

Compte-rendu

TP 2 – POO : héritage et polymorphisme

Yohan GRACIA et Zifan YAO | 3IF-POO1 | Décembre 2016

[Introduction 2](#_Toc468755953)

[Choix d’implémentation 2](#_Toc468755954)

[Arbre d’héritage : 2](#_Toc468755955)

[Structure de données : 2](#_Toc468755956)

[Manuel 3](#_Toc468755957)

[Classe Trajet : 3](#_Toc468755958)

[ATTRIBUTS: 3](#_Toc468755959)

[CONSTRUCTEUR: 3](#_Toc468755960)

[METHODES : 3](#_Toc468755961)

[Classe TrajetSimple : 3](#_Toc468755962)

[ATTRIBUTS : 3](#_Toc468755963)

[CONSTRUCTEUR: 4](#_Toc468755964)

[METHODES : 4](#_Toc468755965)

[Classe Parcours : 4](#_Toc468755966)

[ATTRIBUTS : 4](#_Toc468755967)

[CONSTRUCTEUR: 4](#_Toc468755968)

[METHODES : 4](#_Toc468755969)

[Classe TrajetCompose : 5](#_Toc468755970)

[Classe Catalogue : 5](#_Toc468755971)

[Conclusion 8](#_Toc468755972)

[Annexe : code source 8](#_Toc468755973)

# Introduction

L'objectif de ce TP était de nous familiariser avec le langage C++, notamment avec le concept de classe, d'héritage, de polymorphisme et l'utilisation des pointeurs. Le sujet du TP était de réaliser un catalogue de trajets simples et composés, de pouvoir afficher ce catalogue et d'implémenter une fonction de recherche de parcours simple et une fonction de recherche de parcours avancée.

# Choix d’implémentation

//todo : décrire les choix qu’on a fait : pk l’arbre d’héritage, les algo de rechercge etc

# Arbre d’héritage :

// graphe d’héritage

# Structure de données :

//avec dessin de la mémoire : t’avait raison yohan !

# Manuel

## Classe Trajet :

La classe Trajet est la classe mère de TrajetSimple et TrajetCompose, elle contient donc les attributs et méthodes communes à ces deux classes.

ATTRIBUTS:

char\* depart, char\* arrivee ;

Ces deux char\* représentent respectivement le point de départ et d'arrivée d'un trajet. Ce sont deux attributs protected.

CONSTRUCTEUR:

Trajet (char\* unDepart, char\* unArrivee );

Le constructeur de Trajet prend en paramètre deux pointeurs char\* et fait pointer les attributs de Trajet vers les char pointés par les paramètres formels.

### METHODES :

virtual void afficherTrajet() const =0;

Description: La méthode afficherTrajet est abstraite pure, car on ne cherchera jamais à l'utiliser pour un Trajet, et qu'elle ne s'implémentera pas de la même façon pour TrajetSimple et TrajetCompose. Le const présent ici s'assure que l'on ne modifie pas le Trajet invoquant la méthode au cours de son utilisation.

virtual void afficherTrajet() const =0;

Description: La méthode comparerTrajet sert pour la réalisation de la recherche de parcours simple. Elle prend en paramètre formels deux char\*, les compare aux attributs du Trajet invoquant la méthode et renvoie un booléen, true si ils sont identiques et false dans le cas contraire.

## Classe TrajetSimple :

La classe TrajetSimple hérite de la classe Trajet, et dispose donc des attributs et des méthodes de sa classe mère. Elle dispose cependant de spécificités qui la différencient de TrajetCompose notamment.

### ATTRIBUTS :

char\* transport ;

Cet attribut représente le moyen de transport permettant le voyage entre le départ et l'arrivée du Trajet. C'est un attribut protected.

### CONSTRUCTEUR:

TrajetSimple (char\* unDepart, char\*unArrivee, char\* unTransport);

Le constructeur de TrajetSimple prend en paramètre formels 3 pointeurs char\*. Il fait appel au constructeur de Trajet en plaçant les 2 premiers pointeurs en paramètre, puis fait pointer l'attribut transport vers le char pointé par le troisième paramètre formel.

### METHODES :

void afficherTrajet() const;

Description: La méthode afficherTrajet est redéfinie ici (car abstraite dans la classe mère). Elle ne renvoie aucun paramètre et ne modifie pas le TrajetSimple appelant la méthode. Elle affiche les attributs du TrajetSimple appelant la méthode sous cette forme:

//TODO Compléter l'exemple

Ville de départ : 'départ'

Ville d'arrivee: 'arrivee'

Moyen de transport: 'transport'

## Classe Parcours :

Comme nous l'avons expliqué précédemment, nous avons choisi de réaliser une liste chaînée pour structurer nos données. La classe Parcours est un élément de cette liste, avec un pointeur vers un Trajet et un pointeur vers le Parcours suivant.

