Ministerul Educației al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatica și Micorelectronică

Catedra Automatica și Tehnologii Informaționale

Raport

Lucrarea de laborator nr. 1

La disciplina**:** Ingineria Produselor Program

Tema: Șabloane creaționale

A efectuat: st.gr.TI-143 Cornita Constantin

A verifica: lector asistent Chetrusca Ecaterina

Chișinău 2017

Scopul și sarcina

De studiat și de implimentat șabloane creationale Abstract Factory, Singleton, Prototype, Builder, Factory Method.

Noțiuni Teoretice

Șabloanele creaționale de proiectare, abstractizeaza procesul de instanțiere. Ele ajuta de a face un system independent de modul in care sunt create compuse si reprezentate obiectele acestuia. Un șablon creațional de clasa folosește moștenirea pentru a varia clasa care este instantiată, in timp ce un șablon creational de obiect va delega instanțierea către un alt obiect.

Șabloanele creaționale devin din ce în ce mai importante, pe măsura ce sistemele evolueaza spre a depinde mai mult de compunerea obiectelor decat de moștenirea de clasă. Prin urmare, crearea obiectelor cu un anumit comportament va necesita mai mult decăt simpla instțantiere a unei clase.

Abstract Factory

Abstract Factory furnizează un nivel de abstractizare în crearea de familii de obiecte înrudite sau dependente fără a specifica direct clasele lor concrete. Obiectul „fabrică“ are responsabilitatea de a oferi servicii pentru crearea unei întregi familii de platforme. Clienții nu vor crea niciodată platforme în mod direct, ci vor apela la „fabrică“ pentru a face acest lucru.

Acest mecanism face schimbarea familiilor de produse mai simplă deoarece clasa concretă de tip factory apare o singură dată în aplicație, atunci când este instanțiată. Aplicația poate înlocui întreaga familie de produse prin simpla instanțiere a unei alte clase concrete de tipul abstract factory.

Patternul Abstract Factory va defini o metodă Factory la nivel de produs. Fiecare astfel de metodă va încapsula operatorul new precum și implementarea concretă a produsului. Fiecare „platformă“ este apoi modelată folosind o clasă de tip Factory, derivată.

Abstract Factory are urmatoarele avanaje:

- utilizeaza clase concrete
- simplifica schimbul famiilor de produse
- asigura compatibiliatea produselor

Dezavantaje:

- este dificil de a aduga suport pentru noi tipuri de produse

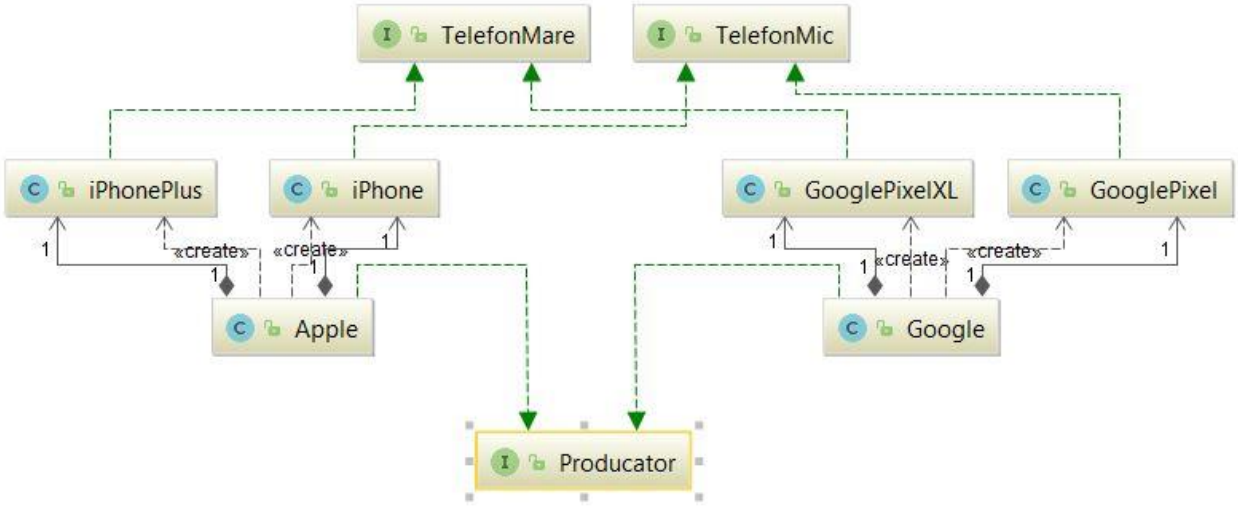

Fig.1 Abstract Factory

Factory Method

Șablonul definește o interfața pentru crearea unui obiect, dar permite subclaselor să decidă ce clasa vor instanția. De asemenea acest șablon permite unei clase sa delege instanțierea la alte subclase.

Cadrele de dezvoltare utilizeaza clase abstracte pentru a define si intreține relațiile între obiecte. Deseori, un cadru de dezvoltare este de asemenea responsabil și de crearea acestor obiecte.

Șablonul Factory Method incapsulează cunoașterea tipului de subclasa Document care va fi creat și scoate aceasta cunoaștere din cadrul de dezvoltare. Subclasele Application redefines operatia abstractă CreateDocument a clasei Application, astfel incat aceasta sa returneze subclasa Document corecta.

