

## Do wykonania zadania użyty został ChatGPT (model o4-mini) od OpenAI.

**Prompt:** Zaproponuj użycie jakichś wzorców projektowych do poprawy tego kodu w Javie.

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

import magazyn.Towar;

public class Faktura {
    Date dataSprzedazy;
    String kontrahent;
    ArrayList<Pozycja> pozycje;
    double suma;
    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();
    }
    public double getSuma()
    {
        return suma;
    }
    public Date getDataSprzedazy()
    {
        return dataSprzedazy;
    }

    //jak sie zmieni cos na fakturze to trzeba wywolac te metode
    private void przeliczSume()
    {
        Iterator<Pozycja> iteratorPozycji=pozycje.iterator();
        Pozycja pozycja;
        suma=0;
        while(iteratorPozycji.hasNext())
        {
            pozycja = iteratorPozycji.next();
```

```

        suma+=pozycja.getWartosc();
    }
}
public Iterator<Pozycja> getIteratorPozycji()
{
    return pozycje.iterator();
}
public String getKontrahent()
{
    return this.kontrahent;
}
}

```

```
package dokumenty;
```

```
import magazyn.Towar;
```

```

public class Pozycja {
    private Towar towar;
    private double cena;
    private double ilosc;
    private double wartosc;
    private String nazwa;

    public Pozycja(Towar towar, double ilosc) {
        this.towar = towar;
        this.ilosc = ilosc;
        this.cena = towar.getCena();
        this.nazwa = towar.getNazwa();
        this.przeliczWartosc();
    }

    public void setTowar(Towar towar) {
        this.towar = towar;
        this.cena = towar.getCena();
        this.przeliczWartosc();
    }

    public double getIlosc() {
        return ilosc;
    }

    public void setIlosc(double ilosc) {
        this.ilosc = ilosc;
        this.przeliczWartosc();
    }

    public double getCena()
    {
        return this.cena;
    }
}

```

```

    }

    public void setCena(double cena) {
        this.cena = cena;
        this.przeliczWartosc();
    }

    public String getNazwa() {
        return nazwa;
    }

    public double getWartosc() {
        return wartosc;
    }

    // jak sie zmieni cos w pozycji to trzeba wywolac te metode
    private void przeliczWartosc() {
        this.wartosc = this.ilosc * this.cena;
    }
}

```

```

package main;

import java.util.Iterator;
import java.util.Calendar;
import magazyn.Towar;

import dokumenty.Faktura;
import dokumenty.Pozycja;

//ZEWNETRZNY RABAT
import rabatlosowy.LosowyRabat;

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);

        wypiszFaktura(f);
    }
}

```

```

        //TEST ZEWN. rabatu
        LosowyRabat lr=new LosowyRabat();
        System.out.println(lr.losujRabat());
    }
    private static void wypiszFaktura(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("=====");
    }
}

```

```

package magazyn;

```

```

public class Towar {
    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;
    }
}

package rabaty;

public interface ObliczCenePoRabacie {
    double obliczCenePoRabacie(double cena);
}

```

## Odpowiedź:

Oto kilka propozycji, jak można zastosować znane wzorce projektowe, by poprawić czytelność, elastyczność i utrzymanie Twojego kodu:

### 1. Strategy – wydzielenie logiki rabatu/taryfy

Zamiast na stałe bindować w `Ui` „zewnętrzny” rabat (`LosowyRabat`), zrób interfejs:

```

public interface Rabat {
    double oblicz(double suma);
}

```

i pozwól `Faktura` przyjmować dowolną strategię rabatową:

```

public class Faktura {
    private Rabat rabatStrategy;
    // ...
    public Faktura(Date data, String kontrahent, Rabat rabat) {
        // ...
        this.rabatStrategy = rabat;
    }
    public double getSumaPoRabacie() {
        return rabatStrategy.oblicz(suma);
    }
}

```

Dzięki temu możesz podmieniać różne algorytmy obliczania rabatu (losowy, stały %, progi ilościowe itp.) bez modyfikowania `Faktura`.

### 2. Observer / Event – automatyczne przeliczanie sumy

Obecnie musisz ręcznie wywoływać `przeliczSume()`. Zastosuj wzorec Observer, żeby każda zmiana pozycji automatycznie powiadamiała fakturę o konieczności przeliczenia:

```

public interface ChangeListener {
    void onChange();
}

public class Faktura implements ChangeListener {

```

```
// ...
@Override
public void onChange() {
    przeliczSume();
}
}
```

W `Pozycja` trzymaj listę słuchaczy:

```
public class Pozycja {
    private List<ChangeListener> listeners = new ArrayList<>();
    public void addListener(ChangeListener l) { listeners.add(l); }
    private void notifyChange() {
        listeners.forEach(ChangeListener::onChange);
    }
    public void setIlosc(double ilosc) {
        this.ilosc = ilosc;
        przeliczWartosc();
        notifyChange();
    }
    // analogicznie w setCena / setTowar
}
```

I podczas dodawania pozycji w fakturze:

```
Pozycja p = new Pozycja(towar, ilosc);
p.addListener(this); // faktura nasłuchuje zmian
pozycje.add(p);
```

### 3. Builder – czytelniejsze tworzenie faktury

Gdy `Faktura` ma wiele opcjonalnych parametrów (np. różne typy rabatów, komentarze, terminy płatności), warto użyć wzorca Builder:

```
public class FakturaBuilder {
    private Date data;
    private String kontrahent;
    private Rabat rabat = new BrakRabat(); // domyślnie brak rabatu
    // ewentualnie inne pola...

    public FakturaBuilder forKontrahent(String k) { this.kontrahent = k; return this; }
}

public FakturaBuilder atDate(Date d) { this.data = d; return this; }
public FakturaBuilder withRabat(Rabat r) { this.rabat = r; return this; }
public Faktura build() {
    Faktura f = new Faktura(data, kontrahent);
    f.setRabatStrategy(rabat);
    return f;
}
}
```

Użycie:

```
Faktura f = new FakturaBuilder()
    .forKontrahent("Fido")
    .atDate(teraz.getTime())
    .withRabat(new LosowyRabat())
    .build();
```

---

#### 4. Decorator – dodawanie dodatkowych opłat/zdarzeń

Jeśli w przyszłości chcesz dodawać np. podatek VAT, opłatę manipulacyjną czy druki elektroniczne, możesz “owijać” fakturę w dekoratory:

```
public abstract class FakturaDecorator extends Faktura {
    protected Faktura wrapped;
    public FakturaDecorator(Faktura f) { super(f.getDataSprzedazy(),
f.getKontrahent()); wrapped = f; }
}

public class PodatekVAT extends FakturaDecorator {
    private double stawka;
    public PodatekVAT(Faktura f, double st) { super(f); stawka = st; }
    @Override
    public double getSuma() {
        return wrapped.getSuma() * (1 + stawka);
    }
}
```

Dzięki temu dodasz VAT bez zmiany oryginalnej klasy.

---

#### 5. Iterable zamiast Iterator

Zamiast eksportować `Iterator<Pozycja>`, zaimplementuj w `Faktura` interfejs `Iterable<Pozycja>`, co pozwoli używać pętli `for-each`:

```
public class Faktura implements Iterable<Pozycja> {
    // ...
    @Override
    public Iterator<Pozycja> iterator() {
        return Collections.unmodifiableList(pozycje).iterator();
    }
}
```

A w `Ui` :

```
for (Pozycja p : faktura) {
    System.out.println(...);
}
```

---

#### 6. Factory – enkapsulacja tworzenia pozycji

Jeśli tworzenie `Pozycja` stanie się bardziej złożone (np. różne reguły rozliczeń czy waluty), warto zastosować fabrykę:

```

public class PozycjaFactory {
    public static Pozycja createStandard(Towar t, double ilosc) {
        return new Pozycja(t, ilosc);
    }
    public static Pozycja createPromocyjna(Towar t, double ilosc, double promCena) {
        Pozycja p = new Pozycja(t, ilosc);
        p.setCena(promCena);
        return p;
    }
}

```

**Prompt:** popraw powyższy kod w javie, dodając wzorce strategię oraz builder. Podawaj całe kody klas, nie tylko co należy dodać

## Odpowiedź:

Stworzyłem klasy `Faktura` i `FakturaBuilder` wykorzystujące wzorzec **Strategy** (interfejs `Rabat`) oraz **Builder** do tworzenia faktur. Poniżej pełne kody:

- dokumenty/Faktura.java
- dokumenty/FakturaBuilder.java

```

package rabaty;

// Strategia rabatu
public interface Rabat {
    double oblicz(double suma);
}

package rabaty;

// Brak rabatu\ npublic class BrakRabat implements Rabat {
    @Override
    public double oblicz(double suma) {
        return suma;
    }
}

package rabaty;

import java.util.Random;

// Przykład: losowy rabat 0-10%
public class LosowyRabat implements Rabat {
    private Random rnd = new Random();

    @Override
    public double oblicz(double suma) {
        double procent = rnd.nextDouble() * 0.10;

```



```
        return suma * (1 - procent);
    }
}
```

I w `Ui.main` :

```
package main;

import java.util.Calendar;
import dokumenty.Faktura;
import dokumenty.FakturaBuilder;
import dokumenty.Pozycja;
import magazyn.Towar;
import rabaty.LosowyRabat;

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

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

        Faktura f = new FakturaBuilder()
            .atDate(teraz.getTime())
            .forKontrahent("Fido")
            .withRabat(new LosowyRabat())
            .build();

        f.dodajPozycje(new Pozycja(t1, 3));
        f.dodajPozycje(new Pozycja(t2, 5));

        System.out.println("Suma przed rabatem: " + f.getSuma());
        System.out.println("Suma po rabacie:   " + f.getSumaPoRabacie());
    }
}
```