### ATTRIBUTS :

Trajet \*trajetAssocie, Parcours \*nextParcours;

Ces attributs représentent, comme dit dans la description de la classe, un pointeur vers un Trajet (qui peut donc être simple ou composé) et un pointeur vers l'élément suivant de la liste chainée. Ces attributs sont publics afin que l'on puisse y accéder depuis la Classe Catalogue et TrajetCompose. (Nous aurions pu mettre les attributs en friend de ces deux classes mais nous n'étions pas sûr d'être autorisés à le faire car ce TP porte sur le cours POO1)

### CONSTRUCTEUR:

Parcours (Trajet \*unTrajet);

Ce constructeur fait pointer son attribut trajetAssocie vers le même Trajet que celui pointé par le paramètre formel, qui est un pointeur vers un Trajet. On initialise ensuite nextParcours à NULL, afin que nextParcours ne pointe pas une case mémoire dont on ne connait rien.

### METHODES :

La classe Parcours n'a pas de méthodes autres que sont constructeur et destructeur.

## Classe TrajetCompose :

La classe TrajetCompose hérite de la classe Trajet, et dispose donc des attributs et des méthodes de sa classe mère. Elle dispose cependant de spécificités qui la différencient de TrajetSimple, notamment le fait que TrajetCompose dispose d'un pointeur vers un début de liste chaînée (des éléments de la classe Parcours). Un TrajetCompose n'est en fait qu'une liste de TrajetSimple.

ATTRIBUTS :

Parcours\* listeEscales;

Cet attribut est le pointeur vers le premier élément de la liste chaînée. C'est un attribut protected.

CONSTRUCTEUR:

TrajetCompose(char\*unDepart,char\*unArrivee,Parcours\*uneListeEscales);

Ce constructeur prend deux char\* et un Parcours\* en paramètres formels. Il fait tout d'abord appel au constructeur de Trajet en y passant en paramètre les deux char\*. Il copie ensuite l'adresse contenue dans le troisième paramètre pour la mettre dans listeEscales, qui pointe donc vers un Parcours, qui sera le premier élément de la liste représentant le Trajet composé.

METHODES :

void afficherTrajet() const;

Description: La méthode afficherTrajet est redéfinie ici (car abstraite dans la classe mère). Elle ne renvoie aucun paramètre et ne modifie pas le TrajetCompose appelant la méthode. Elle parcourt la liste chaînée associée au TrajetCompose, et pour chaque Parcours, fait appel à la méthode afficherTrajet() de l'attribut "trajetAssocie". Comme chaque TrajetCompose peut se simplifier en une succession de TrajetSimple, la méthode afficherTrajet() de TrajetCompose appelle donc plusieurs fois la méthode afficherTrajet() de TrajetSimple.

//TODO Exemple

## Classe Catalogue :

Cette classe est la plus complexe de celles que nous avons implémentées car elle nécessite toutes les autres classes pour fonctionner. Elle est composée d'une classe servant de menu, qui permet de choisir les différentes opérations réalisables sur ce catalogue (l'afficher, ajouter un TrajetSimple ou un TrajetCompose, faire une recherche simple ou avancée et quitter le catalogue). Comme expliqué précédemment, un Catalogue est une liste chaînée de Trajet.

ATTRIBUTS :

Parcours \* listeTrajets, char \* nomCatalogue;

Le premier attribut est un pointeur vers un Parcours, qui contient le premier élément du Catalogue. Le second attribut correspond à un pointeur vers le nom du Catalogue, afin de pouvoir simplement identifier le Catalogue courant.

CONSTRUCTEUR:

Catalogue (char\* unNom);

Le constructeur prend en paramètre formel un char\* , et le copie dans nomCatalogue. Le constructeur initialise un TrajetSimple avec ' unNom ' pour les 3 attributs. Le constructeur initialise ensuite un Parcours à partir d'un pointeur vers ce TrajetSimple, et fait pointer l'attribut ' listeTrajets' sur ce Parcours.

METHODES :

void MenuCatalogue ();

Description: Cette méthode, comme son nom l'indique, est un menu associé au catalogue l'ayant appelée. Cette méthode ne renvoie aucun paramètre. Elle demande à l'utilisateur de choisir une opération, et agis en fonction de sa réponse. Voici l'interface donnée à l'utilisateur:

"Choisissez une operation:

1: Afficher le catalogue

2: Ajouter un trajet simple

3: Ajouter un trajet compose

4: Faire une recherche simple de parcours

5: Faire une recherche avancee de parcours

6: Quitter le catalogue"

Si l'utilisateur donne un nombre autre que ceux indiqués, le message " Cette option n'est pas dans le catalogue " s'affiche et le menu s'affiche à nouveau, lui permettant de choisir une des options proposées. Choisir une réponse entre 1 et 5 fera appel à une nouvelle méthode (associée à l'action demandée) et choisir 6 quittera la méthode.

void AfficherCatalogue () const;

Description: Cette méthode ne renvoie aucun paramètre. Elle ne modifie pas la Catalogue qui l'appelle. Cette classe parcourt la liste chaînée et affiche chaque Trajet associé aux Parcours en faisant appel à leur méthode afficherTrajet.

