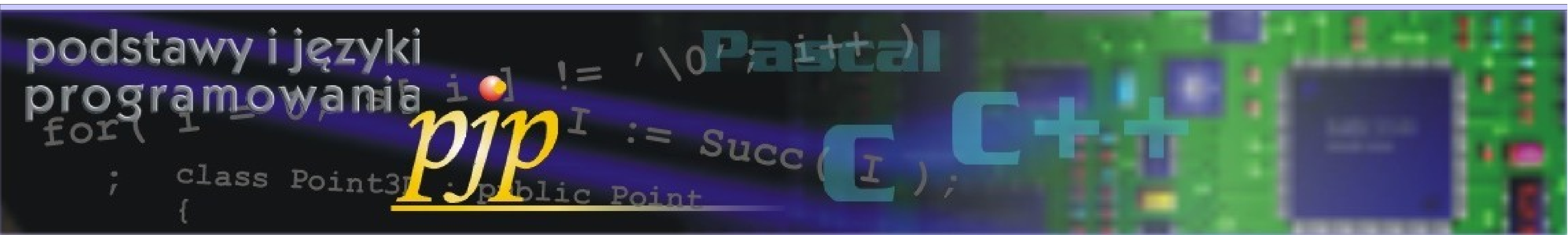
Projektowanie i programowanie obiektowe

Roman Simiński

roman.siminski@us.edu.pl
roman@siminskionline.pl
programowanie.siminskionline.pl

Wzorce projektowe

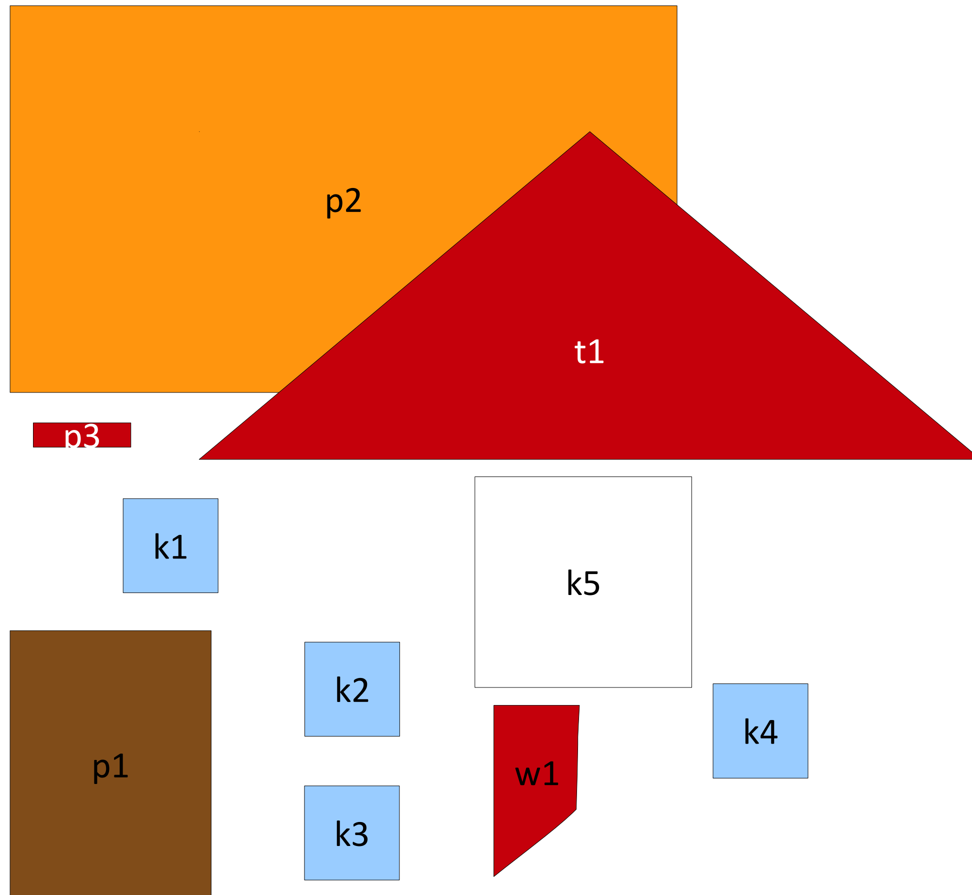
Wybrane wzorce strukturalne: Kompozyt



