

<p align="center">Programmation orientée objet II Hiver 2023 Cégep Limoilou Département d'informatique</p> <p>Professeur : Martin Simoneau</p>	<p align="center">Formatif 1 S1-C1</p> <p align="center"><i>Réutilisation avec héritage et délégation</i></p>
--	--

Exercice1 - Vente et tip (extends)

S2-C1-p2

Comment réutiliser une méthode

a) avec l'héritage :

- Copiez les classes *ProduitEnVente* et *ProduitTip* dans le sous-package *heritage*.
- Créez une nouvelle classe nommée ***ProduitTaxe*** dans le package *heritage*.
- Copie la méthode *calculeTaxe()* et la constante *TAUX_TAXE* dans la nouvelle classe *ProduitTaxe*. Effacez-les dans les classes *ProduitEnVente* et *ProduitTip*.
- Faites hériter *ProduitEnVente* et *ProduitTip* de *ProduitTaxe* en ajoutant ***extends ProduitTaxe*** après la déclaration de classe.

Exemple : `public class ProduitAvecTip extends ProduitTaxe`

- Comme la classe parente fournit la méthode *calculeTaxe* à tous ses enfants, les classes *ProduitEnVente* et *ProduitTip*, continueront de fonctionner parce qu'elles reçoivent la méthode *calculeTaxe* de la classe parent *ProduitTaxe*.

S2-C2-p3

b) avec la délégation :

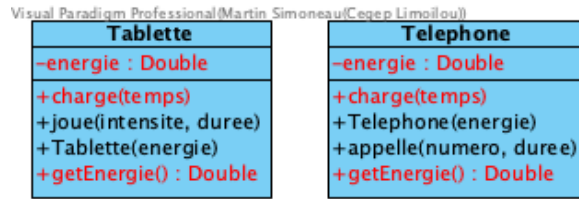
- Copiez les classes *ProduitEnVente* et *ProduitTip* dans le sous-package *delegation*.
- Créez une nouvelle classe nommée ***CalculateurDeTaxe*** qui contiendra la méthode *calculeTaxe()* et la constante *TAUX_TAXE*.
- Dans les classes *ProduitEnVente* et *ProduitTip*, effacez la méthode *calculeTaxe()* et la constante *TAUX_TAXE*.
- Dans les classes *ProduitEnVente* et *ProduitTip*, ajoutez un attribut ***calculateurDeTaxe*** de type *CalculateurDeTaxe*.
- Ajoutez un constructeur pour initialiser l'attribut *calculateurDeTaxe*
- Ajustez le code pour utiliser le *calculateurDeTaxe* dans la méthode.

Pour ce cas, quelle stratégie (délégation ou héritage) vous semble la plus simple et la plus intéressante, pourquoi ?

S2-C1-p4

Exercice 2 – téléphone et tablette (assert, chaînage de constructeurs)

Avant :

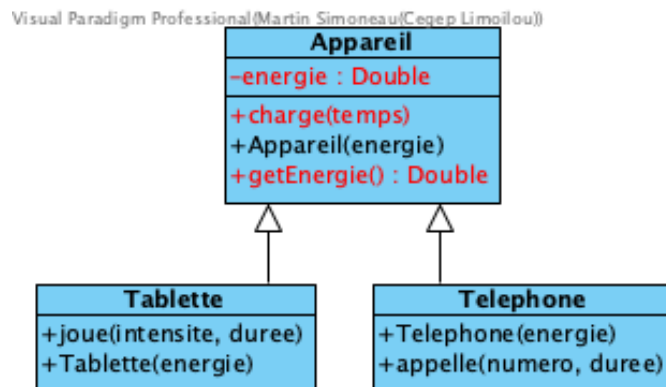


Comment réutiliser une méthode qui dépend d'un attribut! On va refaire le même genre de manipulations que l'exercice 1, mais cette fois on réutilise une méthode qui dépend d'un attribut de la classe. On devra donc déplacer l'attribut *energie* avec les méthodes *getEnergie* et *charge*.

