Wzorce projektowe cz. 2

1. Ściągnięto projekt na lokalną maszynę i zapoznano się z jego strukturą.
2. Dlaczego warto zostawić redundantne dane w klasach Towar i Pozycja?

   Dzięki takiemu zabiegowi klient nie jest świadom o istnieniu klasy Towar. Nie musi sięgać do tej klasy w celu wydobycia danych tylko korzysta z tych, które są zamieszczone w klasie Pozycja.

STRATEGIA

1. Zaimplementowano wzorzec strategia, który udostępnia dwie możliwości naliczania rabatu dla faktury (po dodawaniu nowego towaru albo stała kwota jest odliczana od ceny, albo cena jest przemnażana przez odpowiednią zniżkę wyrażoną w procentach).

- wspólny interfejs dla różnych strategii naliczania rabatu:

```
package rabaty;

public interface IObliczCenePoRabacie {
    double obliczCenePoRabacie(double cena);
    String typRabatu();
}
```

- obliczanie rabatu kwotowego:

```
package rabaty;

public class ObliczCenePoRabacieKwotowym implements
IObliczCenePoRabacie {

    private int znizkaWZlotowkach = 50;
    @Override
    public double obliczCenePoRabacie(double cena) {
        return Math.max(cena - znizkaWZlotowkach, 0);
    }

    public int getZnizkaWZlotowkach() {
        return znizkaWZlotowkach;
    }
```

```
    @Override
    public String typRabatu() {
        return "rabat kwotowy";
    }
}
```

- obliczanie rabatu procentowego:

```
package rabaty;

public class ObliczCenePoRabacieProcentowym implements
IObliczCenePoRabacie {

    private int znizkaWProcentach = 50;
    @Override
    public double obliczCenePoRabacie(double cena) {
        return cena * znizkaWProcentach / 100;
    }

    @Override
    public String typRabatu() {
        return "rabat procentowy";
    }

    public int getZnizkaWProcentach() {
        return znizkaWProcentach;
    }
}
```

Dodano do Faktury referencje do metody obliczania rabatu (wykorzystano interfejs). Podczas dodawania pozycji na fakturę obliczono rabat i zmieniono cenę na pozycji (setCena).

- kod klasy Faktura:

```
package dokumenty;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.Date;
```

```java
import magazyn.Towar;
import rabaty.IObliczCenePoRabacie;


public class Faktura {
    Date dataSprzedazy;
    String kontrahent;
    ArrayList<Pozycja> pozycje;
    double suma;
    IObliczCenePoRabacie iObliczCenePoRabacie;

    public Faktura(Date dataSprzedazy, String kontrahent) {
        this.dataSprzedazy = dataSprzedazy;
        this.kontrahent = kontrahent;
        pozycje = new ArrayList<Pozycja>();
        suma = 0;
    }

    public void dodajPozycje(Towar towar, double ilosc) {
        pozycje.add(new Pozycja(towar, ilosc));
        this.przeliczSume();
        this.dodajRabat(this.iObliczCenePoRabacie);
    }

    public double getSuma() {
        return suma;
    }

    public Date getDataSprzedazy() {
        return dataSprzedazy;
    }

    //jak sie zmieni cos na fakturze to trzeba wywolac te metode
    public Iterator<Pozycja> getIteratorPozycji() {
        return pozycje.iterator();
    }

    public String getKontrahent() {
        return this.kontrahent;
    }

    public void wybierzSposobNaliczaniaRabatu(IObliczCenePoRabacie
```

```
iObliczCenePoRabacie) {
        this.iObliczCenePoRabacie = iObliczCenePoRabacie;
    }

    public String zwrocTypRabatu() {
        return this.iObliczCenePoRabacie.typRabatu();
    }

    private void przeliczSume() {
        Iterator<Pozycja> iteratorPozycji = pozycje.iterator();
        Pozycja pozycja;
        suma = 0;
        while (iteratorPozycji.hasNext()) {
            pozycja = iteratorPozycji.next();
            suma += pozycja.getWartosc();
        }
    }

    private void dodajRabat(IObliczCenePoRabacie
iObliczCenePoRabacie) {
        this.suma =
iObliczCenePoRabacie.obliczCenePoRabacie(this.suma);
    }
}
```

