TD2 : Polymorphisme, interface, stratégie, factory

Ce qui ne varie pas dans le développement logiciel, la seul et unique constante, c'est le changement. L'idée est donc de se préparer au changement.

- mettre de côté ce qui va varier
- limiter les dépendances entre modules pour limiter le nombre de modules à modifier (le plus de modules possible « fermés »)

(ce TD est inspiré de « Head first design pattern »)

Contexte : création d'un jeu dont les acteurs principaux sont des canards.

1 Quel est le problème du code suivant ?

```
class Duck {
    private String species;
    Duck(String species){
        this.species = species;
    }
    void display() { if (species.equal(« mandarin ») { ... } else if ... ;
    void swim() { ... }
}


class Td2 {
    public static void main(String[] args) {
        Duck bob= new Duck(« mandarin »);
        // *** afficher le canard ...
        bob.display();
    }
}
```

2 et maintenant ?

```
class Duck {
    private String species;
    Duck(String species){
        this.species = species;
    }
    abstract void display();
    void swim() { ... }
    void quack() { ... }
}
class MandarinDuck extends Duck {
     MandarinDuck(){
        super(« Mandarin »);
    }
    void display() { ... }
}
class BlackDuck extends Duck {
     BlackDuck(){
        super(« American Black »);
    }
    void display() { ... }
}

class Td2 {
    public static void main(String[] args) {
        Duck bob= new MandarinDuck();
        // *** afficher le canard ...
        bob.display();
    }
}
```

3 On les fait voler ?

En ajoutant une méthode void fly() { ... } dans la classe Duck().

Faites le ...

Et on ajoute une nouvelle classe dérivée pour les canards en caoutchouc : RubberDuck.

Faites le ...

Quel est le problème ?

Comment le résoudre ?

Décrivez les grandes lignes de votre solution.

4 implémenter les comportements (suite 3)

Implémenter les interfaces de comportement Flyable, Quackable

Et les implémentations pour voler ou ne pas voler.

Intégrer ces comportements à vos 3 classes de Canards (penser à la délégation)

5 Rappel : Use Interfaces / super type

« program to an interface, not on implementation »

- pour le comportement des objets :

```
Dog d = new Dog();
d.bark();
```

ou

```
d.makeSound();
```

- pour les références

```
Dog d = new Dog();
```

ou

```
Animal a = new Dog()
```

- pour la création : le type concret utilisé est déterminé à l'exécution

```
Animal a = new Dog()
```

ou

```
Animal a = getAnimal(species);
```

6 Des comportements qui peuvent changer dynamiquement

(pattern stratégie)

Comment pouvoir changer les comportements à l'exécution ?

Modifier vos classes en conséquences.

7 Et la création ...

Comment créer des canards sans être dépendant des types de canards existant ?

8 modules ouverts / fermés

La version de base du jeu est livrée, le CD est gravé.

Le jeu a été prévu pour pouvoir supporter des extensions, vendues séparément par des éditeurs indépendants :

- des nouveaux canards
- des nouvelles manières de voler, de cancaner, ...

Comment ?