**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ**

**“ЛЬВІВСЬКА ПОЛІТЕХНІКА”**

**Кафедра систем штучного інтелекту**



**Звіт**

з дисципліни:

**«Системний аналіз»**

про проходження курсу:

**«Design Patterns»**

Виконав:

студент групи КН-209

Ярчак А.В

Перевірив:

Шамуратов О. Ю.

**Львів – 2020р**

**Предметна область**

|  |  |
| --- | --- |
| 29 | Оплата договорів |

**ФАБРИЧНИЙ МЕТОД**

Я реалізував цей патерн тому що він визначає загальний інтерфейс для створення об’єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об’єктів, адже підкласи успадковують код абстрактного класу фабрики. У моєму випадку це дозволяє створювати контракт різних видів при цьому міняючи лише метод створення договорів.

*Структура:*

Contract.java

package com.factory.contracts;  
  
public interface Contract {  
  
 String getName();  
 void setName(String name);  
 int getPrice();  
 void setPrice(int price);  
 String getDate();  
 void setDate(String date);  
 String getSign();  
 void setSign(String sign);  
 String getInfo();  
 void setInfo(String info);  
}

ContractOfSale.java

package com.factory.contracts;  
  
public class ContractOfSale implements Contract {  
  
 private String name="Sale house";  
 private int price=2000;  
 private String date="2020-05-06";  
 private String sign="OOO sign";  
 private String info="Some info about location, characters etc";  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getPrice() {  
 return price;  
 }  
  
 public void setPrice(int price) {  
 this.price = price;  
 }  
  
 public String getDate() {  
 return date;  
 }  
  
 public void setDate(String date) {  
 this.date = date;  
 }  
  
 public String getSign() {  
 return sign;  
 }  
  
 public void setSign(String sign) {  
 this.sign = sign;  
 }  
  
 public String getInfo() {  
 return info;  
 }  
  
 public void setInfo(String info) {  
 this.info = info;  
 }  
  
 @Override  
 public String toString() {  
 return "I am a Contract of sale and that's my info:\n" + getName()+'\n'+getInfo()+'\n'+getPrice()+"$\n"+getDate()+'\n'+getSign();  
 }  
}

SupplyContract.java

package com.factory.contracts;  
  
public class SupplyContract implements Contract{  
  
 private String name="Arms supply contracts";  
 private int price=10000;  
 private String date="2020-06-06";  
 private String sign="Censored";  
 private String info="Guns, rockets, mines, bullets";  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getPrice() {  
 return price;  
 }  
  
 public void setPrice(int price) {  
 this.price = price;  
 }  
  
 public String getDate() {  
 return date;  
 }  
  
 public void setDate(String date) {  
 this.date = date;  
 }  
  
 public String getSign() {  
 return sign;  
 }  
  
 public void setSign(String sign) {  
 this.sign = sign;  
 }  
  
 public String getInfo() {  
 return info;  
 }  
  
 public void setInfo(String info) {  
 this.info = info;  
 }  
  
 @Override  
 public String toString() {  
 return "I am supply Supply contract and that's my info:\n" + getName()+'\n'+"Supplies: "+getInfo()+'\n'+getPrice()+"$\n"+getDate()+'\n'+getSign();  
 }  
}

Factory.java

package com.factory.factory;  
  
import com.factory.contracts.Contract;  
  
public abstract class Factory {  
  
 public abstract Contract createContract();  
  
}

SaleFactory.java

package com.factory.factory;  
  
import com.factory.contracts.Contract;  
import com.factory.contracts.ContractOfSale;  
  
public class SaleFactory extends Factory {  
 @Override  
 public Contract createContract(){  
 return new ContractOfSale();  
 }  
}

SupplyFactory.java

package com.factory.factory;  
  
import com.factory.contracts.Contract;  
import com.factory.contracts.SupplyContract;  
  
public class SupplyFactory extends Factory {  
 @Override  
 public Contract createContract(){  
 return new SupplyContract();  
 }  
}