Powyższa klasa jest klasą kontekstową we wzorcu strategii, ponieważ to właśnie ona przechowuje referencje do jednej z wybranych strategii obliczania rabatu przez klienta. Metoda `wybierzSposobNaliczaniaRabatu(IObliczCenePoRabacie iObliczCenePoRabacie)` wywoływana po stronie klienta jest odpowiedzialna za utworzenie wybranej strategii i korzystanie z jej podczas dodawania nowych towarów do faktury.

> Utworzono klasę Konfiguracja w oparciu o wzorzec Singleton. Klasa ta posiada metodę getObliczanieRabatu() zwracająca odpowiedni rabat. Faktura w konstruktorze od teraz odwołuje się do Konfiguracji i pobiera aktualną metodę obliczania rabatu (zamiast tworzyć kalkulator rabatu własnoręcznie).

- kod klasy Konfiguracja w oparciu o wzorzec Singleton:

```
package rabaty;

public class Konfiguracja {
```

```
    private static Konfiguracja instance;

    private Konfiguracja() {}

    public static Konfiguracja getInstance() {
        if (instance == null) {
            instance = new Konfiguracja();
        }
        return instance;
    }

    public IObliczCenePoRabacie getObliczanieRabatu() {
        return new ObliczCenePoRabacieProcentowym();
    }
}
```

Widzimy więc, że rola klasy kontekstowej została oddelegowana do nowo utworzonej klasy Konfiguracja. Dzięki temu zdjęliśmy z klasy Faktura obowiązek wyboru sposobu naliczania rabatu (poprzednio łamana była zasada pojedynczej odpowiedzialności). Teraz klasa Faktura zarządza tworzeniem faktury a sposób wyboru odpowiedniego rabatu leży w zakresie obowiązków klasy Konfiguracja.

- konstruktor klasy Faktura:

```
public Faktura(Date dataSprzedazy, String kontrahent) {
    this.dataSprzedazy = dataSprzedazy;
    this.kontrahent = kontrahent;
    pozycje = new ArrayList<Pozycja>();
    suma = 0;
    iObliczCenePoRabacie =
Konfiguracja.getInstance().getObliczanieRabatu();
}
```

Usunięto z niej metodę odpowiedzialną za przypisywanie odpowiedniego rabatu oraz jej wywołanie w klasie Ui.

FASADA

1. Dodano klasę WydrukFaktury:

```
package dokumenty;
```

```
import java.util.Iterator;

public class WydrukFaktury {

    public void drukujFakture(Faktura faktura) {

System.out.println("==========================================================
=");
        System.out.println("FA z dnia: " +
faktura.getDataSprzedazy().toString());
        System.out.println("Wystawiona dla: " +
faktura.getKontrahent());
        System.out.println("Na kwote: " + faktura.getSuma());
        Iterator<Pozycja> iteratorPozycji =
faktura.getIteratorPozycji();
        while (iteratorPozycji.hasNext()) {
            Pozycja pozycja = iteratorPozycji.next();
            System.out.println("Towar: " + pozycja.getNazwa() + " Ilosc:
" + pozycja.getIlosc() + " Wartosc:" + pozycja.getWartosc());
        }
        System.out.println("Naliczono " + faktura.zwrocTypRabatu() +
".");

System.out.println("==========================================================
=");
    }
}
```

To jest nasza fasada, która umożliwia komunikację z klientem.

