Wzorce strukturalne

Łukasz Demski

Czym jest wzorzec?

"Wzorzec to sprawdzona koncepcja, która opisuje problem powtarzający się wielokrotnie w określonym kontekście, działające na niego siły, oraz podaje istotę jego rozwiązania w sposób abstrakcyjny„

- Christopher Alexander

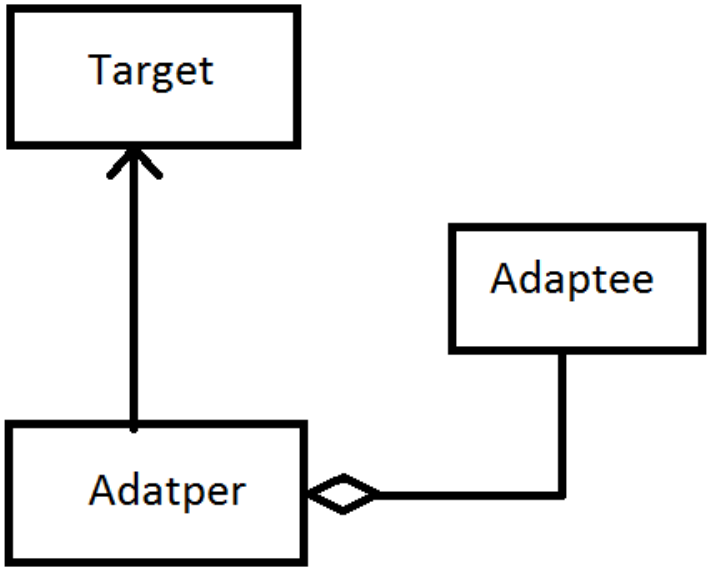
Na czym się skupimy...

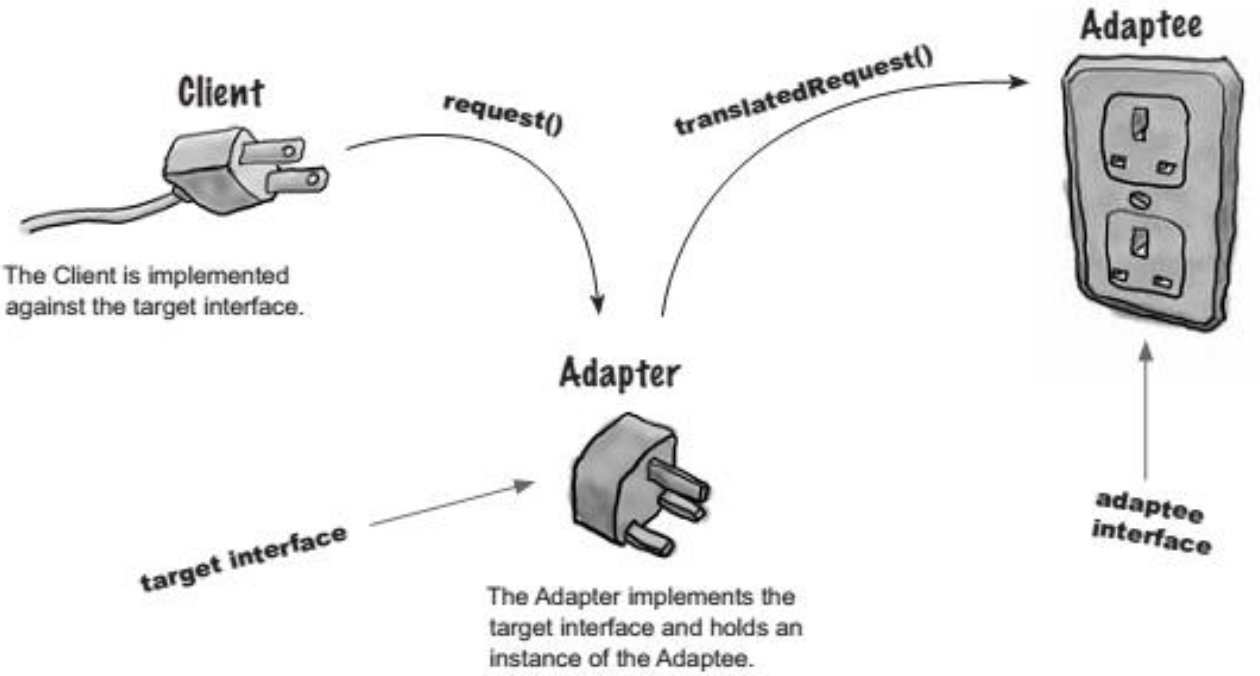
Wzorce strukturalne – opisują pewne sposoby konstrukcji struktur obiektowych.

