DESIGN PATTERN FLYWEIGHT

Group 6:
Lê Duy Bách
Đào Thanh Danh
Nguyễn Khắc Tuấn

I. Introduction

- Flyweight is one of structural patterns.
- The concept of flyweight is saving space by reuse the object shared same property.
- In the flyweight pattern, there is the concept of Intrinsic and Extrinsic state

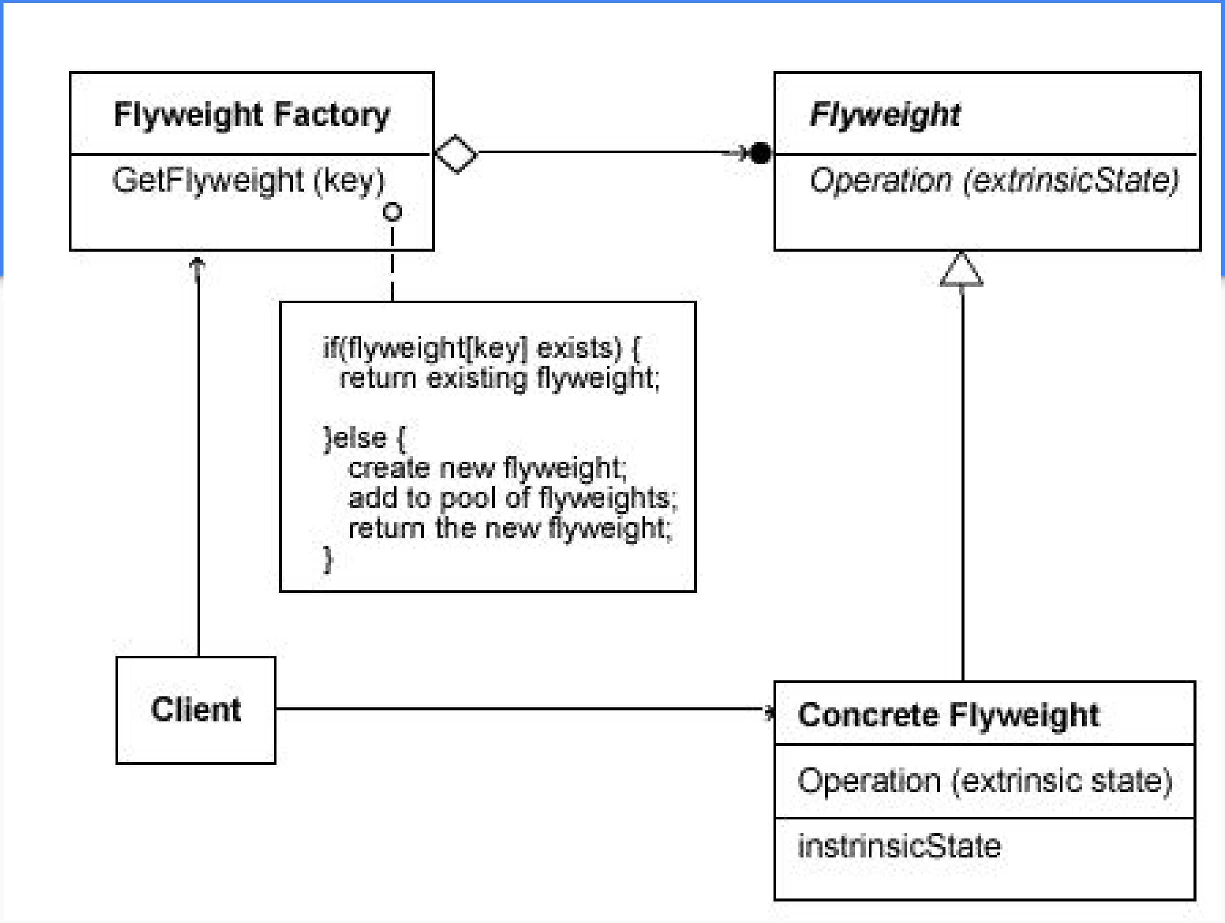
II. Intrinsic and Extrinsic state

1) Intrinsic states are things that are constant and are stored in the memory
2) Extrinsic states are things that are not constant and need to be calculated, therefore not stored in the memory and can be passed in through arguments

III. Diagram

Flyweight Factory handles create flyweight objects.

