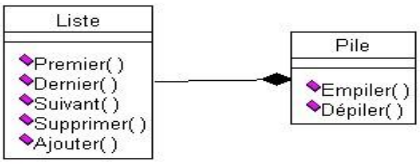
TP 2. Livrable par binôme à déposer sur campus

Diagramme de classes - Classes et objets – Visibilité - Cast – Encapsulation – Héritage – Packages

Veuillez lire tous les questions avant de répondre.

Soient les deux classes *Liste* et *Pile*. Le modèle UML du diagramme de classes est reproduit ci-dessous :



Pour la classe *Liste*, le rôle des méthodes est :

- *Premier()* : sans argument, se positionne sur le premier objet de la liste
- *Dernier()* : sans argument, se positionne sur le dernier objet de la liste
- *Suivant()* : sans argument, retourne l'objet suivant de celui sur lequel on est positionné. Cette méthode affiche un message d'erreur et retourne un objet vide si la liste est vide (position à 0) ou si on est positionné sur le dernier élément.
- *Supprimer()* : sans argument, supprime et retourne l'objet sur lequel on est positionné. Cette méthode affiche un message d'erreur et retourne un objet vide si la liste est vide.
- *Ajouter()* : en argument un objet générique de type **Object** à ajouter dans l'élément sur lequel on est positionné

Indices utiles :

- Pour connaitre la position d'un élément, vous aurez besoin d'un attribut

- La liste d'éléments doit se stocker dans un attribut soit dans d'un tableau, soit dans un **ArrayList** (prochain cours) : voir le tutoriel Exemples avec la classe ArrayList et toutes les caractéristiques de la classe **ArrayList** sur le site suivant : **https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html**
- Si la liste est stockée dans un tableau, sa taille est limitée (ce qui n'est pas le cas pour un **ArrayList**) et elle doit être stockée dans un attribut

Pour la classe *Pile*, le rôle des méthodes est :

- *Empiler()* : en argument un objet générique de type **Object** à empiler
- *Depiler()* : sans argument, retourne l'objet générique de type **Object** à dépiler

Complétez le diagramme des classes avec une nouvelle classe *PileEntiers* qui hérite des propriétés de la classe *Pile*, cette nouvelle classe se spécialisant dans une pile composée uniquement d'entiers. La visibilité des méthodes et attributs des classes n'est pas spécifiée sur le diagramme UML. Vous devez ajouter le nécessaire à votre code pour une encapsulation maximale. Étudiez le niveau de visibilité **protected**, et trouvez où dans votre code il pourrait être d'utilité ...

Indices utiles :

- Un extrait du code de la classe Pile se trouve dans le Cours 2
- La méthode *Depiler()* doit être redéfinie dans la classe *PileEntiers* composé d'objets de type **Integer** en castant la méthode *Depiler()* de la classe *Pile*

1. Dans un même package, codez chacune des deux classes dans son propre fichier **.java** correctement nommé. Pensez à ajouter les attributs, constructeurs et autres méthodes qui vous semblent utiles.

2. Dans le même package, implémentez la classe *PileEntiers* en castant le type générique **Object** en **Integer** (attention : pas **int** car c'est un type primitif qui n'hérite pas de **Object**).

3. Soient trois piles d'entiers *P1*, *P2* et *P3*. Dans le même package, codez une nouvelle classe contenant la méthode ***main()*** exécutable. Elle doit créer un objet pour chacune de ces 3 piles d'entiers de *PileEntiers*. Puis remplir la pile *P1* avec 5 entiers aléatoires avec la méthode **Math.random**. En utilisant uniquement ces 3 piles et les méthodes de manipulation de piles implémentées ci-dessus, complétez votre main permettant d'empiler dans *P2* les nombres pairs et dans la pile *P3* les nombres impairs contenus dans *P1*. Le contenu de *P1* après exécution du programme doit être identique à celui avant exécution. Les nombres pairs dans P2 doivent être dans l'ordre où ils apparaissent dans *P1*. Idem pour les nombres impairs dans *P3*. Votre ***main()*** affichera toutes les étapes d'empilement et de dépilement dans les 3 piles.

4. Le problème des tours de Hanoï consiste à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d'«arrivée » en passant par une tour « intermédiaire » et ceci en un minimum de coups, tout en respectant les règles suivantes : on ne peut déplacer plus d'un disque à la fois, on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide. On suppose que cette dernière règle est également respectée dans la configuration de départ.

   4.1) Complétez le diagramme des classes avec une nouvelle classe *Hanoi*.
   4.2) Dans un autre package que les classes précédentes, implémentez la classe *Hanoi*. Le package des classes précédentes doit être complètement indépendant de votre jeu de Hanoi. Le diamètre d'un disque sera représenté par un entier. Chacune des 3 tours sera donc représentée par une pile d'entiers. En utilisant uniquement les 3 piles d'entiers et les méthodes de manipulation de piles implémentées dans la classe *PileEntiers*, écrivez un ***main()*** qui crée un objet pour chacune des 3 piles d'entiers.
   4.3) Une méthode saisira et blindera le nombre *n* de disques. Dans le cas où *n* est négatif ou nul, vous afficherez un message d'erreur.
   4.4) Une méthode remplira la première pile avec *n* entiers (du plus grand au plus petit).
   4.5) Une dernière méthode sera chargée de déplacer les disques vers la troisième pile, en affichant toutes les étapes d'empilement et de dépilement.
   4.6) Votre ***main()*** appellera les méthodes.

Livrable à déposer sur campus

Ce livrable pourra être fait par binôme dans le même groupe de TD, et exceptionnellement un seul trinôme si le groupe de TD est impair. Il sera composé d'un rapport et de tous les dossiers et fichiers de votre code, en citant vos sources.

Ce rapport (doc, docx ou pdf) paginé de 10 pages maximum contiendra les éléments suivants :

- Une page de garde avec vos noms, votre groupe de TD et l'intitulé du TP
- Un sommaire indexé sur les pages
- Le diagramme de classes complet, y compris de la question **4.1**.
- Des jeux d'essais de vos exécutables des questions **3** et **4** sous forme de copies d'écrans à commenter brièvement.
- Une bibliographie des sources consultées sur le web ou/et des livres
- Un bilan et une conclusion personnalisés

JP Segado

Déposez votre livrable dans le lien Rendu du TP2 Java deadline lundi 29 novembre midi. Compresser dans une archive *zip* ou *rar* contenant le rapport et tous les dossiers et fichiers de votre code. Un seul dépôt par binôme (ou trinôme) en indiquant tous les noms de l'équipe (par exemple, *hina-segado.zip*).
Deadline le lundi 23 novembre 23h55

Il y aura une pénalité de -2 points par tranche d'heure de retard lors de la soumission déposée sur campus. Tout code sans les fichiers sources .java vaudra 0. Une analyse de plagiat sera bien entendu effectuée sur tous les codes déposés pour l'ensemble de la promotion. Tout plagiat détecté sera sanctionné par 0 et un avertissement, sans souci de savoir qui a plagié sur qui. N'oubliez pas de citer vos sources : auteurs, liens web, etc.