2. Z klasy Ui usunięto zbędne importy i metody oraz stworzono obiekt klasy WydrukFaktury, który umożliwia komunikację klienta z całym mechanizmem drukowania faktur poprzez fasadę (klasa WydrukFaktury).

```
WydrukFaktury wydrukFaktury = new WydrukFaktury();
wydrukFaktury.drukujFakture(f);
```

ADAPTER

1. Zaimportowano do klasy obliczającej cenę po rabacie procentowym plik jar:

```
package wzorceprojektowe.strategia;
```

```java
import rabatlosowy.LosowyRabat;

public class ObliczCenePoRabacieProcentowym implements
IObliczCenePoRabacie {

    private double znizkaWProcentach = new LosowyRabat().losujRabat();
    @Override
    public double obliczCenePoRabacie(double cena) {
        return cena * znizkaWProcentach;
    }

    @Override
    public String typRabatu() {
        return "rabat procentowy";
    }

    public double getZnizkaWProcentach() {
        return znizkaWProcentach;
    }
}
```

Sama klasa LosowyRabat może być adapterem, ponieważ zwraca z pliku jar liczbę zmiennoprzecinkową za pomocą metody `losujRabat()`.

METODA SZABLONOWA

1. W celu zaimplementowania dwóch dostępnych szablonów do drukowania faktur skorzystano z metody szablonowej.

- kod klasy nadrzędnej (abstrakcyjnej):

```java
package wzorceprojektowe.metodaszablonowa;

import dokumenty.Faktura;
import dokumenty.Pozycja;

import java.util.Iterator;

public abstract class AbstrakcyjnyWydrukFaktury {

    public final void drukujFakture(Faktura faktura) {
        drukujNaglowek(faktura);
        drukujPozycje(faktura);
        drukujStopke(faktura);
    }
```

```
    protected abstract void drukujNaglowek(Faktura faktura);

    protected void drukujPozycje(Faktura faktura) {
        Iterator<Pozycja> iteratorPozycji =
faktura.getIteratorPozycji();
        while (iteratorPozycji.hasNext()) {
            Pozycja pozycja = iteratorPozycji.next();
            System.out.println("Towar: " + pozycja.getNazwa() + " Ilosc:
" + pozycja.getIlosc() + " Wartosc:" + pozycja.getWartosc());
        }
    }

    protected abstract void drukujStopke(Faktura faktura);
}
```

- kod klasy pochodnej (pierwszy wzór):

```
package wzorceprojektowe.fasada;

import dokumenty.Faktura;
import wzorceprojektowe.metodaszablonowa.AbstrakcyjnyWydrukFaktury;

public class WydrukFakturyA extends AbstrakcyjnyWydrukFaktury {

    protected void drukujNaglowek(Faktura faktura) {

System.out.println("==================================================
=");
        System.out.println("PIERWSZY SZABLON FAKTURY");
        System.out.println("Faktura z dnia: " +
faktura.getDataSprzedazy().toString());
        System.out.println("Wystawiona dla: " +
faktura.getKontrahent());
    }

    @Override
    protected void drukujStopke(Faktura faktura) {
        System.out.println("Na kwote: " + faktura.getSuma() + " zł");
        System.out.println("Naliczono " + faktura.zwrocTypRabatu() + ":
" + faktura.zwrocWielkoscRabatu() + ".");

System.out.println("==================================================
=");
    }
```

```
}
```

- kod klasy pochodnej (drugi wzór):

```
package wzorceprojektowe.fasada;

import dokumenty.Faktura;
import wzorceprojektowe.metodaszablonowa.AbstrakcyjnyWydrukFaktury;

public class WydrukFakturyB extends AbstrakcyjnyWydrukFaktury {
    @Override
    protected void drukujNaglowek(Faktura faktura) {

System.out.println("------------------------------------------------");
        System.out.println("DRUGI SZABLON FAKTURY");
        System.out.println("Faktura z dnia: " +
faktura.getDataSprzedazy().toString());
        System.out.println("Wystawiona dla: " +
faktura.getKontrahent());
        System.out.println("---------------------------------------");
    }

    @Override
    protected void drukujStopke(Faktura faktura) {
        System.out.println("---------------------------------------");
        System.out.println("Na kwote: " + faktura.getSuma() + " zł");
        System.out.println("Naliczono " + faktura.zwrocTypRabatu() + ":
" + faktura.zwrocWielkoscRabatu() + ".");

System.out.println("------------------------------------------------");
    }
}
```