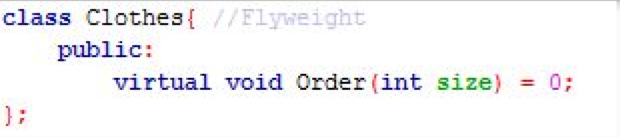
Concrete Flyweight inherited from flyweight (like usually we do)



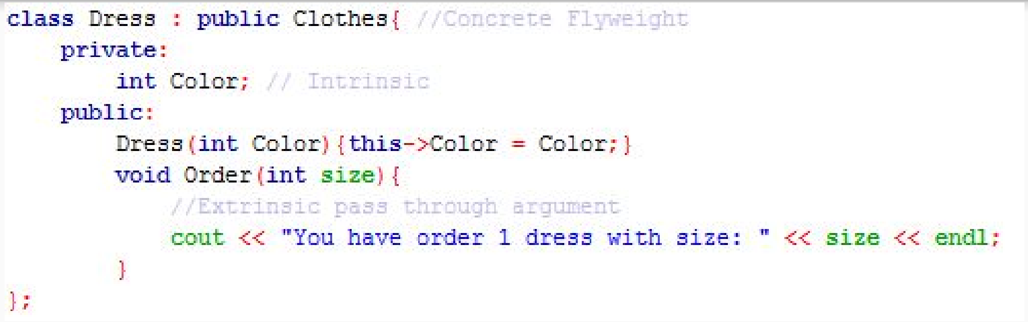
```
if(flyweight[key] exists) {
  return existing flyweight;

}else {
   create new flyweight;
   add to pool of flyweights;
   return the new flyweight;
}
```

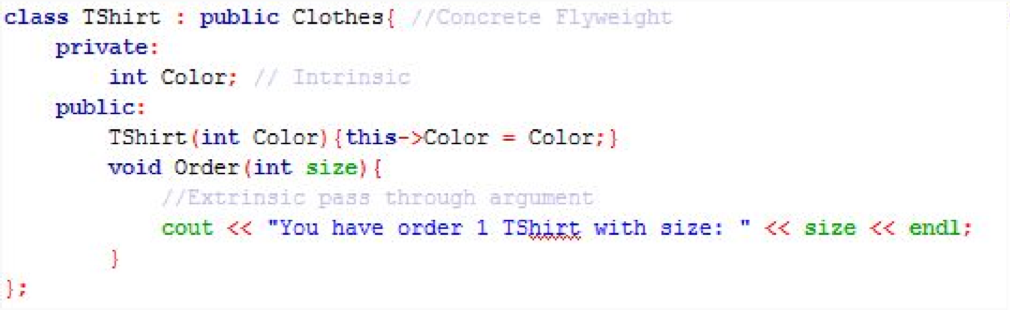
IV. Example



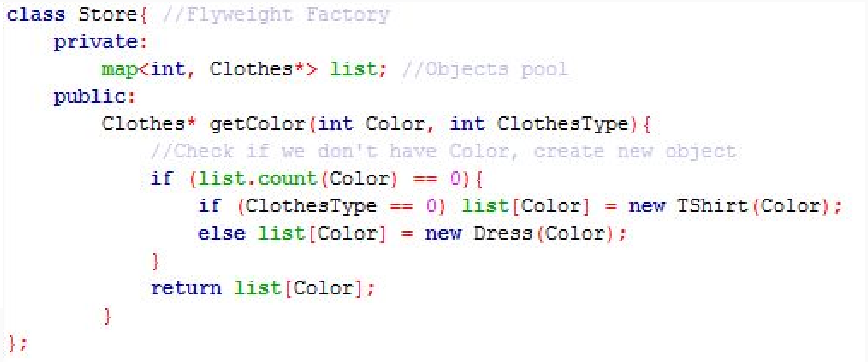
```
class Clothes{ //Flyweight
    public:
        virtual void Order(int size) = 0;
};
```



```
class Dress : public Clothes{ //Concrete Flyweight
    private:
        int Color; // Intrinsic
    public:
        Dress(int Color){this->Color = Color;}
        void Order(int size){
            //Extrinsic pass through argument
            cout << "You have order 1 dress with size: " << size << endl;
        }
};
```



```
class TShirt : public Clothes{ //Concrete Flyweight
    private:
        int Color; // Intrinsic
    public:
        TShirt(int Color){this->Color = Color;}
        void Order(int size){
            //Extrinsic pass through argument
            cout << "You have order 1 TShirt with size: " << size << endl;
        }
};
```



```
class Store{ //Flyweight Factory
    private:
        map<int, Clothes*> list; //Objects pool
    public:
        Clothes* getColor(int Color, int ClothesType){
            //Check if we don't have Color, create new object
            if (list.count(Color) == 0){
                if (ClothesType == 0) list[Color] = new TShirt(Color);
                else list[Color] = new Dress(Color);
            }
            return list[Color];
        }
};
```

V. Summary

- .A factory object is needed to control creates objects
- limit the number of instances created
- Reuse previous object if a similar object is needed later
- can improve performance and reduce needed resources significantly
- make your code more complicated, harder to debug, and harder to maintain

VI. References

https://dzone.com/articles/design-patterns-flyweight

http://www.televis.at/assets/files/Dokumente/flyweight%20pattern.pdf

https://ideone.com/XPJTIE