Mittuniversitetet
MID SWEDEN UNIVERSITY

Lab 8 – ”Factories”

Designmönster med C++

Syfte: Användning av Factory-objekt.

Lab 8 - Factories

I den här laborationen ska du återvända till två av de föregående inlämningsuppgifterna och förbättra lösningarna genom att använda två designmönster som GoF har kategoriserat som ’creational’.

Uppgift 1

I lab 4 tillämpade du DP Abstract Factory för att instansiera rätt uppsättning objekt för varje typ av spel. En nackdel med den lösning som blev resultatet är att ett nytt factoryobjekt instansieras varje gång ett spel ska spelas. Du ska lösa problemet genom att modifera de båda konkreta GameFactory-klasserna så de blir *Singletons*. Gör de anpassningar som behövs av den övriga koden.

Uppgift 2

Om du tittar på din lösning av laboration 7, ”Den dekorerade kaffeautomaten” så kommer du (förmodligen...) att kunna konstatera att många klasser är i stort sett identiska. Alla konkreta Components (Coffee, osv) skiljer sig bara åt genom värden på instansvariabler. Samma sak gäller för alla konkreta Decorators

Den kod som bygger upp kedjan av Components beroende på användarens val måste känna till exakta typer, i stil med

```
Component comp = nullptr;
...
switch (drinkChoice) {
      case 1: comp = new Coffee; break;
      ...
switch (extraChoice) {
      case 1: comp = new Sugar(comp); break;
      ...
```

Du ska använda DP Prototype för att ändra på detta.
Egenskaper hos Prototype som är intressanta i detta sammanhang är

- Antalet klasser kan reduceras när klasserna bara skiljer sig åt ifråga om värden på instansvaraiabler
- Klienter behöver inte veta exakt typ för de objekt som skapas

Prototype kallas ibland ‘Virtual constructor’. En klient använder en Prototype genom operationen `clone` som returnerar en kopia av det aktuella objektet.
Du ska själv söka den information du behöver om DP Prototype. Om du ’googlar’ på Prototype Design Pattern så hittar du mer än du orkar läsa...

Riktlinjer och tips för lösningen

- Alla konkreta Components (dryckerna: Coffee, osv) ska ersättas av EN klass, DrinkPrototype.
- Alla konkreta Decorators (Sugar osv) ska ersättas av EN klass DecoratorPrototype

Båda dessa klasser ska ha en metod `clone` som returnerar en kopia av objektet. Attributvärden ska kunna sättas genom metoden `initialize` som tar de aktuella värdena som argument: namnet, priset, och för DecoratorPrototype, en pekare till nästa komponent.

Klassen PrototypeManager (som ska vara en Singleton) skapar EN instans av vardera Prototype-klass. Klientkoden använder PrototypeManager för att få objekt av 'rätt sort'. Detta görs med två publika metoder hos PrototypeManager, `getDrink` och `getDecorator`, med lämpliga argument.

Vid anrop till en get-funktion i PrototypeManager:
- ett nytt objekt skapas genom att 'klona' något av prototyp-objekten
- rätt attributvärden sätts på det nya objektet genom metoden initialize
- objektet returneras till klienten

Kodensnutten här ovan skulle med denna lösning bli

```
Component comp = null;
...
switch (drinkChoice) {
      case 1:
            comp =
            PrototypeManager::instance()->getDrink("Coffee");
            break;
      ...
switch (extraChoice) {
      case 1:
            comp =
            PrototypeManager::instance()->getDecorator(comp, "Sugar");
            break;
      ...
```

Klientkoden behöver alltså inte veta exakt vilka objekt den behöver. Andra klasser i din gamla lösning kan behöva förändras när det gäller placering av instansvariabler och val av accessnivåer. Alla objekt som skapas dynamiskt ska också deallokeras.

Redovisning

Kompletta lösningar till uppgifterna zippas ihop i **separata** zipfiler.