FLYWEIGHT
restitution Design Patterns

MÉRY Andy
QUINTANA Gonzalo
SANTOS Daniel

Advanced C++ programming

3 décembre 2019

Présentation de Flyweight

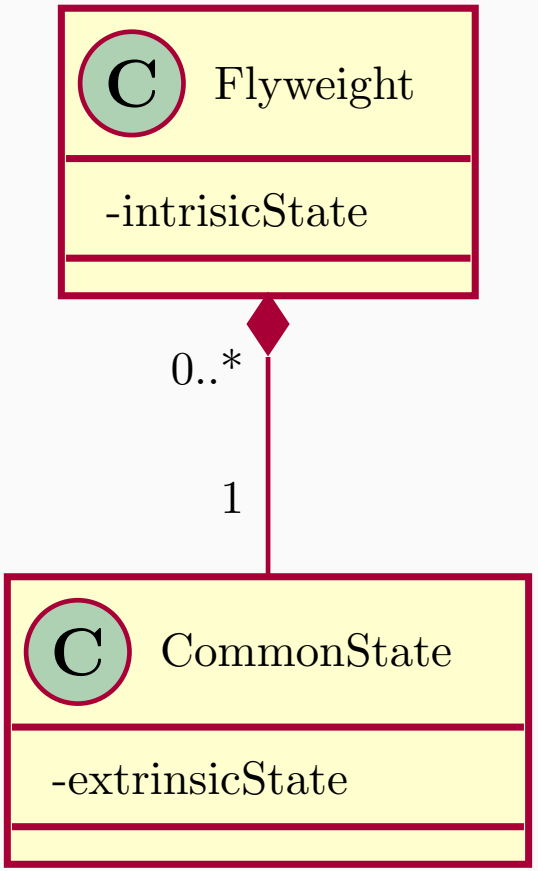


Figure – Diagramme de classe du pattern Flyweight

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

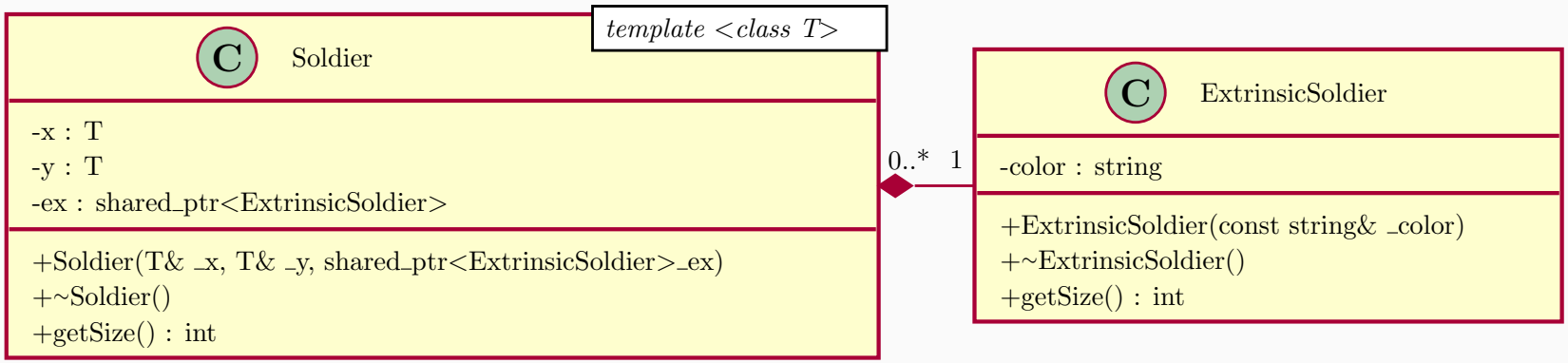


Figure – Diagramme de classes utilisant le modèle FlyWeight pour la représentation du soldat

Notre implémentation

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

Overview :