Kompozyt
Composite Pattern

Kompozyt - koncepcja

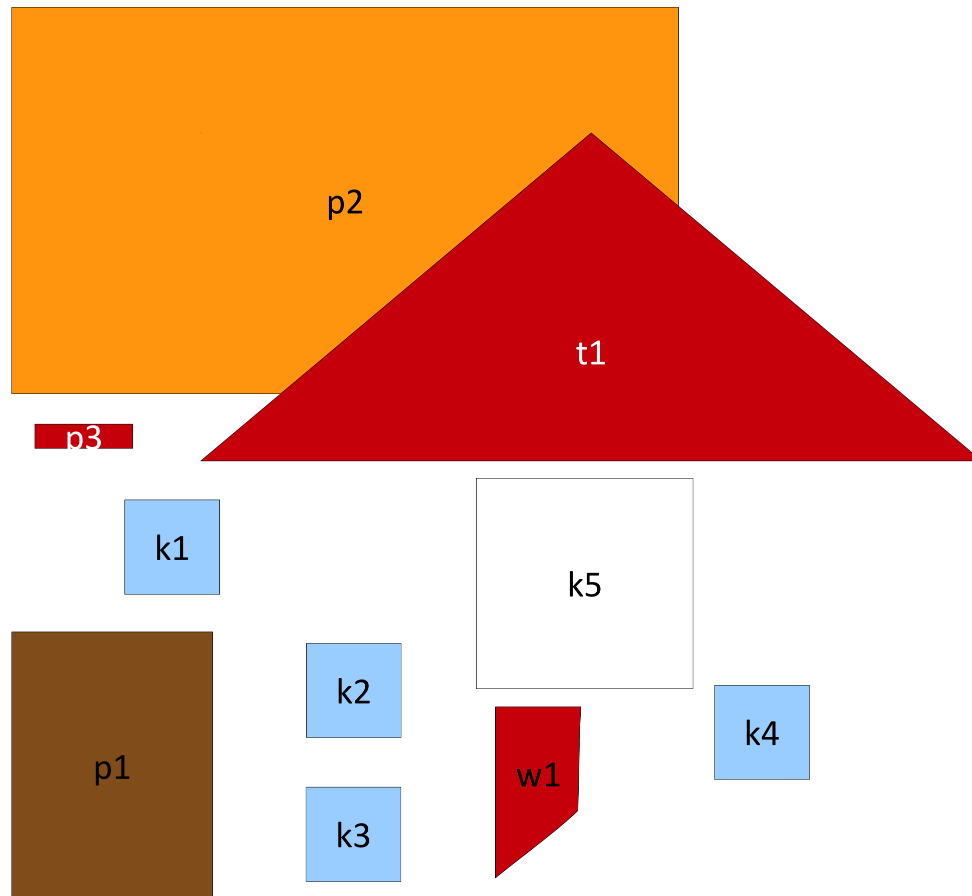
Obiekty graficzne proste



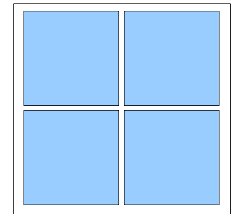

```
k1.rysuj();
k2.rysuj();
k3.rysuj();
k4.rysuj();
k5.rysuj();
p1.rysuj();
p2.rysuj();
p3.rysuj();
w1.rysuj();
p2.rysuj();
```