- Adapter
- Fasada
- Dekorator
- Kompozyt
- Pełnomocnik
- Most

Adapter

- Konwertuje interfejs klasy w inny interfejs klasy klienta.




Client
request()
translatedRequest()
Adaptee
The Client is implemented against the target interface.
Adapter
target interface
adaptee interface
The Adapter implements the target interface and holds an instance of the Adaptee.

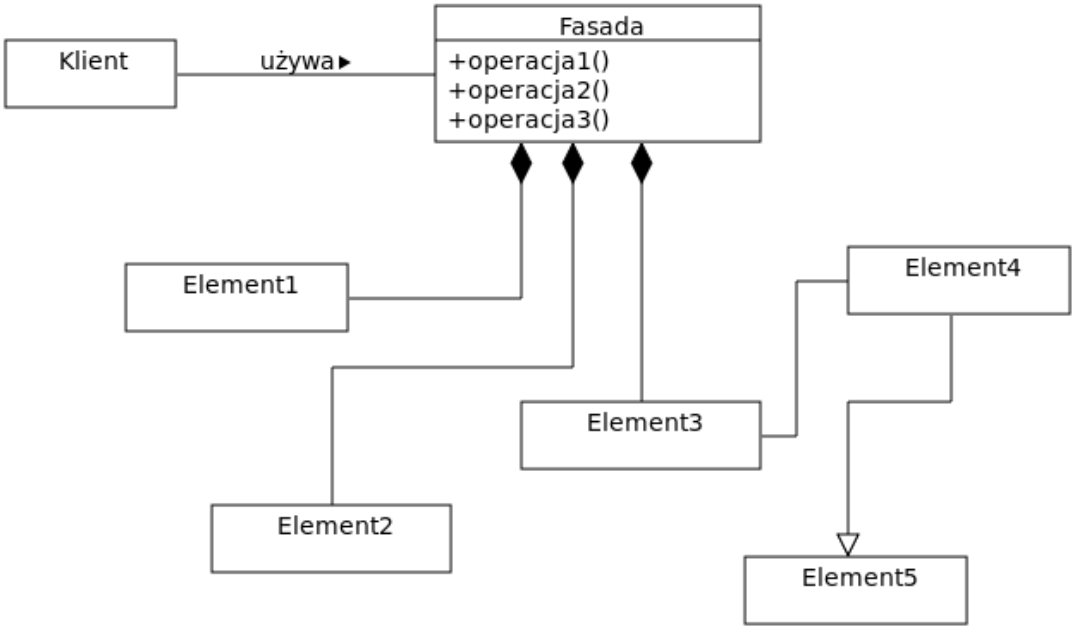
```
class AdapterPLdoUK : public AbstrakcyjnaWtyczkaPl
{

public:
AbstraktycjneGniazdoUk * mojeAbstrakcyjneGniazdoUk;
AdapterPLdoUK(AbstraktycjneGniazdoUk *Agu)
{
	mojeAbstrakcyjneGniazdoUk = Agu;
}

void bolceOkragle()
{
	mojeAbstrakcyjneGniazdoUk->bolcePlaskie();
}
void iloscBolcow()
{
	mojeAbstrakcyjneGniazdoUk->iloscBolcow();
}
};
```

Fasada

- Służy do uproszczenia dostępu do złożonego systemu poprzez udostępnienie uporządkowanego interfejsu.

