

CPOA - Sujet TD 2

PreReq	<ol style="list-style-type: none">1. Je sais programmer en Java.2. J'ai conscience qu'il faut réfléchir avant de se lancer dans le codage.3. Je maîtrise les concepts objet de base (héritage, polymorphisme, ...).4. J'ai compris ce qu'est un patron et j'ai grand soif d'en apprendre d'autres que <i>Strategy</i>
ObjTD	Aborder le patron singleton
Durée	1 TD et 2 TP de 1,5h (sur 2 semaines).

1. La fabrique de chocolat

Vous participez au développement d'un simulateur de fabriques de chocolat modernes dont des bouilleurs sont assistés par ordinateur.

La tâche du bouilleur consiste à contenir un mélange de chocolat et de lait, à le porter à ébullition puis à le transmettre à la phase suivante où il est transformé en plaquettes de chocolat.

1.1. Problème initial

Voici la classe contrôleur du bouilleur industriel de *Bonchoco*, SA.

```
/**
 * @author bruel (taken from Design Pattern - Head First, O'Reilly, 09/2004)
 *
 */
public class BouilleurChocolat {
    private boolean vide;
    private boolean bouilli;

    public BouilleurChocolat() {
        vide = true;
        bouilli = false;
    }

    public void remplir() {
        if (estVide()) {
            vide = false;
            bouilli = false;
            // remplir le bouilleur du mélange lait/chocolat
        }
    }

    public void vider() {
        if (!estVide() && estBouilli()) {
            // vider le mélange
            vide = true;
        }
    }

    public void bouillir() {
        if (!estVide() && !estBouilli()) {
            // porter le contenu à ébullition
            bouilli = true;
        }
    }

    public boolean estVide() { return vide;}

    public boolean estBouilli() { return bouilli;}
}
```



QUESTION

1. À quoi servent les attributs **vide** et **bouilli**?

Vous faites un cauchemar horrible (quoique) où vous vous noyez dans du chocolat. Vous vous réveillez en sursaut avec une crainte terrible.



QUESTION

1. Que pourrait-il se passer avec plusieurs instances de contrôleurs (pour un seul et même bouilleur)?
2. De quoi faudrait-il s'assurer pour éviter ce problème?
3. Trouvez des exemples de situations où il est important de n'avoir qu'une seule instance d'une classe donnée.

1.2. Amélioration 1

Vous vous souvenez des premiers exercices [Java](#) sur les variables de classe et vous proposez d'utiliser un compteur d'instance pour solutionner le problème.

QUESTION

Vous essayez de modifier le constructeur pour qu'il ne fonctionne que si le compteur d'instance est à 0. Qu'est-ce qui ne va pas dans le code suivant :



```
public class BouilleurCptChocolat {
    private boolean vide;
    private boolean bouilli;
    private static int nbInstance = 0;

    public BouilleurCptChocolat() {
        vide = true;
        bouilli = false;
        if (nbInstance == 0) {
            nbInstance = 1;
            return this;
        }
        else {
            return null;
        }
    }
}
```

1.3. Amélioration 2

Vous changez de stratégie car vous vous souvenez avoir déjà vu ce type de code :

Idée!

```
public class MaClasse {
    private MaClasse() {...}
}
```

QUESTION

1. Est-ce autorisé de rendre privé le constructeur?
2. Comment créer une instance dans ces conditions? N'a-t'on pas tout simplement une classe inutilisable?
3. Complétez le code suivant de façon à résoudre le problème :



```
public class BouilleurChocolat {  
    private boolean vide;  
    private boolean bouilli;  
    ...  
    ...  
  
    BouilleurChocolat() {  
        ...  
        ...  
    }  
  
    ...  
    ...  
    ...  
    ...  
  
    public void remplir() {  
        if (estVide()) {  
            vide = false;  
            bouilli = false;  
            // remplir le bouilleur du mélange lait/chocolat }  
        }  
        // reste du code de BouilleurChocolat...  
    }  
}
```

4. Donnez un exemple d'utilisation de cette classe.

1.4. C'est pas fini!

Vos cauchemars continuent! Mais cette fois ils sont en anglais! Vous voyez un grand gaillard irlandais vous menacer (en fait vous confondez *threat* et *thread*...).