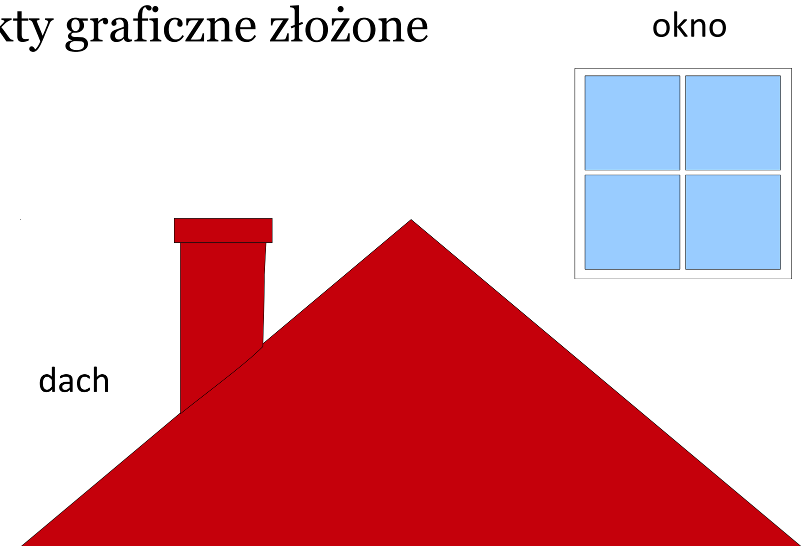
Kompozyt - koncepcja

Obiekty graficzne proste



Obiekty graficzne złożone

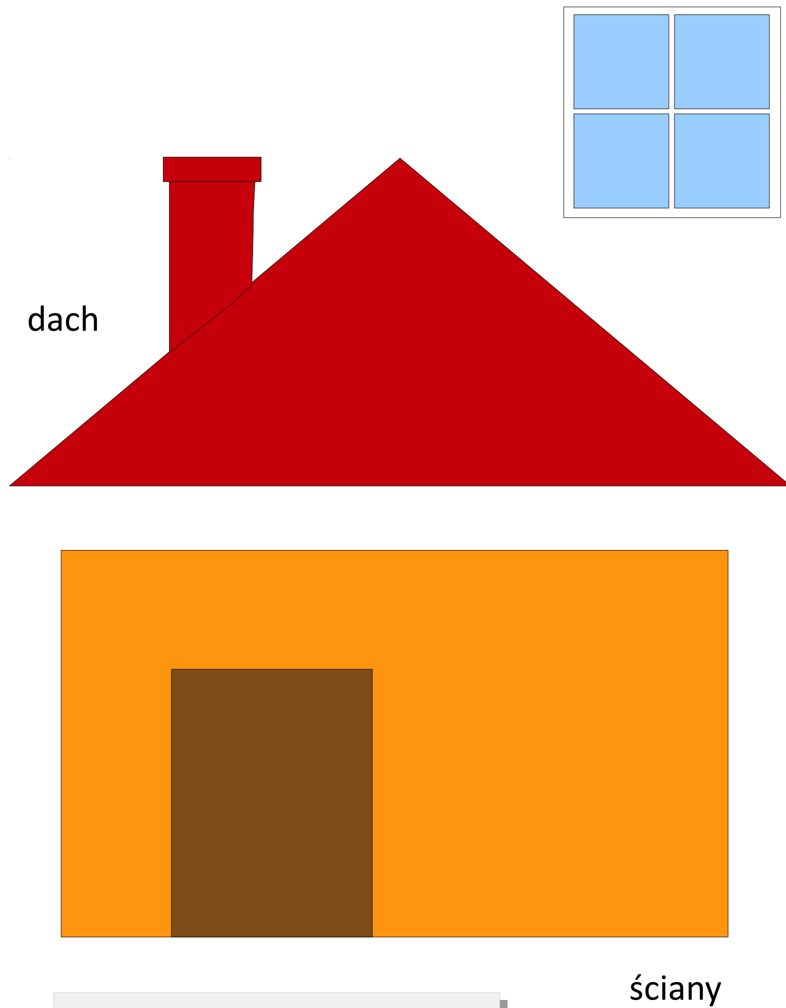




```
ściany.rysuj();
dach.rysuj();
okno.rysuj();
```

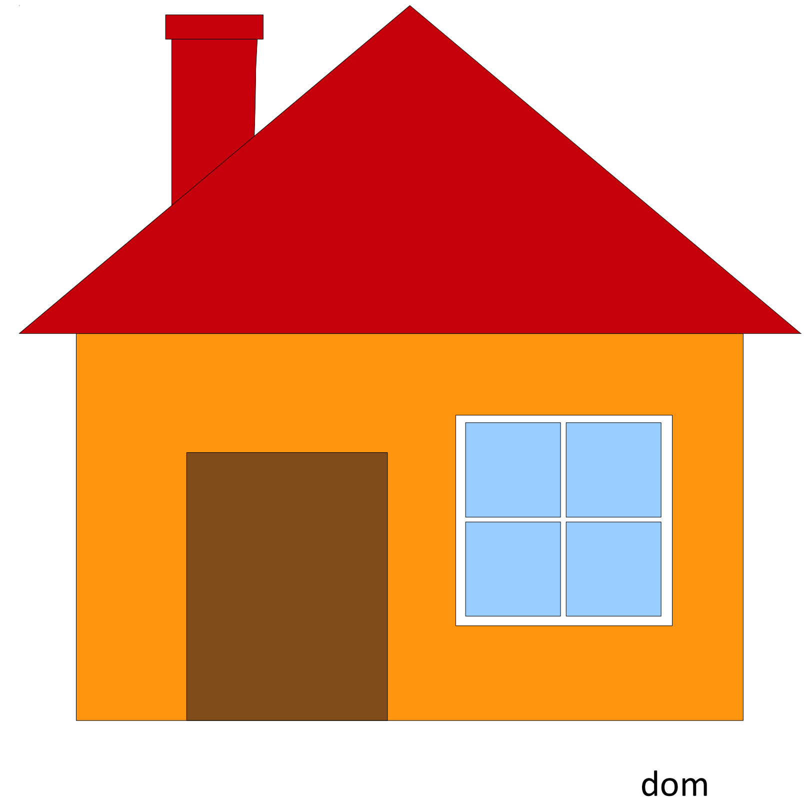
Kompozyt - koncepcja

Obiekty graficzne złożone




```
ściany.rysuj();
dach.rysuj();
okno.rysuj();
```