- kod klasy klienta:

```
package main;

import java.util.Calendar;
import wzorceprojektowe.fasada.WydrukFakturyA;
import magazyn.Towar;
import dokumenty.Faktura;
import wzorceprojektowe.fasada.WydrukFakturyB;
```

```
public class Ui {

    public static void main(String[] args) {
        Calendar teraz = Calendar.getInstance();

        //Tworzymy towary
        Towar t1 = new Towar(10, "buty");
        Towar t2 = new Towar(2, "skarpety");

        //I przykladowa faktura
        Faktura f = new Faktura(teraz.getTime(), "Fido");
        f.dodajPozycje(t1, 3);
        f.dodajPozycje(t2, 5);

        WydrukFakturyA wydrukFakturyA = new WydrukFakturyA();
        wydrukFakturyA.drukujFakture(f);

        System.out.println();

        WydrukFakturyB wydrukFakturyB = new WydrukFakturyB();
        wydrukFakturyB.drukujFakture(f);
    }
}
```

- do klasy Faktura dodano także wywołanie metody zwracającej wielkość rabatu w postaci napisu:

```
public String zwrocWielkoscRabatu() {
    return this.iObliczCenePoRabacie.wielkoscZnizki();
}
```

- a poniżej zamieszczono ich implementacje:

```
@Override
public String wielkoscZnizki() {
    return "(" + this.znizkaWZlotowkach + " zł)";
}
```

```
@Override
public String wielkoscZnizki() {
    return "(" + this.znizkaWProcentach + " %)";
}
```

2. Wynik na standardowym wyjściu:

```
==================================================
PIERWSZY SZABLON FAKTURY
Faktura z dnia: Wed Jul 19 21:56:04 CEST 2023
Wystawiona dla: Fido
Towar: buty Ilosc: 3.0 Wartosc:30.0
Towar: skarpety Ilosc: 5.0 Wartosc:10.0
Na kwote: 28.4 zł
Naliczono rabat procentowy: (0.29 %).
==================================================

------------------------------------------------
DRUGI SZABLON FAKTURY
Faktura z dnia: Wed Jul 19 21:56:04 CEST 2023
Wystawiona dla: Fido
--------------------------------------
Towar: buty Ilosc: 3.0 Wartosc:30.0
Towar: skarpety Ilosc: 5.0 Wartosc:10.0
--------------------------------------
Na kwote: 28.4 zł
Naliczono rabat procentowy: (0.29 %).
------------------------------------------------
```

KOMPOZYT

1. Korzystając ze wzorca Composite, stworzono drzewo kategorii towarów. Towary posiadają kategorie i podkategorie, każda z nich ma nazwę i ArrayList będący agregatem Towarów. Kategoria jest liściem drzewa, Podkategoria jest węzłem (i zawiera kolejne liście lub węzły). Stworzono kilka przykładowych towarów, przypisano ich do kategorii i podkategorii oraz wypisano drzewo wraz z zawartościami kategorii i podkategorii.

- najpierw stworzono interfejs, który będą implementować wszystkie węzły drzewa:

```java
package kategorie;

public interface Komponent {
    void wypisz();
}
```

- kod klasy Kategoria:

```
package kategorie;

import java.util.ArrayList;

public class Kategoria implements Komponent {
    private final String nazwa;
    private final ArrayList<Komponent> komponenty;

    public Kategoria(String nazwa) {
        this.nazwa = nazwa;
        this.komponenty = new ArrayList<>();
    }

    public void dodajPodkategorie(Komponent komponent) {
        komponenty.add(komponent);
    }

    @Override
    public void wypisz() {
        for (Komponent komponent: komponenty) {
            System.out.println(komponent);
            komponent.wypisz();
        }
    }

    @Override
    public String toString() {
        return "Kategoria: " + this.nazwa;
    }
}
```