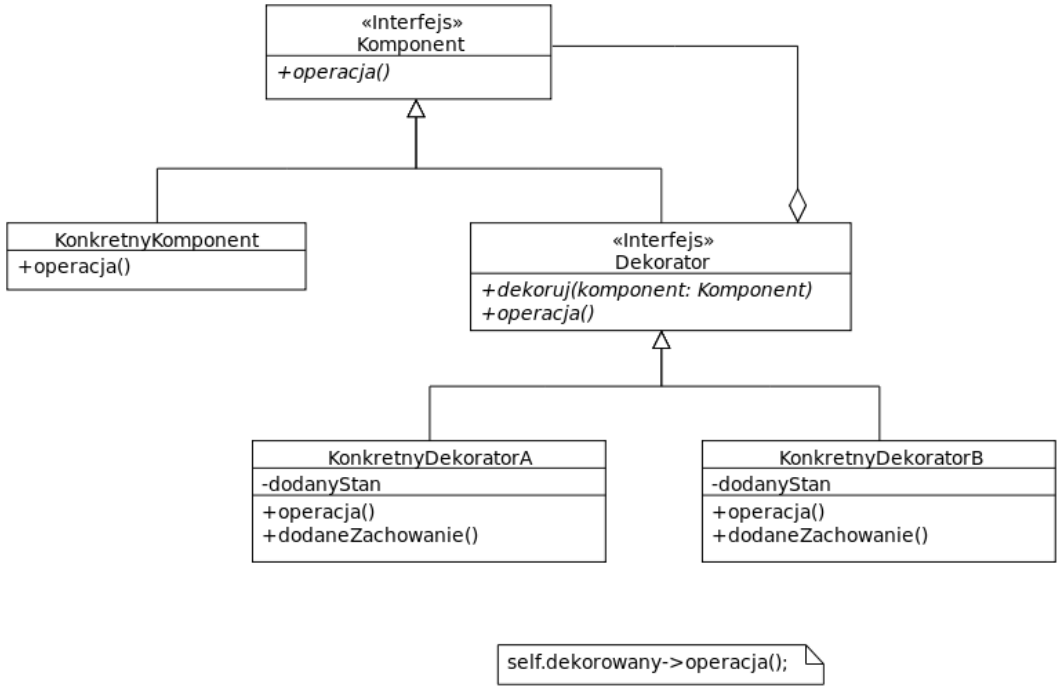


```
class KinoDomoweFasada
{
        Tv * tv;
        Dvd * dvd;
        Oswietlenie * oswietlenie;
public:
KinoDomoweFasada(Tv * tv, Dvd * dvd, Oswietlenie * oswietlenie)
{
        this->tv = tv;
        this->dvd = dvd;
        this->oswietlenie = oswietlenie;
}
public:

void ogladajFilm(string tytulFilmu)
{
        oswietlenie->on();
        tv->wlacz();
        dvd->wlacz();
        dvd->odtwarzajFilm(tytulFilmu);
}
};
```

Dekorator

- Pozwala na dodanie nowej funkcji do istniejących klas dynamicznie podczas działania programu.

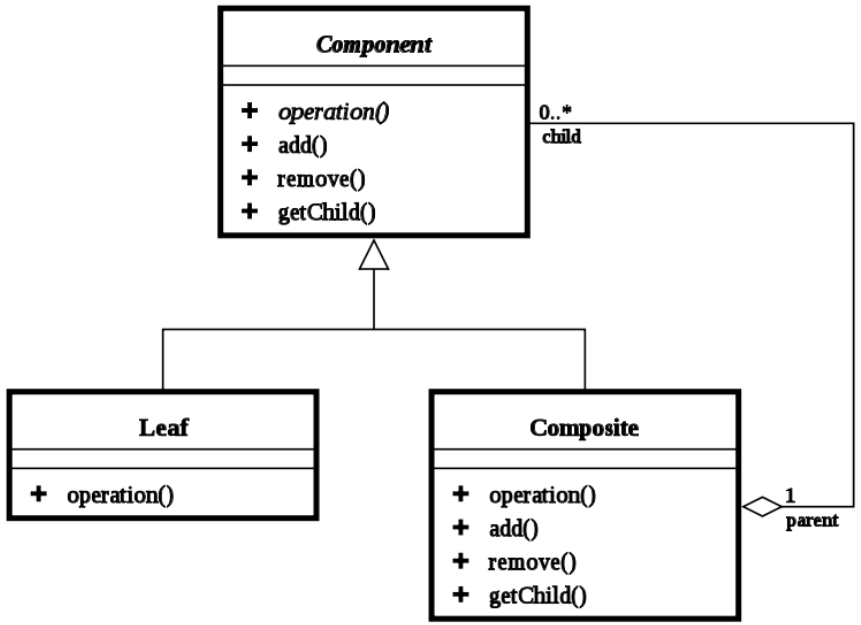


```
class Dekorator : public Widzet
{
        Widzet * widzet;
public:
        Dekorator(Widzet * widzet)
        {
                this->widzet = widzet;
        }

        void rysuj()
        {
                widzet->rysuj();
        }
};
```

Kompozyt

Założenie jest takie aby możliwe było korzystanie z pewnych metod dostępnych dla danego obiektu w kontekście całej grupy podobnych do siebie obiektów.