Șablonul Factory Method se utilizeaza cînd:

- O clasă nu poate anticipa clasa obiectelor pe care trebuie să le creeze
- O clasa dorește ca subclasele sale sa precizeze obiectele pe care le creeaza
- Clasele deleaga responsabilitatea către una dintre mai multe subclase ajutatoare, de asemenea duce contul subclasei ajutatoare delegate

Avanataje:

- Permite crearea codului mult mai universal, adică obiectul să nu fie legat de anumite clase, si care functioneaza cu o interfaîa comună.

Dejavantaje:

— Creatorul trebuie să creeze un nou moștenitor pentru fiecare tip de produs (ConcreteProduct). Cu toate acestea, limbaje de programare moderne permit relizarea șablonui Fabrica fară ierarhizare claselor Creator.

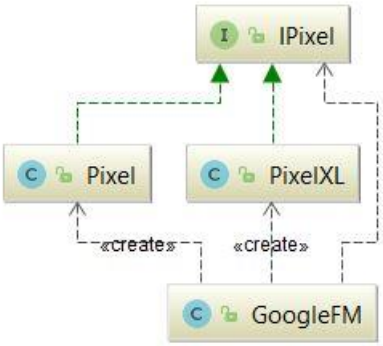

Fig.2 Factory Method

Singleton

În inginerie software, modelul Singleton este un model de design care restricționează instanțierea unei clase la un singur obiect. Acest lucru este util atunci când este nevoie de exact un obiect pentru a coordona acțiunile în cadrul sistemului. Conceptul este uneori generalizat la sisteme care funcționează mai eficient atunci când există un singur obiect, sau care limitează instanțierea unui anumit număr de obiecte. Sunt unii care sunt critici fața de model Singleton și consideră că acesta este un anti-model, care este frecvent utilizat în scenarii în cazul în care aceasta nu este benefic, introduce restricții inutile în situații în care nu este necesară, de fapt o instanță exclusivă a unei clase, și introduce starea globală într-o aplicație.

O implementare a modeluli Singleton trebuie sa ne asigure cu:

— Să asigure că o instanța a clasei sinleton exista și oferă acces global la acea instanță.

Acest lucru se realizează prin: constructorii sa fie private, oferind o metoda statică care returneaza o referința la instanta.

Instanța este stocată, de obicei ca o variabilă statică privată; instanța este creată atunci când variabila este inițializat, la un moment dat, înainte de metoda statică este solicitată. Ceea ce urmează este o implementare de a șablonului singleton în Java. În acest exemplu, initializatorul statică este rulat atunci când clasa este inițializat, după clasa de încărcare, dar înainte de clasa este utilizat de orice tip de fir.

Builder Pattern

Șablonul Builder determină un algoritm care separă faza de proiectare a produsului complex, (obiect) al reprezentării sale externe, astfel încât, prin intermediul aceluiași algoritm poate se obține puncte de vedere diferite ale produsului. Pentru acesta, șablonul Builder determină un algoritm de crearea treptată a unui produs într-o clasă specială de director (manager) și responsabil pentru coordonarea procesului de asamblare a părților individuale ale produsului impune ierarhiei clasei Builder. În această ierarhie, in șablonul Builder, clasa de bază anunță interfețe pentru construirea părților ale produsului și subclasele relevante ConcreteBuilder le pune în aplicare în mod adecvat, de exemplu, să creeze sau să obțină resursele necesare pentru a menține rezultatele intermediare, de a monitoriza rezultatele operațiunilor.

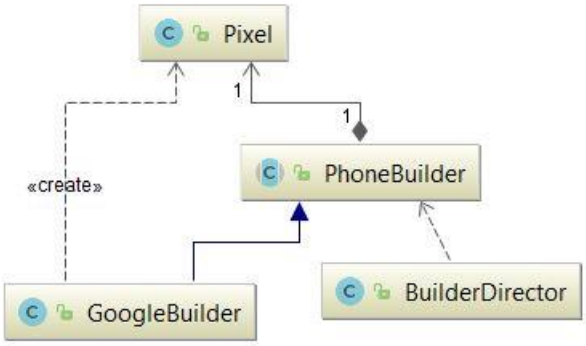

Fig.3 Builder Pattern

Șablonul Prototype

Acest șablon creaza obiectele clonînd un obiect existent. Șablonul acesta poate fi utilizat atunci cind sistemul ar trebui sa fie independent de modul in care sunt create produsele sale.

Șablonul Prototype se utilieaza în urmatoarele cazuri:

- Atunci cînd un anumit tip de obiecte care urmează a fi create, clasele care desfinesc aceste obiecte sunt create in mod dinamic.
- Atunci cind se dorește creare unei ierahii de clase separată de clasele factory.
- În cazul cînd clonarea unui obiect este opțiunea cea mai perfectă, mai degraba chiar și de cît constuctorul. De asemena cînd obiectul poate accepta un număr mic de stări posibile.

Concluzie

În acestă lucrare de laborator am studiat șabloanele de proiectare. Fiecare șablon descrie o problemă care apare in domeniul de acivitate și indică soluția acelei probleme intr-un mod care permite reutilizarea acestei soluții de nenumarate ori în contexte diferite. Un șablon reprezintă o solutie comuna a unei probleme într-un anumit context. Un șablon nu este o clasa sau o librărie, el este un model care trebuie impelmenta în situația corectă. De asemena șabloanele nu sunt specifiie unui anumit limbaj.

Anexa

Repozitoriu distant: https://github.com/CornitaConstantin/Lab1IPP