Obiekty graficzne złożone

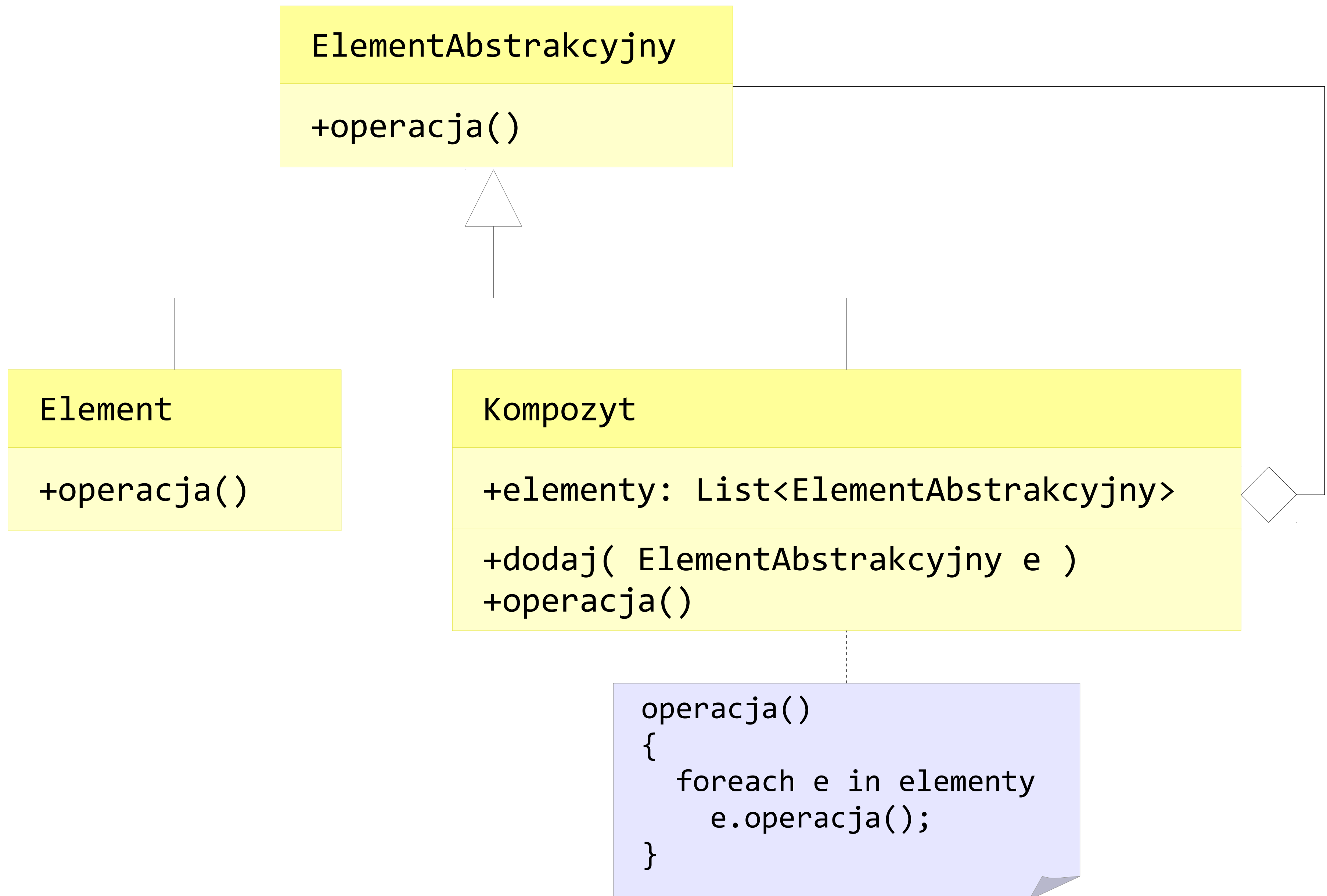



```
dom.rysuj();
```

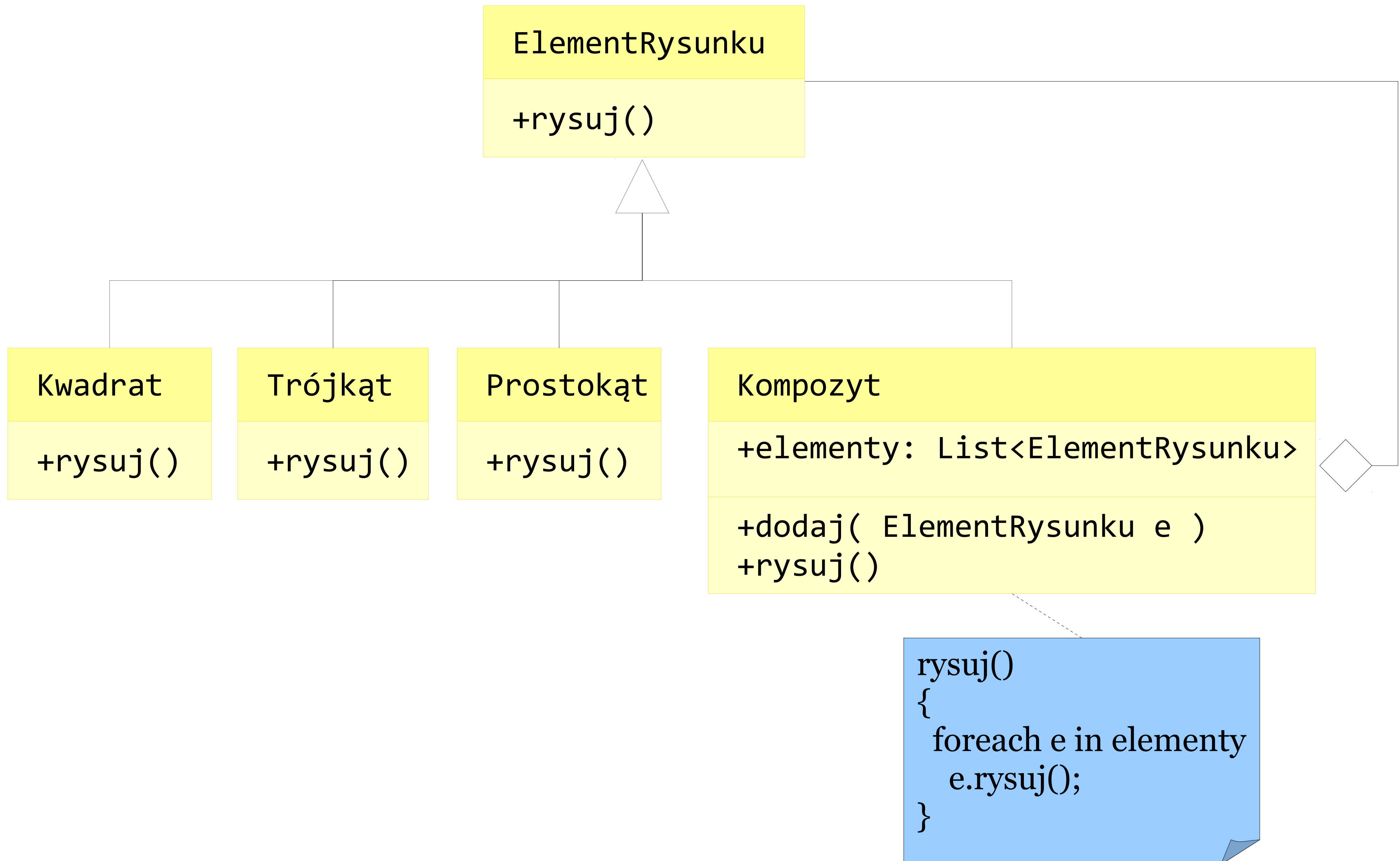
Kompozyt

- Wzorzec ***Kompozyt*** pozwala nam na wykonywanie operacji na pewnym obiekcie złożonym w taki sam sposób jak na obiekcie prostym.
- Operując na *obiekcie złożonym* odnosimy wrażenie, że operujemy na *obiekcie prostym*.
- Obiekt złożony dzięki dziedziczeniu nabywa *cech obiektu prostego*, a dzięki kompozycji staje się *właścicielem podobiektów prostych*.

Kompozyt, schemat UML



Kompozyt, schemat UML, przykład



Przykładowa implementacja kompozytu w języku Java

Element abstrakcyjny lub podstawowy interfejs, klasa elementu

```
abstract class JednostkaWzorcowa
{
    public JednostkaWzorcowa( String nazwa, String typ ) {
        this.nazwa = nazwa;
        this.typ = typ;
    }

    public abstract void wypiszInfo();
    protected String nazwa;
    protected String typ;
}

class JednostkaPodstawowa extends JednostkaWzorcowa
{
    public JednostkaPodstawowa( String nazwa, String typ ) {
        super( nazwa, typ );
    }

    @Override
    public void wypiszInfo() {
        System.out.println( typ + ": " + nazwa );
    }
}
```

Klasa kompozytu

```
class JednostkaOrganizacyjna extends JednostkaWzorcowa
{
    public JednostkaOrganizacyjna( String nazwa, String typ )
    {
        super( nazwa, typ );
    }

    public void dodaj( JednostkaWzorcowa j )
    {
        skladowe.add( j );
    }

    protected ArrayList<JednostkaWzorcowa> skladowe = new ArrayList();

    @Override
    public void wypiszInfo()
    {
        System.out.println( typ + ": " + nazwa + ", zawiera: ");
        for( JednostkaWzorcowa j : skladowe )
            j.wypiszInfo();
    }
}
```

