

Compte Rendu du TD numéro 5 de l'équipe Eirb'reteau

TD du 17 octobre, pour le 21 octobre

Tandem 1 : Aurélien Nizet, Lionel Adotevi Coordinateur : Pierre Gaulon

Relation de type/sous-type

Dans cette section, nous allons établir des relations de type/sous-type

Cela ne peut pas être fait avec une interface, qui ne peut pas contenir d'attributs. Il n'est donc pas possible d'utiliser seulement un héritage d'interfaces dans cet objectif de factorisation de code. la montée ou à un arrêt, change. Pour factoriser le code, il est nécessaire de

va donc rester indépendant de sa réalisation. Bens l'exemple du cours, c'est un lien a-un qui est utilisé. La classe Verminimal, une sorte de cahier des charges que devront respecter tous les passagers. Garder Passager sans code est en faveur de l'abstraction. Le Passager L'objectif de garder Passager sans code est de définir un comportement yaya besomdi

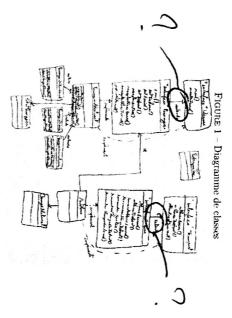
rou contient des instances des classes PorteCharniere et PorteCoulissante, et il applique ses méthodes sur ces instances. En fait, ces instances sont déclarées du type de leur classe mère, la classe Porte. Ainsi, les mécanismes de polymorphisme se chargent du reste.

En utilisant la solution de relation de type/sous-type, et non de lien a-un, l'objectif est toujours le même : l'abstraction, rendre les objets les plus indépendants les uns des autres en favorisant l'encapsulation. tout a qui est

nois whitesetton d'en angunlach et à chaque kouveaux pussager attigation du modifiet le code la relation de typelsoustype int si tit diajouter

L'héritage de classe

une classe nommée Passager Abstrait, sous-type de Passager et Usager. Les classes PassagerStandard, PassagerStresse et PassagerLunatique en héritent Nous rassemblons maintenant le code commun à toutes les classes, dans



2.1 La classe Passager Abstrait

sont supprimées de cette classe. Une méthode qui ne définit pas toutes les méthodes des interfaces qu'elle implémente, qui est incomplète, doit être abstraite. En effet, elle ne doit pas pouvoir être instanciée, puisqu'il reste ret(). Or, ces méthodes ne figurent pas dans PassagerAbstrait, puisqu'elles Cette classe Passager Abstrait implémente l'interface Passager et l'inter-

Cela n'aurait pas été possible sans constructeur dans cette classe abstraite, même si elle n'est pas instanciable. de la classe mère, peut-être même en utilisant des méthodes de la classe mère. constructeur peut être utilisé par les classes filles pour initialiser les attributs Cette classe abstraite PassagerAbstrait possède un constructeur car ce

tactories l'indonciative et Printabation

and divide by his directions of the land o

monterDans() et nouvelArret(). Passager Abstrait, et à définir le comportement standard pour les méthodes La classe PassagerStandard est remaniée, de façon à hériter de la classe

classe PassagerAbstrait peut être modifiée par d'autres classes du paquet gnifie qu'elle est accessible dans le paquet tec. Ainsi, cette variable de la Si on définit la portée de la variable destination en protected, cela si-

la classe PassagerAbstrait, qui compte deux paramètres, il va y avoir une erreur, indiquant qu'il manque des paramètres dans ce constructeur per(), sans paramètres. La méthode super() se référant au constructeur de hérite de PassagerAbstrait, le code vide va être remplacé par l'appel à su-

avant ceux de la classe dérivée, et le code du constructeur de la classe dérivée celui de la classe dérivée, car les attributs de la classe de base sont initialisés peut utiliser des méthodes de la classe de base. Il est obligatoire de déclencher un constructeur de la classe de base avant

2.3 Les deux nouveaux caractères

tique et PassagerStresse est contenu dans les deux méthodes nouvelArret() et monterDans() de PassagerAbstrait : Le code en commun entre les classes PassagerStandard, PassagerLuna-

- Dans nouvelArret(), on trouve le code if(destination == numeroArret)
- Dans monterDans(), on trouve le cast
 Bus b = (Bus) t; bus.demanderSortie(this);

Définir ce qu'il faut paramétres

ret() et monterDans() dans la classe abstraite, et y déclarer deux nouvelles afin de le factoriser. Pour ce faire, il faut transferer les méthodes nouvelArcune des classes dérivée définira selon le comportement souhaité. méthodes abstraites choixPlaceMontee() et choixChangerPlace(), que cha-Nous allons mettre dans la classe PassagerAbstrait le code dupliqué,

bonne classe. Cette liaison est dynamique, elle se fait à l'exécution du programme en fonction de l'instance manipulée, pour obtenir l'adresse de la rectement les méthodes choixChangerPlace() et choixPlaceMontee() vers la C'est le mécanisme de polymorphisme qui va se charger d'aiguiller cor-

