Université de Paris **Langages à objets avancés**
M1 **Année 2021—2022**

TP Noté n° 2

Ce travail est à réaliser chez vous individuellement et à déposer sur Moodle le dimanche 28/11/2021 à minuit sous forme d'une archive (utiliser impérativement la commande `tar` [1]).

Soyez bien attentifs à ce qui est demandé. Il est précisé que des attributs doivent être explicitement invariants, et que d'autres non. Les contraintes sur les types doivent être clairement assumées. Pensez aux copies indirectes qui sont parfois possibles implicitement etc etc.

Grandes lignes

On s'intéresse à une classe d'objets abstraits, et à deux de ses sous classes, les objets atomiques et les objets composés. Un objet composé n'est qu'une collection d'objets atomiques à laquelle on donne un nouveau nom. Par exemple un "stylo bic" serait le nom d'un objet composé, possédant un exemplaire de "tête en métal", "bille", "cartouche d'encre", "corps hexagonal", "bouchon" ; qui eux sont des objets atomiques.

Nous allons nous placer dans la situation où les ressources en objets atomiques sont limitées, ce qui conduira à une gestion fine des copies, et de leur libération à la destruction (il seront alors recyclés c.à d. reversés à l'ensemble des composants disponibles). Naturellement, un exemplaire d'un objet atomique ne peut être utilisé qu'à un seul endroit à la fois.

Fichiers fournis

Nous vous fournissons une trame : utilisez ces fichiers, et complétez les. Ils sont essentiellement vides, mais ils permettront d'éviter que vous n'utilisiez d'autres noms de classe, et ils nous permettront de localiser plus facilement vos réponses.

`main.cpp` prévoit des test, numérotés de 1 à 9. Les tests 1 puis 5 jusqu'à 9 sont explicitement écrits dans le `main`, alors qu'une instruction vous redirige vers une méthode de `ObjetAbst` où vous trouverez les tests 2,3 et 4.

La validation de ces tests devraient vous permettre de bien comprendre ce qui est attendu. Les choses à faire sont expliquées dans ce sujet, elles sont même ré-expliquées dans les commentaires au niveau des tests. Il ne devrait pas y avoir de malentendus. Le code est souvent à compléter de commentaires explicatifs.

1. `tar cvf mon_fichier.tar fic1 fic2 ...` pour créer l'archive `mon_fichier.tar` composée des fichiers `fic1`, `fic2`, ...

Les classes ObjetAbst et ObjetAtomique

— Les objets abstraits portent un nom définitif, et on leur associe un commentaire. Celui ci est indicatif : il correspond à une information qu'on a voulu noter à un moment donné, il pourra changer, et on l'utilisera parfois pour nos tests.

— Les objets atomiques sont une sous classe d'`ObjetAbst`.

Ce qu'on souhaite, c'est que la liste des objets atomiques concrètement disponibles soit fixée **une fois pour toutes** dans un vecteur statique de la classe `ObjetAbst`.

Dans ce vecteur, chaque objet atomique est associé à un entier, correspondant au nombre de copies qui sont encore disponibles. Vous utiliserez un vecteur de `pair<,>` pour modéliser cette association.

— Dans le squelette qui vous est fourni, vous verrez que nous vous demandons d'initialiser ce vecteur avec 4 objets atomiques "A","B","C","D" ayant respectivement des fréquences 2,25,50,100. Puis il vous faudra vérifier que vous avez bien respecté la contrainte **une fois pour toutes**. Pour cela il vous faudra écrire des tests précis dans `ObjetAbst::testObjetAbst()` (suivez y les consignes).

— On souhaite que les quantités disponibles restent invisibles de l'extérieur, mais on peut retourner un vecteur des noms. Ecrivez une méthode `getAllNames()`

— Puisque la liste des objets atomiques prévue au départ est exhaustive et définitive, personne ne doit pouvoir construire d'objet atomique autrement qu'en en "sortant" un de cette liste. Il faudra donc faire très attention à ce qu'aucun constructeur ne soit appelé par mégarde. En fait, l'utilisation libre d'un constructeur est à proscrire, puisque cela ne permettrait pas de respecter le fait qu'ils sont disponibles en nombre limité. C'est pourquoi nous allons utiliser une `forge` dans `ObjetAbst`. C'est l'objet des tests 5 à 7.

— La destruction d'un `ObjetAtomique` doit restituer la ressource. On vérifie cela dans le test 8.

La classe `ObjetCompose`

— Un objet composé est un objet abstrait qui regroupe en son sein des `objetAtomique`. On le construit en ne lui donnant qu'un nom ; il est vide au départ.

— Pour le nourrir, on choisit une syntaxe particulière, basée sur les `[]`. L'idée est qu'un objet composé, à qui on soumet entre crochet un objet atomique, va l'intégrer dans ses composants. La syntaxe est illustrée dans le test 9. Remarquez que pour esquisser une bonne gestion des ressources, l'objet soumis n'est alors plus disponible, son pointeur passe à null. [2]

— Enfin, on peut s'intéresser à la destruction d'un objet composé. Faites en sorte de libérer la ressource.

2. cette solution est incomplète, mais ce sera suffisant pour terminer cet exercice

Modalités de rendu

Vous devrez nous fournir l'ensemble de vos fichiers de code, qui complètent la trame de l'archive, ainsi qu'un `Makefile` qui marche sans encombres. `make` effectuera la compilation seule et `make test` permettra l'exécution du `main`.

Attention à vous assurer que les `.hpp` soient bien pris en compte dans votre `Makefile`, nous vous rappelons que nous allons commenter/décommenter ici ou là des portions de votre code, make doit donc pouvoir prendre en compte toute modification (ne trichez pas en y faisant un `clean` systématique).