**ДВНЗ**

**ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**

**ЗВІТ ДО ЛАБОРАТОРНОЇ РОБОТИ №3**

з дисципліни «Методи інженерії комп’ютерних систем»

Виконав:

Студент гр. КІ-20

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Глущенко Д.С.

(дата, підпис)

Перевірив(ла):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Любимов А.С.

(дата, підпис) ( ПІБ)

Луцьк – 2023

**Тема: Структурні патерни**

**Мета: ознайомлення із призначенням, сферою застосування, структурою та особливостями реалізації структурних патернів.**

**Завдання:**

1. Composite

Конструктор меблів

Ви розробляєте веб-додаток для виробника корпусних шаф. У додатку користувач може «зібрати» власний варіант з певних елементів. Для додавання доступні елементи різного рівня складності – як елемент «ящик», так й «тумба» (складається з декількох ящиків) або «стійка» (тумба + кілька елементів «полка»). Кожен елемент має свою вартість, яка складається з ціни матеріалу і вартості виготовлення. Реалізуйте такий конструктор шаф за допомогою патерну «Компонувальник», причому для кожного елемента друкуйте його ціну, а в кінці підрахуйте підсумкову вартість всього шафи.

1. Proxy

Дані з метеостанції

Метеостанція відправляє дані з датчиків (температура, вологість, вітер) на сервер кожні 30 секунд. Дані приймаються сервером і записуються в БД. Згодом виявилося, що подібна періодичність є надмірною, і просто безцільно збільшує розмір БД, проте змінити налаштування на метеостанції не є можливим. За цих умов було прийнято рішення розробити проксі-клас, який приймає від метеостанції 10 (кількість налаштовується) повідомлень, розраховує середнє для кожного показника і одного разу записує їх в БД. Реалізуйте необхідну систему класів за допомогою патерну «Проксі» та продемонструйте їх використання.

**Хід роботи:**

Компонувальник — це структурний патерн проектування, що дає змогу згрупувати декілька об’єктів у деревоподібну структуру, а потім працювати з нею так, ніби це одиничний об’єкт.

Патерн Компонувальник має сенс тільки в тих випадках, коли основна модель вашої програми може бути структурована у вигляді дерева.

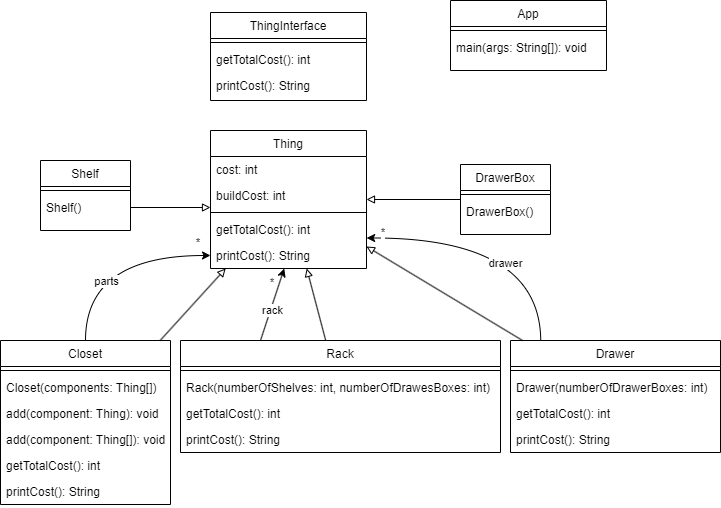


Рисунок 1 – UML-діаграма компонувальника за варіантом

**Лістинг програми:**

Зміст App:

**package** main;

**public** **class** App {

**public** **static** **void** main(String[] args) {

Closet a = **new** Closet(**new** Drawer(4));

a.add(**new** DrawerBox());

System.***out***.print(a.printCost());

System.***out***.print("\nTotal cost = "+a.getTotalCost()+"\n\n");

a.add(**new** Rack(2,2));

System.***out***.print(a.printCost());

System.***out***.print("\nTotal cost = "+a.getTotalCost()+"\n\n");

}

}

Зміст ThingInterface:

**package** main;

**public** **interface** ThingInterface {

**int** getTotalCost();

String printCost();

}

Зміст Thing:

**package** main;

**public** **class** Thing **implements** ThingInterface {

**int** cost;

**int** buildCost;

@Override

**public** **int** getTotalCost() {

**return** buildCost + cost;

}

@Override

**public** String printCost() {

**return** **this**.getClass().getSimpleName() + ": Part cost = " + cost + " Build cost = " + buildCost + "\n";

}

}

Зміст Closet:

**package** main;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** Closet **extends** Thing{ //шафа

**protected** List<Thing> parts = **new** ArrayList<>();

**public** Closet(Thing... components) {

add(components);

}

**public** **void** add(Thing component) {

parts.add(component);

}

**public** **void** add(Thing... components) {

parts.addAll(Arrays.*asList*(components));

}

@Override

**public** **int** getTotalCost() {

**int** a=0;

**for**(**int** i =0; i<parts.size();i++) {

a+=parts.get(i).getTotalCost();

}

**return** a;

}

@Override

**public** String printCost() {

String a = "";

**for**(**int** i =0; i<parts.size();i++) {

a+=parts.get(i).printCost();

}

**return** a;

}

}

Зміст Drawer:

**package** main;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** Drawer **extends** Thing { //тубма

**protected** List<Thing> drawer = **new** ArrayList<>();

**public** Drawer(**int** numberOfDrawerBoxes) {

**super**.cost=15;

**super**.buildCost=40;

**for**(**int** i = 0; i< numberOfDrawerBoxes; i++) {

drawer.add(**new** DrawerBox());

}

}

@Override

**public** **int** getTotalCost() {

**int** a=0;

**for**(**int** i = 0; i< drawer.size(); i++) {

a+=drawer.get(i).getTotalCost();

}

**return** **super**.getTotalCost()+a;

}

@Override

**public** String printCost() {

String a="";

**for**(**int** i = 0; i< drawer.size(); i++) {

a+="\n" + drawer.get(i).printCost();

}

a = a.replace("\n", "\n\t");

**return** **super**.printCost() + a + "\n";

}

}

Зміст DrawerBox:

**package** main;

**public** **class** DrawerBox **extends** Thing { //ящик

DrawerBox(){

**super**.cost=40;

**super**.buildCost=10;

}

}

Зміст Rack:

**package** main;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** Rack **extends** Thing {

**protected** List<Thing> rack = **new** ArrayList<>();

**public** Rack(**int** numberOfShelves, **int** numberOfDrawerBoxes) {

**super**.cost=35;

**super**.buildCost=45;

rack.add(**new** Drawer(numberOfDrawerBoxes));

**for**(**int** i = 0; i< numberOfShelves; i++) {

rack.add(**new** Shelf());

}

}

@Override

**public** **int** getTotalCost() {

**int** a=0;

**for**(**int** i = 0; i< rack.size(); i++) {

a+=rack.get(i).getTotalCost();

}

**return** **super**.getTotalCost()+a;

}

@Override

**public** String printCost() {

String a="";

**for**(**int** i = 0; i< rack.size(); i++) {

a+="\n" + rack.get(i).printCost();

}

a = a.replace("\n", "\n\t");

**return** **super**.printCost() + a + "\n";

}

}

Зміст Shelf:

**package** main;

**public** **class** Shelf **extends** Thing {//полка

**public** Shelf() {

**super**.cost=10;

**super**.buildCost=0;

}

}

**Приклад виконання коду з Main:**

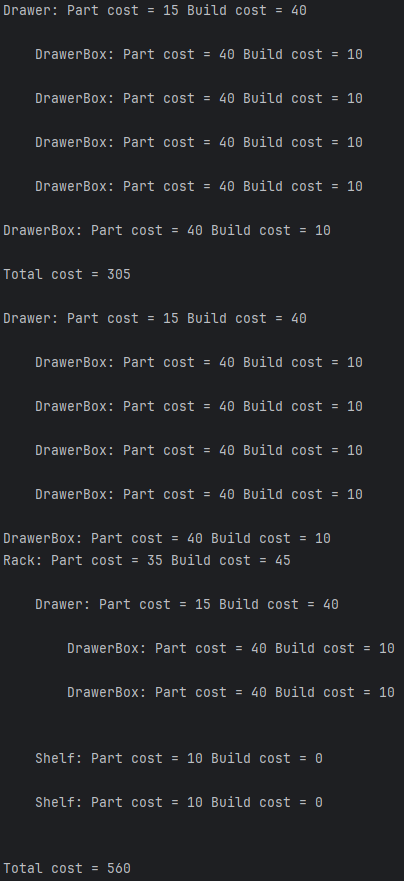


Рисунок 2 – Результат роботи абстрактної фабрики

Замісник (Proxy) — це структурний патерн проектування, що дає змогу підставляти замість реальних об’єктів спеціальні об’єкти-замінники. Ці об’єкти перехоплюють виклики до оригінального об’єкта, дозволяючи зробити щось до чи після передачі виклику оригіналові.

Цей патерн дозволяє контролювати сервісний об’єкт непомітно для клієнта, може працювати, навіть якщо сервісний об’єкт ще не створено, може контролювати життєвий цикл службового об’єкта.

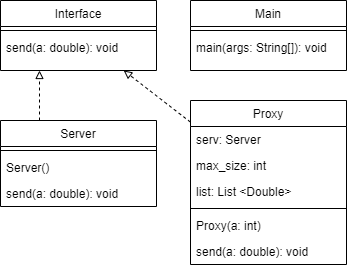


Рисунок 3 – UML-діаграма замісника за варіантом

**Лістинг програми:**

Зміст Interface:

**package** main;

**public** **interface** Interface {

**public** **void** send(**double** a);

}

Зміст Main:

**package** main;

**import** java.util.Calendar;

**import** java.util.concurrent.ThreadLocalRandom;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**int** timeToWait=1;

**int** maxNumber=3;

Interface proxy = **new** Proxy(maxNumber);

**while**(**true**) {

**try** {

Thread.*sleep*(timeToWait\*1000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**double** a = ThreadLocalRandom.*current*().nextDouble(-5, 10);

System.***out***.print("Sent "+ a +" at time "+Calendar.***HOUR\_OF\_DAY***+":"+Calendar.*getInstance*().getTime().~~getMinutes~~()+":"+Calendar.*getInstance*().getTime().~~getSeconds~~()+"\n");

proxy.send(a);

}

}

}

Зміст Proxy:

**package** main;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** Proxy **implements** Interface{

**private** **static** Server *serv*;

**int** max\_size = 10;

List<Double> list = **new** ArrayList<Double>();

**public** Proxy(**int** a) {

max\_size = a;

**if** (*serv*==**null**)

*serv* = **new** Server();

}

**public** **void** send(**double** a) {

list.add(a);

**if**(list.size()==max\_size) {

**double** temp=0;

**for**(**int** i = 0 ; i<list.size();i++) {

temp+=list.get(i);

}

temp/=list.size();

*serv*.send(temp);

list.clear();

}

}

}

Зміст Server:

**package** main;

**public** **class** Server **implements** Interface{

**public** Server() {

}

**public** **void** send(**double** a) {

System.***out***.print("Received "+a+"\n");

}

}

**Приклад виконання коду з Main:**

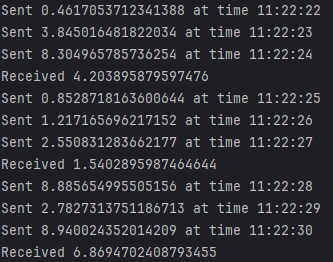


Рисунок 4 – Результат роботи замісника

**Висновок:** був **ознайомлений із призначенням, сферою застосування, структурою та особливостями реалізації структурних патернів.**

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Компонувальник [Електорний ресурс]–Режим доступу: https://refactoring.guru/uk/design-patterns/composite
2. Замісник [Електорний ресурс]–Режим доступу: https://refactoring.guru/uk/design-patterns/proxy