Main.java

package com.factory;  
  
import com.factory.contracts.Contract;  
import com.factory.factory.Factory;  
import com.factory.factory.SaleFactory;  
import com.factory.factory.SupplyFactory;  
  
public class Main {  
  
 public static void main(String[] args) {  
  
 Factory saleFactory = new SaleFactory();  
 Factory supplyFactory = new SupplyFactory();  
  
 Contract ofSale = saleFactory.createContract();  
 Contract supply = supplyFactory.createContract();  
  
 System.*out*.println(ofSale.toString());  
 System.*out*.println();  
 System.*out*.println(supply.toString());  
 }  
}

**ФАСАД**

Я реалізував цей патерн тому що він надає простий інтерфейс до складної системи класів, бібліотеки тощо. Фасад може бути спрощеним відображенням системи, як у даному випадку наприклад складний і довготривалий процес оплати договору. Його можна значно спростити для звичайного користувача адже йому не важливий сам довгий і складний процес оплати, а лише результат.

*Структура:*

Contract.java

package com.facade.components;  
  
public class Contract {  
  
 private String name;  
 private String price;  
 private boolean paid = false;  
  
 public Contract(String name, String price){  
 this.name=name;  
 this.price=price;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getPrice() {  
 return price;  
 }  
  
 public void setPrice(String price) {  
 this.price = price;  
 }  
  
 public boolean isPaid() {  
 return paid;  
 }  
  
 public void setPaid(boolean paid) {  
 this.paid = paid;  
 }  
  
 @Override  
 public String toString() {  
 return "Name:"+getName()+"\nPrice:"+getPrice()+"\t"+isPaid();  
 }  
  
  
}

BankServerRequest.java

package com.facade.components;  
  
public class BankServerRequest {  
  
 private String requestBody;  
  
 public BankServerRequest(String requestBody){  
 this.requestBody=requestBody;  
 }  
  
 public Transaction SendRequestToServer(){  
 System.*out*.println("Sending request body to server");  
 return new Transaction(getRequestBody());  
 }  
  
 public String getRequestBody() {  
 return requestBody;  
 }  
  
 public void setRequestBody(String requestBody) {  
 this.requestBody = requestBody;  
 }  
}

Transaction.java

package com.facade.components;  
  
public class Transaction {  
  
 private String requestBody;  
 private String responseBody;  
  
 public Transaction(String requestBody){  
 this.requestBody=requestBody;  
 }  
  
 public BankServerResponse doTransaction(){  
 System.*out*.println("Processing...");  
 return new BankServerResponse(responseBody);  
 }  
  
 public String getRequestBody() {  
 return requestBody;  
 }  
  
 public void setRequestBody(String requestBody) {  
 this.requestBody = requestBody;  
 }  
}

BankServerResponse.java

package com.facade.components;  
  
public class BankServerResponse {  
 private String responceBody;  
  
 public BankServerResponse(String responceBody){  
 this.responceBody=responceBody;  
 }  
  
 public String SendResponseToServer(){  
 responceBody="Sending request body to server";  
 return responceBody;  
 }  
  
 public String getResponceBody() {  
 return responceBody;  
 }  
  
 public void setResponceBody(String responceBody) {  
 this.responceBody = responceBody;  
 }  
}

Payment.java

package com.facade.facade;  
  
import com.facade.components.BankServerRequest;  
import com.facade.components.BankServerResponse;  
import com.facade.components.Contract;  
import com.facade.components.Transaction;  
  
public class Payment {  
  
 public boolean ContractPay(Contract item) {  
 System.*out*.println("Starting of payment.");  
 BankServerRequest bsrq = new BankServerRequest(item.getPrice());  
 Transaction transaction = bsrq.SendRequestToServer();  
 BankServerResponse bsrp = transaction.doTransaction();  
 String result = bsrp.SendResponseToServer();  
 System.*out*.println(result);  
 item.setPaid(true);  
 System.*out*.println("End of payment.");  
 return true;  
 }  
}

Main.java

package com.facade;  
  
import com.facade.components.Contract;  
import com.facade.facade.Payment;  
  
public class Main {  
  
 public static void main(String[] args) {  
  
 Contract contract = new Contract("Sale of house","10000");  
 Payment payment = new Payment();  
 payment.ContractPay(contract);  
 }  
}