→ création des instances de soldats

→ mesure de la durée de la création

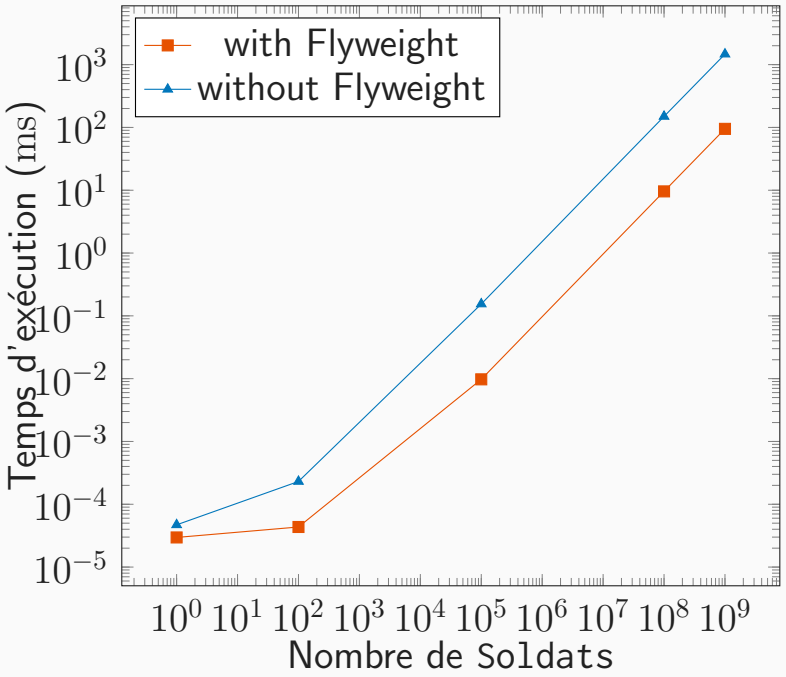
→ calcul de la taille allouée en mémoire pour chaque instance

Try me on GitHub ! [1]

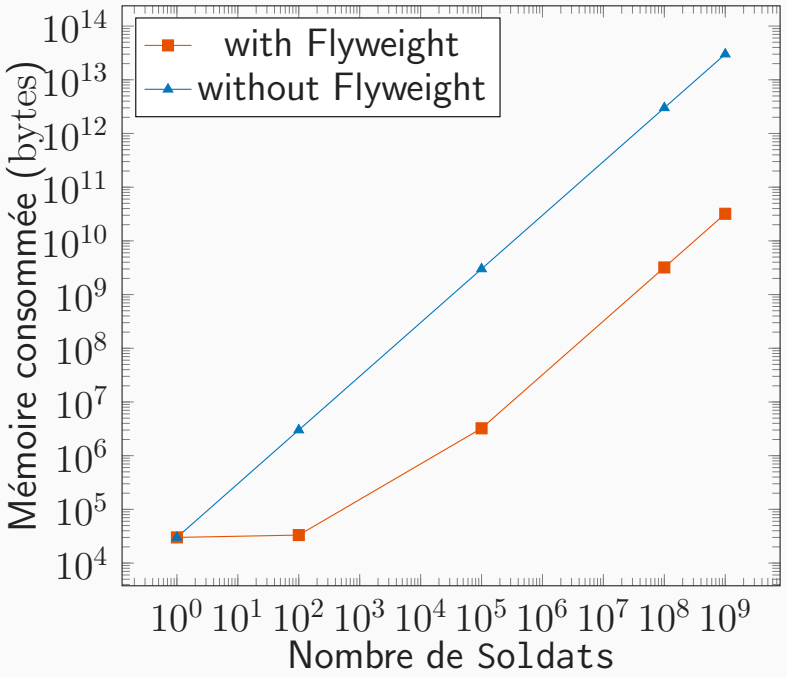
1. Test it on https ://github.com/xdanielsb/ProfilerFlyweight

Résultats et conclusions

Comparaison de temps d'exécution



Comparaison de l'utilisation de la mémoire



→ Forte réduction de l'utilisation de la mémoire

→ Gain en temps d'exécution