QUESTION

1. En quoi les *threads* peuvent-ils poser des problèmes dans votre solution?
2. Recopiez sur des bouts de feuilles les fragments de code ci-dessous en les plaçant dans les colonnes du tableau suivant pour mettre en évidence le problème en reconstituant un enchaînement erroné possible avec deux threads. :



Thread 1	Thread 2	Valeur de <code>uniqueInstance</code>

Bloc 1

```
public static BouilleurChocolat getInstance() {
```

Bloc 2

```
if (uniqueInstance == null) {
```

Bloc 3

```
uniqueInstance = new BouilleurSafeChocolat();
```

Bloc 4

```
}
```

Bloc 5

```
return uniqueInstance;
```

Bloc 6

```
}
```

1.5. Solution au multithreading

Vous vous souvenez heureusement de vos cours de début d'année sur les *threads* :



QUESTION

1. Proposez une solution simple à ce problème.

1.6. Problème de la solution!!

QUESTION



1. Combien de fois le mécanisme mis en place va-t'il être utile ?
2. Que pensez-vous alors de cette solution ?
3. Proposez une solution où l'instance est créé au démarrage plutôt qu'à la demande.



Il peut y avoir des situations où le coût de la synchronisation est inférieur au coût de créer dès le départ une instance (par exemple gourmande en mémoire).

2. Singleton

Félicitations, vous venez de mettre en oeuvre votre deuxième patron, le **Singleton**.

Design pattern : Singleton

Singleton garantit qu'une classe n'a qu'une seule instance et fournit un point d'accès global à cette instance.

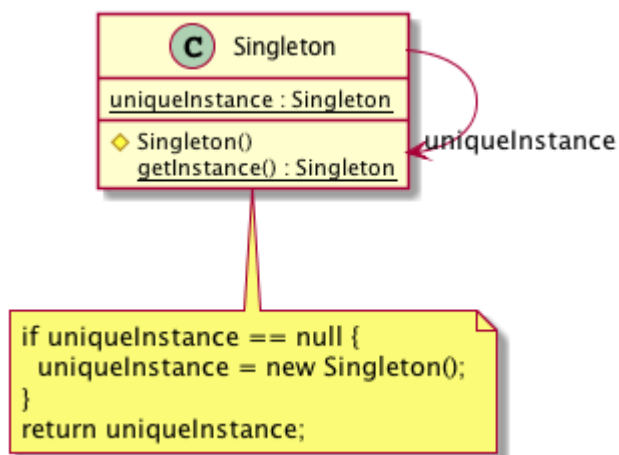


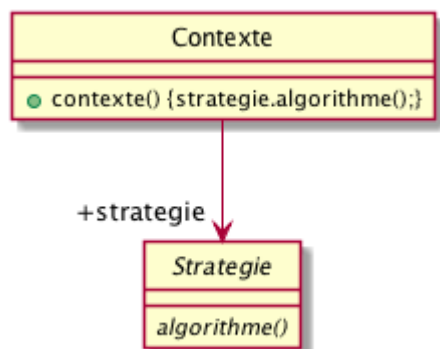
Figure 1. Modèle UML du patron Singleton

3. Le singleton pour le jeu d'aventure

3.1. Combiner plusieurs patrons?

Peut-on combiner les deux derniers patrons vus en TD (*Strategy* et *Singleton*)? En effet, les comportements sont portés par des objets pour l'aspect algorithme, mais il n'y a pas de raison de ne pas les partager entre tous les objets qui "utilisent" ce comportement?!

Dans la plupart des cas ces deux patrons ne vont **pas du tout ensemble**. Cette stratégie n'est recommandée que dans un cas bien précis d'utilisation de *Strategy* : celui où les comportements sont simples et "statiques" (pas de consommation de ressources par exemple) et où l'on utilise une association :



Avec une implémentation du type :

```
...
vol = new VolerAvecDesAiles();
cri = new Cancan();
c1 = new Colvert(vol,cri);
...
```

3.2. Et si on améliorait le jeu d'aventure avec Singleton?



QUESTION

1. Faites en sorte que les instances d'objet affectées à chaque comportement d'un **Personnage** soient uniques pour chaque comportement distinct.
2. Pourquoi ne devrait-on pas utiliser **getInstance()** dans le cas d'une composition (dans le constructeur du composé) ?



On voit que ce n'est pas toujours évident de combiner les patrons entre eux.

Pour aller plus loin



QUESTION

Quelle est la différence entre un singleton et une variable globale?



QUESTION

Comment testeriez-vous la mise en oeuvre du patron **Singleton**?



QUESTION

Il existe une autre façon de gérer le problème du multithreading. Cherchez sur Internet les articles sur le "verrouillage à double vérification" (qui ne fonctionne que depuis Java 1.5).



N'hésitez pas à consulter les liens suivants :

- <http://thecodersbreakfast.net/index.php?post/2008/02/25/26-de-la-bonne-implementation-du-singleton-en-java>
- <http://christophej.developpez.com/tutoriel/java/singleton/multithread/>