

FLYWEIGHT

restitution Design Patterns

MÉRY Andy

QUINTANA Gonzalo

SANTOS Daniel

Advanced C++ programming

29 novembre 2019

Présentation de Flyweight

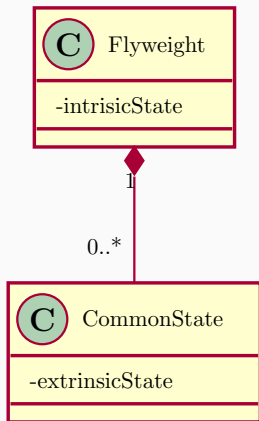


Figure – Diagramme de classe du pattern Flyweight

Situation :

nombreux objets semblables se différenciant seulement sur quelques attributs

Idée :

pointer vers un objet qui regroupe les propriétés extrinsèques

Objectif :

- réduire la complexité en mémoire
- éviter la redondance d'objets

Notre Exemple

- Représentation de soldats dans un jeu de plateau
- Objets Soldats avec une **position** x et y ainsi qu'une string **couleur**

Deux classes de Soldats : une qui implémente le design pattern Flyweight, l'autre non

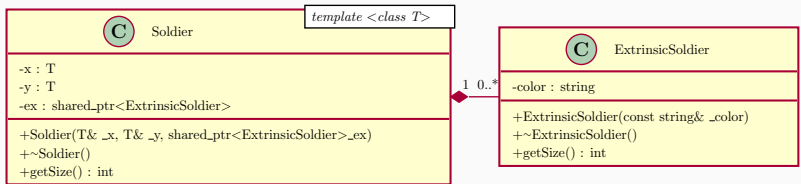


Figure – Diagramme de classes utilisant le modèle FlyWeight pour la représentation du soldat

Notre implémentation

Overview :

- création des instances de soldats
- mesure de la durée de la création
- calcul de la taille allouée en mémoire pour chaque instance

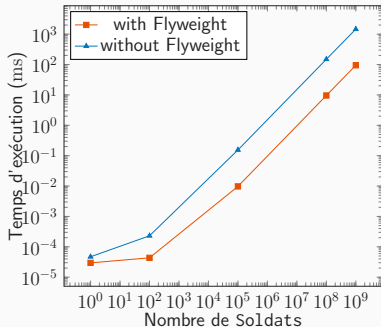
*Try me on GitHub!*¹

```
1 class ExtrinsicSoldier {
2     private :
3         string color; // The extrinsic
4         property.
5     public :
6         ExtrinsicSoldier( const string &
7             _color ) : color(_color){}
8         ~ExtrinsicSoldier(){}
9 };
10
11 template <class T>
12 class Soldier
13 {
14     private :
15         T x;
16         T y;
17         shared_ptr<ExtrinsicSoldier> ex;
18     public :
19         Soldier( T &x, T &y, shared_ptr<
20             ExtrinsicSoldier> _ex){
21             this->x = _x;
22             this->y = _y;
23             this->ex = _ex;
24         }
25         ~Soldier(){}
26 };
27
```

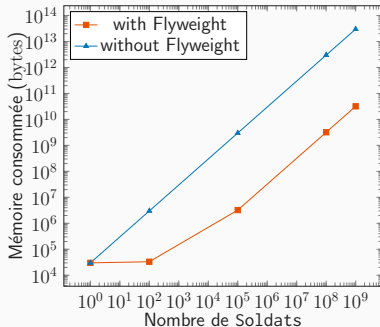
1. Test it on <https://github.com/xdanielsb/ProfilerFlyweight>

Résultats et conclusions

Comparaison de temps d'exécution



Comparaison de la Utilisation de mémoire



- Forte réduction de l'utilisation de la mémoire
- Gain en temps d'exécution