Wykorzystanie

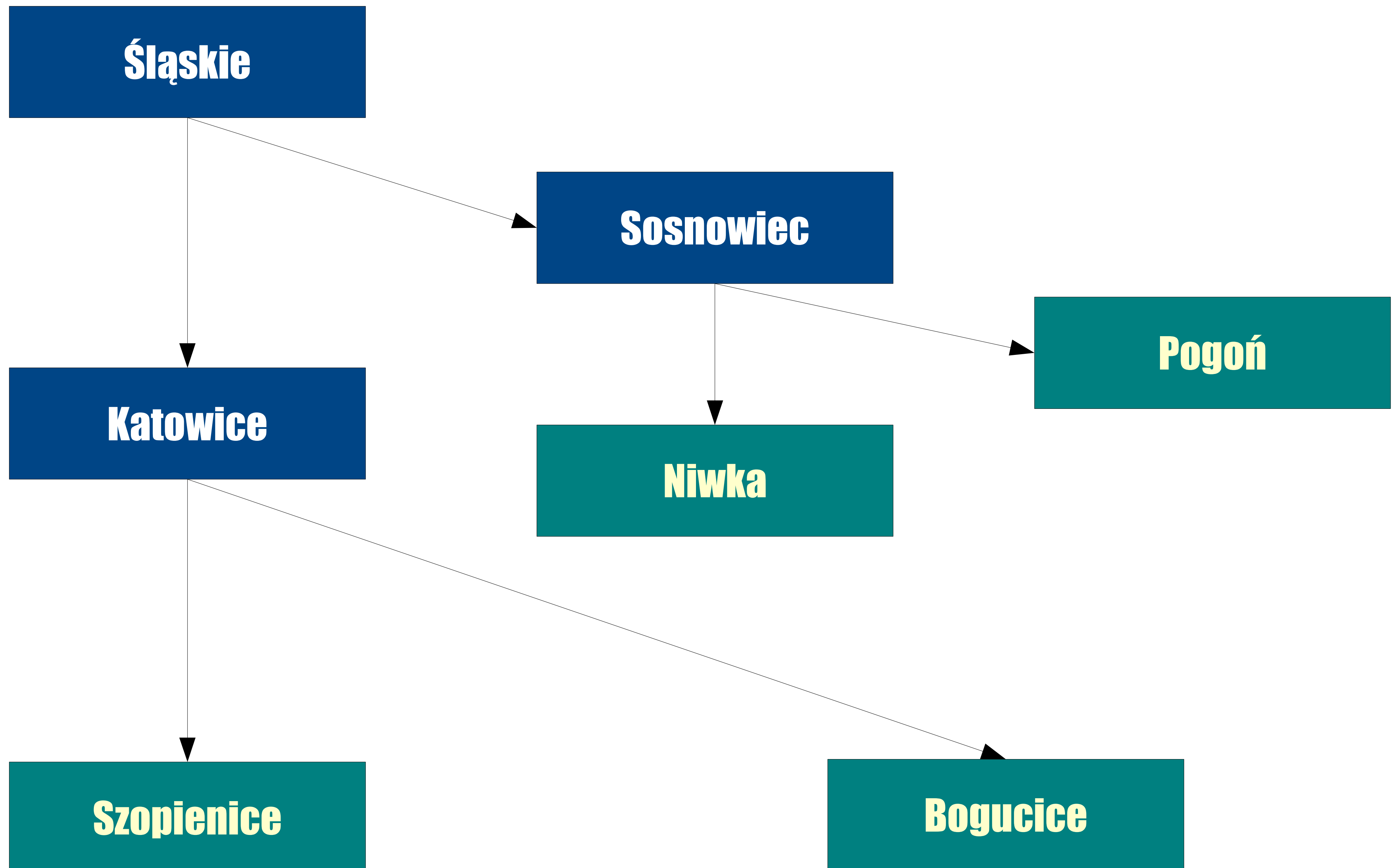
```
// Otworzenie kompozytu miasta
JednostkaOrganizacyjna kato = new JednostkaOrganizacyjna( "Katowice", "miasto" );
// Dodanie dzielnic-elementów do miasta
kato.dodaj( new JednostkaPodstawowa( "Szopienice", "dzielnica" ) );
kato.dodaj( new JednostkaPodstawowa( "Bogucice", "dzielnica" ) );

// Otworzenie kompozytu miasta
JednostkaOrganizacyjna sosno = new JednostkaOrganizacyjna("Sosnowiec", "miasto");
// Dodanie dzielnic-elementów do miasta
sosno.dodaj( new JednostkaPodstawowa( "Pogoń", "dzielnica" ) );
sosno.dodaj( new JednostkaPodstawowa( "Niwka", "dzielnica" ) );

// Otworzenie kompozytu województwo
JednostkaOrganizacyjna slaskie = new JednostkaOrganizacyjna("Śląskie", "województwo" );

// Dodanie miast-kompozytów do kompozytu województwa
slaskie.dodaj( kato );
slaskie.dodaj( sosno );

// Aktywowanie usługi jednakowej dla wszystkich elementów i kompozytów
slaskie.wypiszInfo();
```

Wykorzystanie, rezultat

```
// Otworzenie kompozytu miasta
JednostkaOrganizacyjna kato = new JednostkaOrganizacyjna( "Katowice", "miasto" );
// Dodanie dzielnic-elementów do miasta
kato.dodaj( new JednostkaPodstawowa( "Szopienice", "dzielnica" ) );
kato.dodaj( new JednostkaPodstawowa( "Bogucice", "dzielnica" ) );

// Otworzenie kompozytu miasta
JednostkaOrganizacyjna sosno = new JednostkaOrganizacyjna("Sosnowiec", "miasto");
// Dodanie dzielnic-elementów do miasta
sosno.dodaj( new JednostkaPodstawowa( "Pogoń", "dzielnica" ) );
sosno.dodaj( new JednostkaPodstawowa( "Niwka", "dzielnica" ) );

// Otworzenie kompozytu województwo
JednostkaOrganizacyjna slaskie = new JednostkaOrganizacyjna("Śląskie", "województwo" );

// Dodanie miast-kompozytów do kompozytu województwa
slaskie.dodaj( kato );
slaskie.dodaj( sosno );

// Aktywowanie usługi jednakowej dla wszystki
slaskie.wypiszInfo();
```

```
województwo: Śląskie, zawiera:
miasto: Katowice, zawiera:
dzielnica: Szopienice
dzielnica: Bogucice
miasto: Sosnowiec, zawiera:
dzielnica: Pogoń
dzielnica: Niwka
```