//TODO Exemple

Si le Catalogue est vide, la méthode affiche:

La catalogue est vide

void AddToCatalogue (Trajet \*unTrajet);

Description: Cette méthode ne renvoie aucun paramètre. Elle prend en paramètre un pointeur vers un Trajet. Elle crée un Parcours\* à partir de ce pointeur et insère le parcours associé à la fin de la liste chaînée du Catalogue.

void AddToCatalogueTSSaisie();

Description: Cette méthode ne renvoie aucun paramètre. Cette méthode demande à l'utilisateur de rentrer 3 chaines de caractères, ayant cette présentation:

Donnez la ville de depart

Donnez la ville d'arrivee

Donnez le moyen de transport

La méthode copie ensuite ces chaines de caractères dans des zones allouées à des char\* afin de gérer la mémoire d'une façon optimale. La méthode construit ensuite un TrajetSimple\* en utilisant les 3 char\* comme paramètres. Cette méthode fait ensuite appel à AddToCatalogue(Trajet \*unTrajet) pour ajouter ce TrajetSimple au catalogue.

void AddToCatalogueTCSaisie();

Description: Cette méthode ne renvoie aucun paramètre. Cette méthode demande à l'utilisateur de rentrer 4 chaines de caractères, ayant cette présentation:

Donnez la ville de depart du trajet compose

Donnez la ville d'arrivee du trajet compose

Donnez la ville d'arrivee du premier trajet simple

Donnez le moyen de transport du premier trajet simple

La méthode copie ensuite ces chaines de caractères dans des zones allouées à des char\* afin de gérer la mémoire d'une façon optimale. La méthode construit ensuite un TrajetSimple\* en utilisant les 1er, 3ème et dernier char\* comme paramètres. La méthode construit alors un Parcours\* à partir de ce TrajetSimple\*. En utilisant le premier et second char\* ,ainsi que le Parcours\* comme paramètres, la méthode construit un nouveau TrajetCompose\*. Cette méthode fait ensuite appel à AddToCatalogue(Trajet \*unTrajet) pour ajouter ce TrajetCompose au catalogue. Pour finir de compléter le TrajetCompose, la méthode fait appel à la fonction AddToCatalogueTCFin(char\* arriveeC, Parcours\* premierParcours) en mettant en paramètres la ville d'arrivée du TrajetCompose et le pointeur vers le premier Parcours du TrajetCompose et ce jusqu'à ce que cette méthode renvoie true.

bool AddToCatalogueTCFin(char\* arriveeC, Parcours\* premierParcours);

Description: Cette méthode renvoie un booléen. Elle complète un TrajetCompose. Elle prend en paramètre un char\* correspondant à la ville d'arrivée du TrajetCompose que l'on complète, ainsi qu'un Parcours\* désignant le premier élément du TrajetCompose. Cette méthode demande à l'utilisateur de rentrer des informations sur les trajet intermédiaires faisant partis du TrajetCompose.

Donnez la ville de depart de votre trajet intermediaire

Donnez la ville d'arrivee de votre trajet intermediaire

Donnez le moyen de transport de votre trajet intermediaire

La méthode copie ensuite ces chaines de caractères dans des zones allouées à des char\* afin de gérer la mémoire d'une façon optimale. La méthode construit ensuite un TrajetSimple\* en utilisant les 3 char\* comme paramètres.

La méthode construit alors un Parcours\* à partir de ce TrajetSimple\*. La méthode insère ensuite ce Parcours en fin de liste. Si la ville d'arrivée rentrée par l'utilisateur correspond au char\* qui était en paramètre, la méthode renvoie true . Sinon elle renvoie false.

void RechercheSimple();

Description: Cette méthode ne renvoie pas de paramètre. Elle demande à l'utilisateur de rentrer 2 char qui correspondent au départ et à l'arrivée recherchés. La méthode parcourt ensuite tout le catalogue, et appelle la méthode comparerTrajet en passant en paramètre les 2 char\* entrés par l'utilisateur. Lorsque que le Trajet correspond à la recherche, la méthode fait appel à la méthode afficher du Trajet.

# Conclusion

//todo : faire une conclusion du tp : ce qui a bien marché, ce qui était difficile, ce qu’on a retenu.

# Annexe : code source