- kod klasy Podkategoria:

```
package kategorie;

import magazyn.Towar;

import java.util.ArrayList;

public class Podkategoria implements Komponent {
    private String nazwa;
    private final ArrayList<Komponent> komponenty;
```

```
    public Podkategoria(String nazwa) {
        this.nazwa = nazwa;
        this.komponenty = new ArrayList<>();
    }

    public void dodajTowar(Towar towar) {
        this.komponenty.add(towar);
    }

    @Override
    public void wypisz() {
        for (Komponent komponent: komponenty) {
            komponent.wypisz();
        }
    }

    @Override
    public String toString() {
        return "Podkategoria: " + this.nazwa;
    }
}
```

- klasa Towar jako element drzewa również implementuje ten sam interfejs. Całość klasy wygląda teraz następująco:

```
package magazyn;

import kategorie.Komponent;

public class Towar implements Komponent {
    private double cena;
    private String nazwa;

    public Towar(double cena, String nazwa) {
        this.cena = cena;
        this.nazwa = nazwa;
    }

    //operacje na cenie
    public void setCena(double cena) {
        this.cena = cena;
    }

    public double getCena() {
        return cena;
```

```
    }

    //operacje na nazwie towaru
    public String getNazwa() {
        return nazwa;
    }

    public void setNazwa(String nazwa) {
        this.nazwa = nazwa;
    }

    @Override
    public void wypisz() {
        System.out.println(this.nazwa);
    }
}
```

- stworzono także klasę KompozytKategorii, która odpowiada za tworzenie i modyfikowanie drzewa tak, aby za dużo kodu nie udostępniać klientowi:

```
package kategorie;

import magazyn.Towar;

import java.util.ArrayList;

public class KompozytKategorii implements Komponent {

    private final ArrayList<Komponent> komponenty;

    public KompozytKategorii() {
        komponenty = new ArrayList<>();
    }

    public Kategoria stworzDrzewo() {
        Kategoria kategoria1 = new Kategoria("Odzież");
        Kategoria kategoria2 = new Kategoria("Elektronika");

        Podkategoria podkategoria1 = new Podkategoria("Buty");
        Podkategoria podkategoria2 = new Podkategoria("Słuchawki");
        Podkategoria podkategoria3 = new Podkategoria("Telefony");

        Towar towar1 = new Towar(100, "Yeezy");
        Towar towar2 = new Towar(140, "Jordan");
        Towar towar3 = new Towar(10, "JBL");
```

```
        Towar towar4 = new Towar(15, "Air pods");
        Towar towar5 = new Towar(3, "iPhone 13 Pro");
        Towar towar6 = new Towar(5, "Samsung Galaxy S23");

        this.komponenty.add(kategoria1);
        this.komponenty.add(kategoria2);

        kategoria1.dodajPodkategorie(podkategoria1);
        kategoria2.dodajPodkategorie(podkategoria2);
        kategoria2.dodajPodkategorie(podkategoria3);

        podkategoria1.dodajTowar(towar1);
        podkategoria1.dodajTowar(towar2);
        podkategoria2.dodajTowar(towar3);
        podkategoria2.dodajTowar(towar4);
        podkategoria3.dodajTowar(towar5);
        podkategoria3.dodajTowar(towar6);

        return kategoria1;
    }

    @Override
    public void wypisz() {
        for (Komponent komponent: komponenty) {
            System.out.println(komponent);
            komponent.wypisz();
            System.out.println();
        }
    }
}
```

- i wreszcie kod klienta:

```
package main;


import kategorie.KompozytKategorii;

public class Ui {

    public static void main(String[] args) {
        KompozytKategorii kompozytKategorii = new KompozytKategorii();
        kompozytKategorii.stworzDrzewo();
        kompozytKategorii.wypisz();
    }
```

```
}
```

2. Wynik na standardowym wyjściu:

```
Kategoria: Odzież
Podkategoria: Buty
Yeezy
Jordan

Kategoria: Elektronika
Podkategoria: Słuchawki
JBL
Air pods
Podkategoria: Telefony
iPhone 13 Pro
Samsung Galaxy S23
```