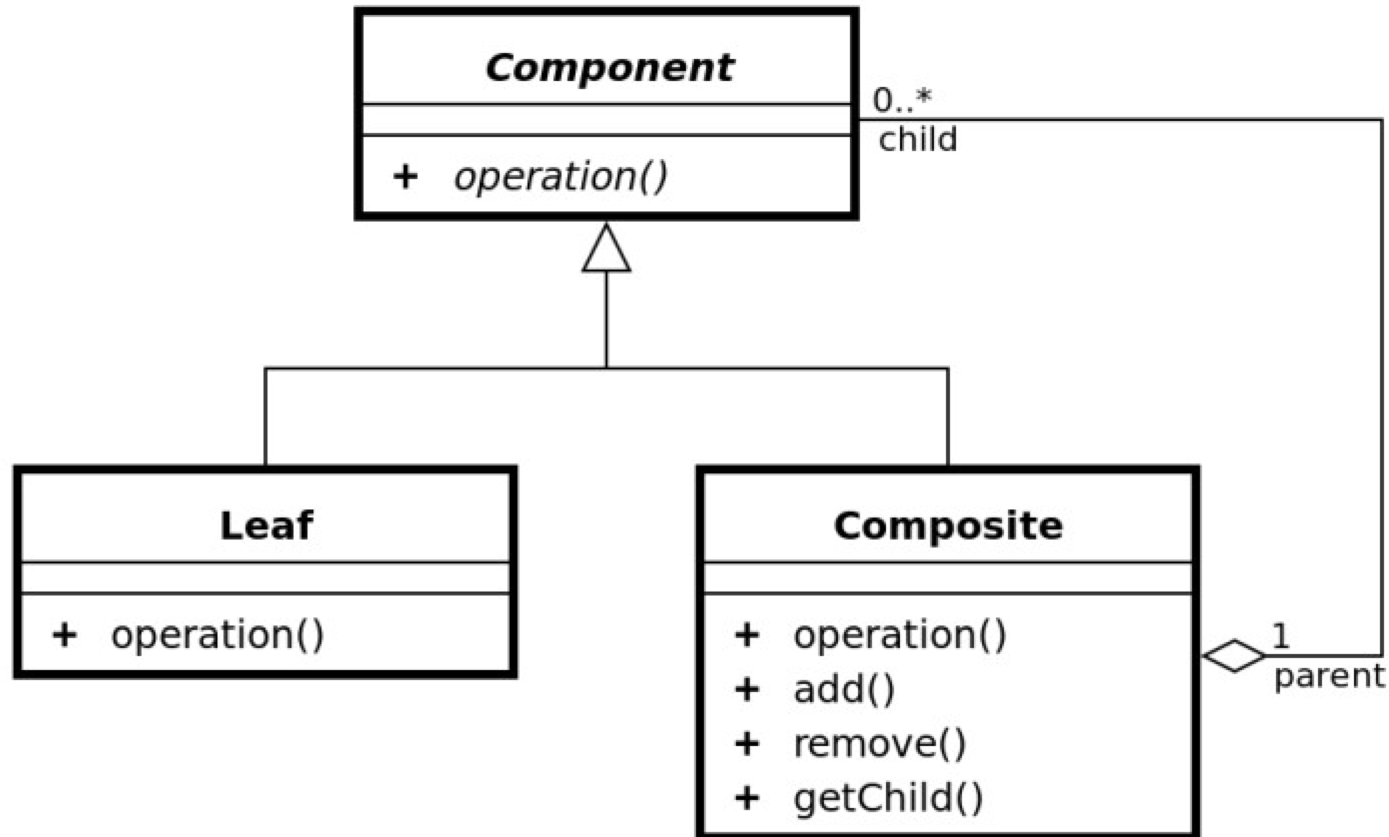
Wzorzec kompozyt tworzy drzewo



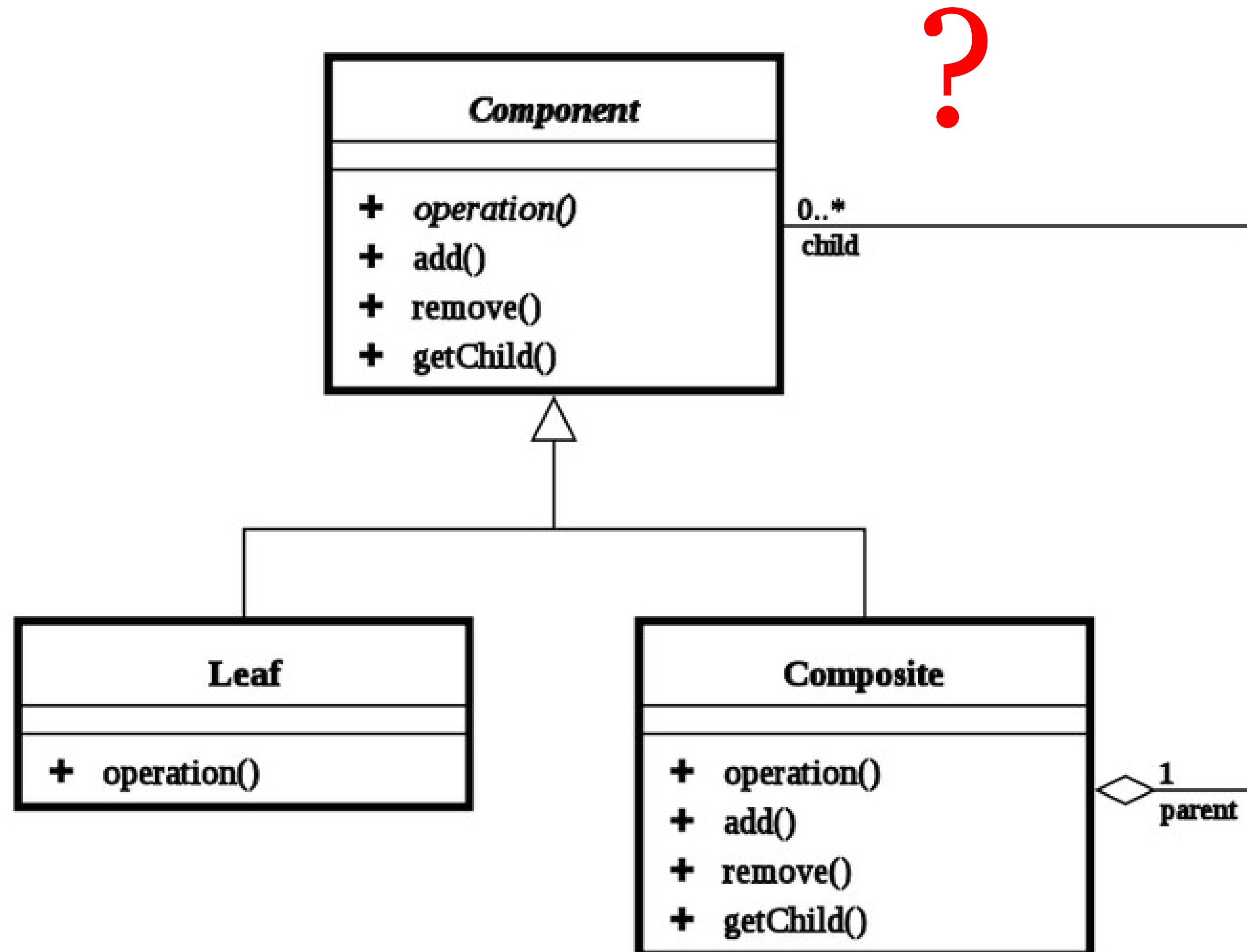
Kompozyt w literaturze i źródłach internetowych

- **Client** – operuje na danych zawartych we wzorcu.
- **Component** – klasa abstrakcyjna albo interfejs który określa pożądane zachowania.
- **Leaf** – klasa obiektu terminalnego, taki obiekt nie ma potomków (obiektów składowych). Dziedziczy z klasy *Component* lub implementujące interfejs *Component*.
- **Composite** – klasa obiektu kompozytu, grupuje obiekty dziedziczące z klasy *Component* lub implementujące interfejs *Component*.

Kompozyt w literaturze i źródłach internetowych



Kompozyt w literaturze i źródłach internetowych



Klasa kompozytu, remanent w przykładzie

```
class JednostkaOrganizacyjna extends JednostkaWzorcowa
{
    public JednostkaOrganizacyjna( String nazwa, String typ )
    {
        super( nazwa, typ );
    }

    public void dodaj( JednostkaWzorcowa j )
    {
        skladowe.add( j );
    }

    protected ArrayList<JednostkaWzorcowa> skladowe = new ArrayList();

    @Override
    public void wypiszInfo()
    {
        System.out.println( typ + ": " + nazwa + ", zawiera: ");
        for( JednostkaWzorcowa j : skladowe )
            j.wypiszInfo();
    }

    . . .
```

Klasa kompozytu

```
    . . .
    public void usun( JednostkaWzorcowa j )
    {
        skladowe.remove( j );
    }

    public void wyczysc()
    {
        skladowe.clear();
    }

    public ArrayList<JednostkaWzorcowa> podajSkladowe()
    {
        return skladowe;
    }
}
```

Czy komponent ma być pojemnikiem?

- Ciekawy artykuł:

https://www.javaworld.com/article/2074564/learn-java/a-look-at-the-composite-design-pattern.html