**ЛАНЦЮЖОК ОБОВ’ЯЗКІВ**

Я реалізував цей патерн тому що він озбавляє від жорсткої прив’язки відправника запиту до одержувача, дозволяючи динамічно вибудовувати ланцюг з різних обробників. У моєму випадку я вирішив реалізувати також транзакцію, але з іншої точки зору. Попередній патерн розв’язував проблему простого укриття більшості непотрібного функціоналу для користувача, у цьому ж я реалізував поетапну перевірку задля валідності транзакції: для початку чи існує юзер, якщо так, то чи існує вказана ним карта, і т.д. \*деяка частина коду написана псевдокодом бо розписування усього зайняло би занадто багато часу і можливо створило би неприємні ситуації, тому що патерн не із найлегший в плані реалізації.

BaseCheck.java

package com.chain.check;  
  
public abstract class Basecheck {  
  
 private Basecheck nextCheck;  
  
 public Basecheck linkNext(Basecheck nextCheck){  
 this.nextCheck=nextCheck;  
 return nextCheck;  
 }  
  
 public abstract boolean check();  
  
 public boolean checkNext(String login, String password){  
 if (nextCheck != null){  
 return nextCheck.check();  
 }  
 else return true;  
 }  
}

AuthCheck.java

package com.chain.check;  
  
public class AuthCheck extends Basecheck {  
  
 private String login;  
 private String password;  
  
 public AuthCheck(String login,String password){  
 this.login=login;  
 this.password=password;  
 }  
  
 public boolean check() {  
 System.*out*.println("Checkfor valid...");  
 if (login == null || password == null)  
 {  
 System.*out*.println("Wrong data!!!");  
 System.*out*.println("Transaction failed!!!");  
 System.*exit*(0);  
 }  
 System.*out*.println("User is registered. Transaction continued...");  
 return checkNext(login,password);  
 }  
}

CardCheck.java

package com.chain.check;  
  
import com.chain.server.Server;  
  
public class CardCheck extends Basecheck {  
  
 private String cardNum;  
 private Server server;  
  
 public CardCheck(String cardNum,Server server){  
 this.cardNum=cardNum;  
 this.server=server;  
 }  
  
 public void linkToServer(){  
 //link info about all to server  
 }  
  
 public boolean check() {  
 System.*out*.println("Check card, cvv and balance for valid");  
 if (!server.hasCardNum(cardNum)){  
 System.*out*.println("Card doesn't exists!");  
 return false;  
 }  
 else{  
 //if (balance != price return false;  
 //...  
 }  
 System.*out*.println("Well done! Transaction continiued...");  
 linkToServer();  
 return true; // or false  
 }  
}

Server.java

package com.chain.server;  
  
import com.chain.check.Basecheck;  
  
public class Server {  
  
 private Basecheck basecheck;  
  
 public void setBasecheck(Basecheck basecheck){  
 this.basecheck=basecheck;  
 }  
  
 public boolean hasCardNum(String cardNum){  
 return true; // or false  
 }  
   
 public void doTransaction(){  
 //some logic  
 }  
}

Main.java

package com.chain;  
  
  
import com.chain.check.AuthCheck;  
import com.chain.check.Basecheck;  
import com.chain.check.CardCheck;  
import com.chain.server.Server;  
  
public class Main {  
  
 public static void main(String[] args) {  
 String username="User";  
 String password="Password";  
 String userCard="1111 0000";  
  
 Server server = new Server();  
 Basecheck basecheck = new AuthCheck(username,password);  
 basecheck.check();  
 basecheck.linkNext(new CardCheck(userCard,server)).check();  
  
 server.setBasecheck(basecheck);  
 }  
}

**Висновок:** я пройшов онлайн-курс «Design Patterns» на Coursera, тим самим дізнався неймовірну кількість цікавої інформації про усі шаблони програмування, та збагатив свої знання про програмування, зокрема вирішення та опис найпоширеніших проблем при ООП та програмуванні в цілому.