Est-un ou A-un

Dans le TD précédent, nous avons vu comment définir une classe de base en factorisant le code commun mais en laissant certaines parties variables dans les classes dérivées.

La classe abstraite **PassagerAbstrait** permet aux classes dérivées d'effectuer deux personnalisations :

— le choix d'une place pour monter dans le bus,

— le choix de l'action à chaque arrêt du bus.

Fixons quatre manières de monter dans le bus ; c'est à dire d'implanter la méthode **choixPlaceMontee**() :

Fatigue prendre une place assise sinon ne pas entrer,

Tetu prendre une place debout même si le bus est plein,

Repos prendre une place de préférence assise sinon debout,

Sportif prendre une place de préférence debout sinon assise.

Fixons cinq actions possibles à un arrêt ; c'est à dire d'implanter la méthode **choixChangerPlace**() :

Calme ne pas changer de place,

Nerveux changer de place à chaque arrêt,

Prudent choisir une place assise si le passager est à plus de 5 arrêts de sa destination et une place debout si le passager est à moins de 3 arrêts de sa destination,

Agoraphobe sortir si le bus n'a plus de place assise ou de place debout,

Poli laisser sa place assise si le le bus n'a plus de place assise.

L'objectif est de concevoir une architecture de classes qui permet de réaliser les 20 combinaisons. On ne s'occupe pas de l'incohérence de certaines combinaisons.

Table des matières

1 Quel lien choisir ?

Étudions le lien à utiliser entre ces nouvelles classes et la classe abstraite **PassagerAbstrait**. Il y a deux liens possibles un lien est-un ou un lien a-un. Quelle est la différence essentielle entre ces deux liens ?

Un lien “est-un”

La manière la plus directe de réaliser ces classes est d'utiliser l'héritage et de redéfinir les deux méhodes abstraites.

Construire le diagramme de classes.

Combien faut-il construire de classes concrêtes ?

Quel code est dupliqué ?

Combien faut-il construire de classes si un autre caractère de passager est ajouté ?

Donner vos conclusions.

Un lien “a-un”

Pour éviter cette duplication de code, nous ne pouvons pas utiliser un lien est-un pour toutes les classes.

Pour les 4 classes **Fatigue**, **Tetu**, **Repos**, **Sportif**, nous avons toujours un lien “est-un”. Les classes **Fatigue**, **Tetu**, **Repos**, **Sportif** héritent de **PassagerAbstrait** et redéfinissent la méthode **choixPlaceMontee**().

Par contre nous utilisons un lien “a-un” pour les cinq classes **Calme**, **Nerveux**, **Prudent**, **Agoraphobe**, **Poli**. La classe **PassagerAbstrait** posséde un lien “a-un” avec les classes **Calme**, **Nerveux**, **Prudent**, **Agoraphobe**, **Poli**.

Ces cinq classes contiennent la méthode **choixChangerPlace**() et son implantation.

Si une classe A posséde un lien a-un avec une classe B, quelle conséquence cela a sur les instances ?

Donner le diagramme de classes de cette architecture.

Déduire le nouveau prototype de la méthode **choixChangerPlace**().

Pourquoi n'y-a-t-il pas besoin de constructeur pour ces cinq classes ?

Indice : Dans le code de la classe abstraite, il faut pouvoir substituer des instances de ces classes.

Combien faut-il construire de classes concrêtes ? Y-a-t-il duplication de code ?

Combien faut-il construire de classes si un autre caractère de passager est ajouté ?

Remarque : La relation d'héritage pour les quatre classes **Fatigue**, **Tetu**, **Repos**, **Sportif** peut être aussi remplacée par un lien “a-un”.

Musclez vos Neurones.

Supposons que java autorise l'héritage multiple.

Remplacer le lien “a-un” précédent par une relation d'héritage (“est-un”).

Construire le diagramme de classes.

Combien faut-il construire de classes concrêtes ? Y-a-t-il duplication de code ?

Combien faut-il construire de classes si un autre caractère de passager est ajouté ?

En conclusion : Quelles différences pouvez vous faire entre un lien “est-un” et un lien “a-un”.

2 Réalisation du lien “a-un”

Il reste à coder cette architecture. Avec l'héritage, l'instance d'une sous-classe contient les variables des classes de base. Par contre avec un lien a-un vous avez forcément deux classes distinctes. Dans notre architecture, pour instancier une des quatre classes **Fatigue**, **Tetu**, **Repos**, **Sportif**, il nous faut obligatoirement une instance d'une des cinq classes **Calme**, **Nerveux**, **Prudent**, **Agoraphobe**, **Poli**.

Question d'instanciation.

Il n'y a pas de questions idiotes : Où se fait l'instanciation des cinq classes **Calme**, **Nerveux**, **Prudent**, **Agoraphobe**, **Poli** ?

Etudier les deux manières suivantes de faire cette instanciation :

1. Faire l'instanciation dans le constructeur d'une sous-classe de **PassagerAbstrait** ; l'instanciation se fait à l'intérieur des sous-classe.
2. Passer l'instance comme arguments aux constructeurs des sous-classes ; l'instanciation se fait à l'extérieur des sous-classes (par exemple le programme principal).

Expliquer pourquoi choisir la deuxième manière.

Décidons que le programme principal se charge de l'instanciation, donner la conséquence sur l'accès de ces cinq classes.

Travaux Pratiques.

Un tandem se charge des quatre classes **Fatigue**, **Tetu**, **Repos**, **Sportif** et l'autre tandem des cinq classes **Calme**, **Nerveux**, **Prudent**, **Agoraphobe**, **Poli**.

N'oubliez pas d'écrire les tests pour chaque classe.

Pour éviter de perdre du temps à les adapter, supprimer les trois classes **PassagerStandard**, **PassagerLunatique**, **PassagerStresse** et leur tests. La création des nouvelles classes se fera comme suit :

— Le comporetment Repos et Calme est identique à celui du **PassagerStandard**
— Le comporetment Fatigue et Prudent est identique à celui du **PassagerStresse**
— Le comportement Sportif et Nerveux est identique à celui du **PassagerLunatique**
— Pour les autres, proposer un comportement

Décider dans le programmme principal des combinaisons de caractères des passagers.

Musclez vos Neurones.

Voici quelques tractions pour muscler vos neurones

Première traction L'interface **Bus** et **Passager** sont privées au paquetage tec pour éviter que le client utilise directement certaines méthodes comme **accepterPlaceAssise**()

Comme le programme principal instancie nos passagers, les classes passagers doivent être accessibles en dehors du paquetage tec. Montrer qu'il est alors possible de faire appel à **accepterPlaceAssise**() directement dans le programme principal. Expliquer pourquoi ?

Comment assurer complétement le masquage d'information ?

indice : Penser au classe abstraite.

Deuxième traction Pour éviter de passer systématiquement les deux paramètres **Bus** et **Passager** à chaque appel de la méthode **choixChangerPlace**(), il suffit de les stocker avec deux variables d'instance dans nos cinq classes **Calme**, **Nerveux**, **Prudent**, **Agoraphobe**, **Poli**. Chaque instance de ces classes a un état contenant le bus et le passager qui interagissent.

Pourquoi profirer la solution d'instance sans état ?

3 Nouvelle architecture

Présenter sur un diagramme de classe la nouvelle architecture obtenue.