Abstract Tree

```
public class MammouthAdapter extends ITree Sleeper {
        Mammouth m;      // en local ou via params de fonctions

    public void sleepInTree (); {
        m.defoncerArbre();
    }
    // implementer les autres fonctions
    ...

}
```

4.

```
public class TreeSleeperDecorateObservable implements ITreeSleeper, Iobservable{

        private ITreeSleeper sleeper;
        private Observable obs;

    public TreeSleeperDecorateObservable (ITreeSleeper p_ITree, Observable Sujet){
        this.sleeper = p_ITree ; this.obs = Sujet;
            setObservable ( this.obs);
    }

void addObserver ( observer observateur) {
            this.obs.addObserver ( observateur);
    }
void notifyObservers ( object o){
        ...
    }
void setObservable (Observable obs) {
        ---
    }
```

```
void sleepInTree () {
    this.sleeper.sleepInTree();
}
void awakening () {
    this.sleeper.awakening();
}
String getSleepLong () {
    ...                                  }
String getSleepShort () {
    ...                                  }
boolean getSleepState () {
    ...
}
```

Notify

8/5

b) Héritage :
⊕ simple à mettre en place

⊖ perte de l'abstraction "pure" → définition de méthode dans les interfaces ne pouvant être appliquée par l'implémentation abstraite

Strategy :

⊕ possibilité de modifier le comportement d'un observeur / Mammouth

⊖ couplage fort, mais tjs moins que la solution actuelle.

Néanmoins, il n'y a pas d'alternatives à l'observeur, quoiqu'on puisse se passer du décorateur via principes de conception ci-dessus.

2/3

c) Non. En effet, via l'adaptateur, on simule un AbstractTreeSleeper la classe TreeSleeperDecorateObservable prenant en paramètre un ITreeSleeper, c'est donc compatible.

2

d) Dans la mesure où TreeSleeperDecorateObservable implémente l'interface Iobservable je ne vois pas l'intérêt de déléguer.
Néanmoins, dans un contexte "pur" de conception, il s'agit du respect des responsabilités uniques, l'observable étant alors le seul à déléguer

de la partie notification / observateur.

5. Un composite représente une hiérarchie. À moins que les animaux ne soient des poupées russes, je ne vois pas l'intérêt de passer par un Composite pour les stocker. Si néanmoins on fait fi de instateur et que l'on classe les animaux par espèces / sous espèces / familles (via les interfaces par exemple), on pourrait alors songer que le composite n'est pas une mauvaise idée (et en supposant qu'on puisse renvoyer un mammouth, ce qui n'est pas le cas atm)
Au final, la collection d'AbstractTreeSleeper restait le plus simple.

6. Il est étrange de se dire qu'on peut fabriquer des animaux, mais soit. ☺
D'une, une abstract factory permettant de fabriquer des IAnimal sans se soucier du type
Deuxièmement, une factory method permettant d'instancier différentes sous-espèces en leur laissant le soin d'implémenter le propre contenu de leurs méthodes.
Cela serait donc très pertinent si on considère qu'il s'agit d'un zoo ou d'une encyclopédie, et non pas d'un simple animal familial.

7 a) Dans la mesure où TreeSleeperDecoratoObservable implémente la même interface que l'animal, je ne vois aucun intérêt. Cela pourrait être utile uniquement si l'animal a une fonction non-définie dans ladite interface.

b) L'Observable a plus de fonctions que IObservable n'en contient (exemple countObservers()). C'est donc pertinent.

FELICIANO Guillaume 4IRC

Conception Orientée Objet

2. La conception mise en place dont le pattern est le plus proche serait le pattern Template Method. En effet, même si on aurait pu penser à un pattern State de part un changement d'état, sur l'aspect conception, il manque la composition si spécifique à ce pattern. À la place, on a donc un Template Method classique dans lequel on redéfinit au niveau de la classe les fonctions pour un algorithme donné : ici on redéfinit la fonction setSleepLib() et setAwakeLib() dans les classes finales (Opossum & Marsupilami), fonctions étant appelées dans SleepInTree() et awakening() définies dans la classe abstraite mère.

3. L'idée derrière cette question est de faire passer un Mammouth pour un AbstractTreeSleeper. Pour cela, 2 solutions : soit on remonte dans la hiérarchie des classes et on considère que le client instancie des AbstractAnimal (via une factory par exemple), soit on fait enfiler un costume d'opossum à notre Mammouth (merci le carnage) via un Adapter ou un héritage classique ou un Proxy

Façon héritage :

```
public class MammouthVolant extends Mammouth implements
ITreeSleeper {
    // Fonction à implémenter des interfaces
    public MammouthVolant (Mammouth m) { ...
    }
}
```