- Faites une copie des classes *Telephone* et *Tablette* dans chacun des packages *delegation* et *heritage*.

a) avec *heritage*

après :



- Créez une nouvelle classe nommée **Appareil**. Elle sera le parent de *Telephone* et *Tablette*. On doit y mettre tout ce qui est commun à *Telephone* et *Tablette* afin de réutiliser ses éléments de code. On parle donc de la méthode *charge()*, mais comme cette dernière dépend de l'attribut *energie* il faudra également déplacer cet attribut. Finalement, le getter de l'attribut doit assurément suivre l'attribut!
 - Déplacez l'attribut *energie* de *Telephone* vers *Appareil*. Effacez l'attribut *energie* dans *Tablette*.
 - Déplacez *getEnergie* de *Telephone* dans *Appareil*. Effacez l'attribut *energie* dans *Tablette*.
 - Faites la même chose avec la méthode *charge*. Cette méthode manipule exclusivement l'attribut *energie*.
 - La classe *Appareil* doit avoir un constructeur pour initialiser l'attribut *energie*. Créez-le.
 - Faites hériter *Tablette* et *Telephone* de *Appareil* (avec *extends*).
 - Vous remarquerez qu'il y a automatiquement une erreur qui est annoncée par le compilateur. Puisque *Appareil* possède un constructeur, on est obligé de l'appeler pendant la construction de chacun de ses enfants. Pour appeler le constructeur de la classe parent, ajoutez :
 - ***super(energie)***

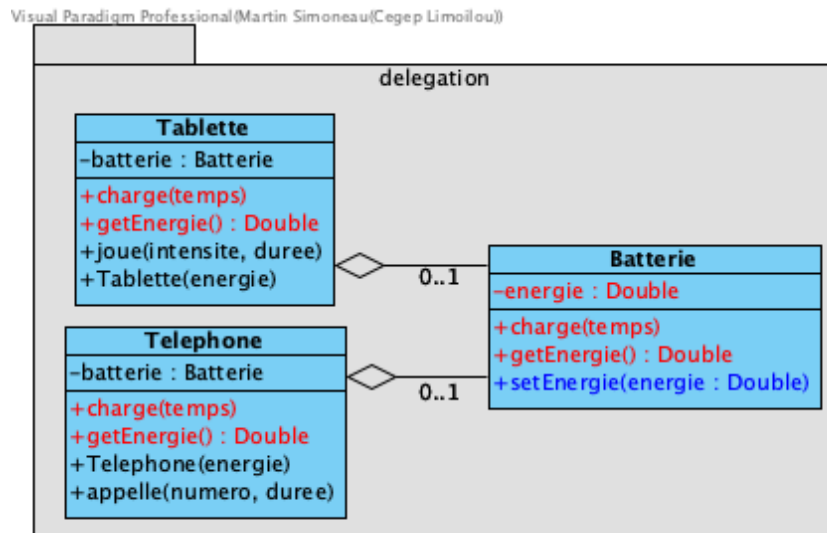
Ainsi la valeur *d'énergie* qui est reçue par *Tablette* ou *Telephone* est transféré à la classe parente *Appareil*. Notez que le constructeur d'une classe parent doit toujours être appelé avant celui de la classe enfant. L'instruction `super(...)` doit donc être la première instruction du constructeur enfant. Essayez de mettre une instruction avant d'appeler `super` et vous recevrez une erreur du compilateur.

- Assurez-vous que les méthodes *main* de *Tablette* et *Telephone* fonctionnent toujours.

S2-C1-p5

b) Avec la délégation

Après :



Ici, on n'utilise pas l'héritage. Vous connaissez déjà tout ce qu'il faut pour réaliser ce projet. C'est la classe *Batterie* qui permet de réutiliser l'énergie et la charge. Programmez ce cas dans le package *delegation*. Commencez par copier les classes *Tablette* et *Telephone* fournies avec le TP.

Pour ce cas, quelle stratégie (délégation ou héritage) vous semble la plus simple et la plus intéressante, pourquoi ?

Exercice 3 – Animaux (*protected*, *extends*, *masquage d'attributs*)

a) Trouver le premier parent

Dans le package *exercice3* vous trouverez 4 classes (*Saumon*, *Ours*, *VerDeTerre* et *Fourmi*). Vous remarquerez que les 4 animaux ont certains attributs en communs.

- Ajoutez une classe ***Animal*** et placez dans cette classe tous les attributs qui sont communs aux 4 animaux.
- Ajoutez un constructeur qui reçoit les valeurs initiales pour chacun des attributs que vous avez mis dans cette classe.

