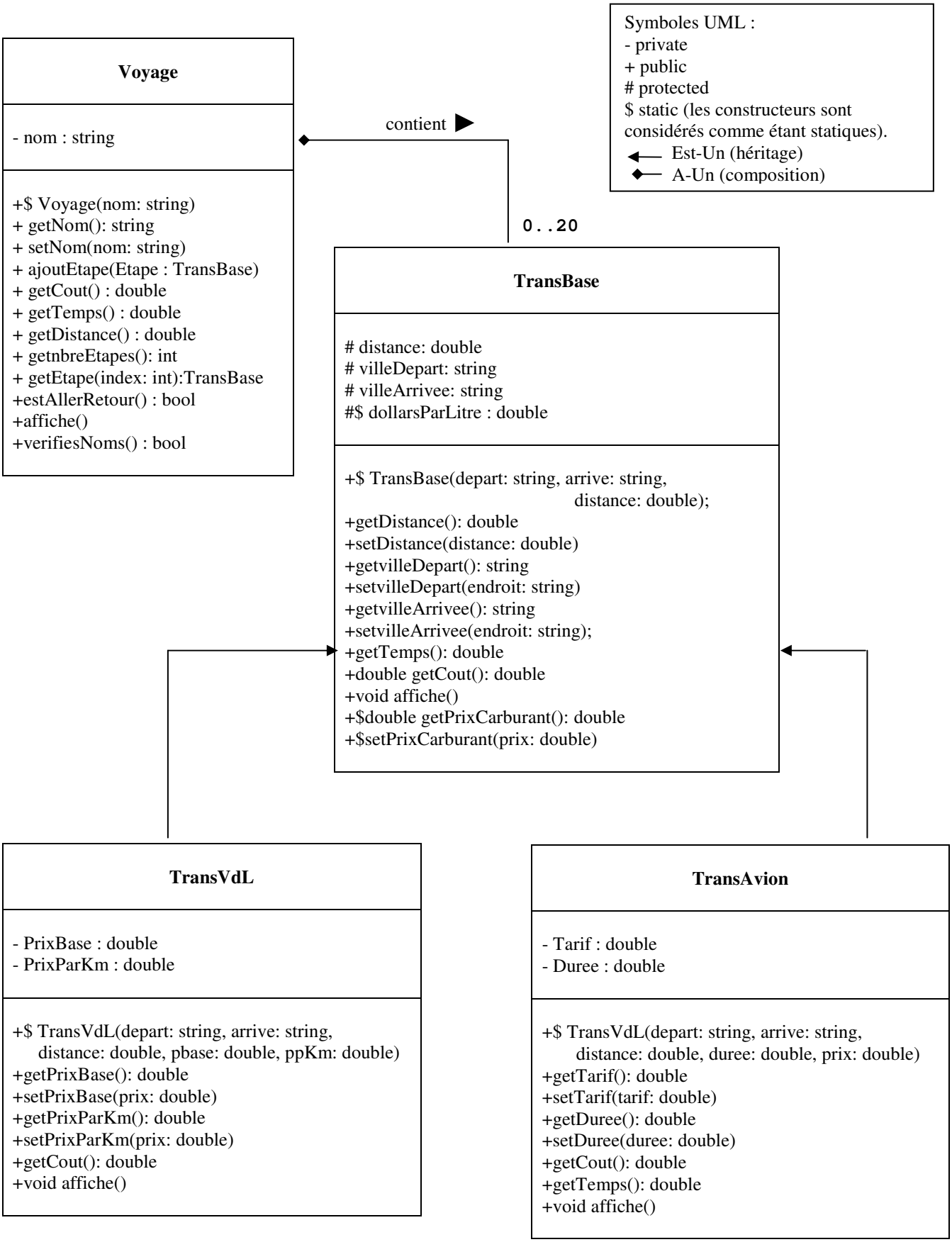
Des questions à propos de ce TP?

Une seule adresse : dift1169@iro.umontreal.ca

Pour faciliter le traitement de votre requête, inclure dans le sujet de votre email, au moins la chaîne: [IFT1169] et une référence au **tp02**.

Mise à jour

14-05-2006 diffusion de l'énoncé.


Symboles UML :
- private
+ public
# protected
$ static (les constructeurs sont considérés comme étant statiques).
Est-Un (héritage)
A-Un (composition)
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
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()