-> ous in grace our polymorphiere.

possibility (discussion ent)

Mary Services

3.1 La classe PassagerAbstrait

sont des méthodes abstraites. Si on met un corps vide pour ces deux méthodes traites), on laisse alors le soin à chaque classe dérivée de définir ces méthodes on sera alors obligés de les réécrire dans les classes dérivées (@Override), comme elle l'entend, sans la réécrire. Ces deux méthodes ont une portée pualors que si on laisse sculement la signature de ces méthodes (méthodes absolique par défaut. Les deux nouvelles méthodes choixPlaceMontee() et choixChangerPlace()

redéfinies, il faut ajouter le mot-clé final à ces deux méthodes. Elles ne pourront ainsi pas être réécrites. Pour s'assurer que les méthodes monterDans() et nouvelArret() ne soient

3.2 Les classes caractères

Lunatique, et PassagerStandard, ainsi que leurs tests. Nous écrivons maintenant le code des classes PassagerStresse, Passager-

autre classe n'en hérite. Ces classes de caractères sont les classes de plus bas ou aucune autre classe qui en dérive. Cela empéche ainsi tout utilisation niveau, c'est-à-dire qu'il n'est pas envisageable qu'il y ait de sous-caractères inappropriée de ces classes. Ces classes de caractères doivent être déclarées final, pour qu'aucune

Factoriser les tests

, type de retour utilisé est PassagerAbstrait, puisque d'une part nous voulons seulement tester les PassagersAbstraits, et non les Passagers ou les Usagers, classes à instancier et d'autre part, parce que ces deux derniers sont des interfaces, et non des abstraite servant de constructeur, qui est redéfinie dans chaque classe dérivée TestPAssagerStandard, TestPassagerLunatique et TestPassagerStresse. Le Le classe TestPassagerAbstrait est définie abstraite, et possède une méthode

Commentaires

5.1 Commentaire de Pierre

créer beaucoup de classe, mais que chacune ait un rôle bien particulier et C'est cette approche qui nous a été suggérée ici, notamment par le fait de sulation et la factorisation de code. En effet, il y a une marge entre produire factoriser le code. Le but étant d'avoir un code facilement maintenable pour propre, l'encapsulation, et par le fait de créer des classes abstraites pour un code qui fonctionne, et produire un code bien pensé, propre et concis Je pense que les deux notions importantes à tirer de ce TD sont l'encap-

を見られていることが、 これをいることがなって、 またっていることが、 またっていることが、 またのでは、

1

gagner du temps par la suite. Ici, on s'est donc abstraits des problèmes techniques du langage en lui-même, pour s'intresser plutôt aux fondements de la programmation orientée objet.

5.2 Commentaire d'Aurélien

Ce TD consistait à créer la classe abstraite PassagerAbstrait ainsi que les classes PassagerLunatique et PassagerStresse. Bien que le code à écrire n'était pas conséquent, il a fallu bien faire attention lors de la factorisation du code et ainsi remanier correctement le code des fichiers sources ainsi que des fichiers tests. Je me suis également rendu compte dans ce TD que ce n'est pas simple de fournir du code bien factorisé, avec des tests unitaires également factorisés. Mais le fait est qu'une fois le code correctement écrit, il est alors beaucoup plus facile de modifier par exemple une fonction bien précise sans avoir à retoucher tout le code!

5.3 Commentaire de Reda

Dans ce TP, j'ai appris à faire évoluer des classes afin de les rendre plus réalistes (les types de passagers). Grâce au système d'héritage sur les classes abstraites cela se fait plutôt simplement. De plus, j'ai pu découvrir l'utilisation et l'intérêt des exceptions.

5.4 Commentaire de Victor

J'ai trouvé ce TD agréable car nous avons pu voir l'abstraction de plusieurs classes en une classe pour éviter le code commun. Nous nous doutions que nous pouvions faire une telle chose, mais nous avons pu constater que ça fonctionnait bien.

5.5 Commentaire de Lionel

Ce TD était pour moi un approfondissement des notions d'héritage, d'abstraction mais aussi un plus sur la factorisation de code et l'importance des tests unitaires. Cela montre les différentes étapes à respecter lors de l'écriture d'un code.

- Fabrique unaque levertique & Stesse - Passager extends Verger ?? | acec les missager le partire le 7 7 ?? | methode ? et code raisonnable.