- Faites hériter les 4 classes d'animaux de la classe *Animal*. Faites le ménage de chacun en enlevant les attributs qui lui sont fournis par la classe parente *Animal*. Faites attention à bien connecter les constructeurs et à lui envoyer les bonnes valeurs.

Attention! Si vous n'enlevez pas les attributs équivalents des classes enfants, ces derniers masqueront ceux de la classe parente et vous risquez d'avoir des bogues difficiles à trouver! Le Java permet de redéfinir un même attribut dans une classe enfant (avec le même nom), mais recommande très fortement de ne jamais le faire!

- À ce stade, vous devriez avoir une erreur dans les méthodes *decritOurs()*, *decritSaumons()*... parce qu'elles n'ont plus accès aux attributs du parent *Animal*. Un attribut avec une visibilité privé n'est pas accessible aux enfants de la classe. On pourrait évidemment passer par les getter et setter (plusieurs préfèrent cette approche), mais nous allons plutôt utiliser un nouveau type de visibilité qui sert à rendre un attribut visible par les enfants d'une classe. Changez la visibilité *private* des attributs de *Animal* par ***protected***.

Attention! En Java, les visibilités sont dans l'ordre : *private*, *package*, *protected* et *public*. La visibilité *protected* implique donc la visibilité *package*. Un attribut protégé peut donc être consulté par toutes les classes de son package!

- Assurez-vous que les méthodes main fonctionnent toujours.

b) Trouver un les parents intermédiaires

Une fois que vous aurez terminé l'ajout de la classe *Animal*, vous remarquerez que *Ours* et *Saumon* ont un attribut *estAquatique* en commun. Vous remarquerez également que *Fourmi* et *VerDeTerre* ont l'attribut *nombreDePatte* en commun. Il serait donc approprié d'ajouter une nouvelle classe enfant de *Animal* et parent de *Ours* et *Saumon*, pour gérer l'attribut commun. Faites de même avec *Fourmi* et *VerDeTerre*. Dans ces classes intermédiaires, créez les constructeurs nécessaires et initialisez chacun des attributs au bon endroit. Faites le chaînage des constructeurs et utilisez la visibilité *protected* aux endroits appropriés.





- Assurez-vous que les méthodes main des 4 classes d'animaux fonctionnent toujours.

Exercice 4 – Citoyen et Personne

On vous donne la classe *Personne*. Créez la classe ***Citoyen***. Un Citoyen est une personne qui a un numéro d'assurance sociale. Connectez les constructeurs correctement. Faites également une classe *Application* qui utilise vos classes *Personne* et *Citoyen*. Testez les getters et setters de personne

Exercice 5 – Analyseur de texte

Pour cet exercice, vous devez créer le code au complet. Vous devez créer 3 classes qui font de l'analyse de chaîne de caractères. Les 3 classes possèdent un attribut qui est la chaîne de caractères à analyser (*private String texte*) et une méthode qui convertit cette chaîne de caractères en un tableau de char[] (*private Char[] convertToArray()*). Utilisez l'héritage pour bien réutiliser l'attribut et la méthode. Voici les 3 classes à réaliser.

1. **CompteurMot** :  compte le nombre de mots dans une chaîne de caractères en comptant le nombre d'espaces. Attention, le dernier mot n'est pas suivi d'un espace, mais d'un point.
 Pour vous pratiquer davantage, considérez les espaces doubles. Si un mot est séparé par 2 espaces consécutifs, il ne faut compter qu'un seul mot.
2. **AnalyseurVoyelle** : Compte le nombre de voyelles dans une chaîne de caractères. Faites-le en comparant chacun des caractères du tableau de caractères que vous transmet la méthode *convertToArray* avec un tableau contenant toutes les voyelles.
3. **AnalyseurConsonne** :  Comme pour les voyelles, mais cette fois-ci en comptant le nombre de consonnes.
 Pour vous pratiquer davantage, créez une classe intermédiaire parente de *AnalyseurVoyelle* et *AnalyseurConsonne*. Cette classe intermédiaire fournit une méthode qui retourne vrai si un caractère reçu en paramètre est une voyelle. L'analyseur de consonne peut se servir de cette méthode pour trouver une consonne sachant qu'un caractère est une lettre, mais qu'il n'est pas une voyelle.