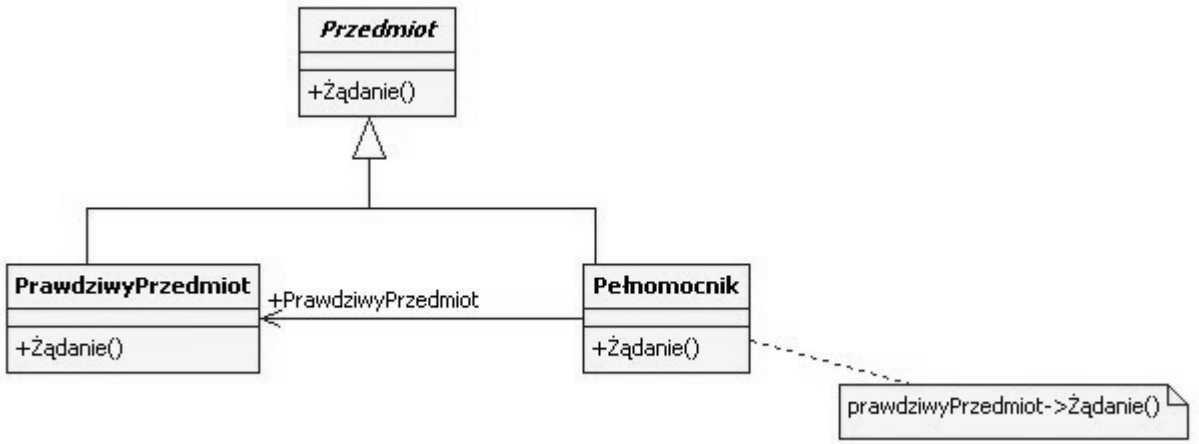
```
class Kompozyt : public InterfejsPracownika
{
vector < InterfejsPracownika * > pracownicy;
public:

void dodaj(InterfejsPracownika * pracownik)
{
        pracownicy.push_back(pracownik);
}

void przedstaw()
{
        for( vector< InterfejsPracownika * >::iterator it =
        pracownicy.begin(); it != pracownicy.end(); ++it) {
                (*it)->przedstaw();
        }
}
};
```

Pełnomocnik

- Pełnomocnik ma zastosowanie zawsze wtedy, kiedy potrzeba bardziej uniwersalnego lub bardziej wyrafinowanego odwołania do obiektu niż zwykły wskaźnik.

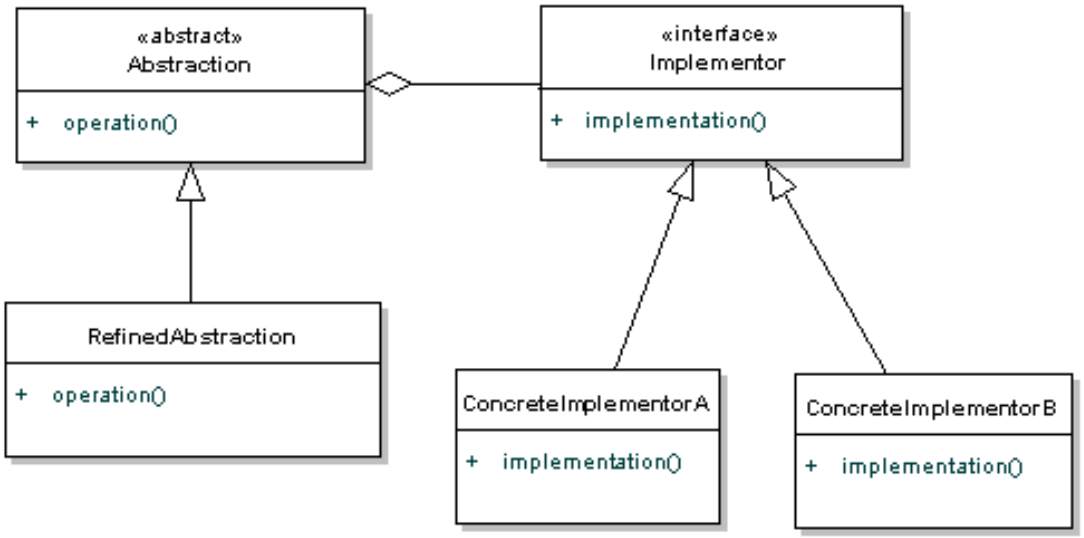


```
class PelnomocnikSamochod : public InterfejsSamochodu
{
InterfejsSamochodu * samochod;
Osoba * osoba;
public:
ProxySamochod(Osoba * osoba)
{
        this->osoba = osoba;
        this->samochod = new Samochod();
}
void prowadz()
{

        if(osoba->isPrawko() == false)
                cout << "Sorry, nie mozesz prowadzic samochodu!";
        else
                samochod->prowadz();
}
};
```

Most

- Stosowanie wzorca pozwala nam odseparować implementację obiektu od jego abstrakcji.



```
class KonkrentyPilotTv : public AbstrakcjaPilotaTv
{
public:
KonkrentyPilotTv(InterfejsTv * iTv)
{
        this->tv = iTv;
}
void przyciskWlacz()
{
        tv->wlacz();
}
void przyciskWylacz()
{
        tv->wylacz();
}
};
```

Demo

Dziękuję za uwagę!