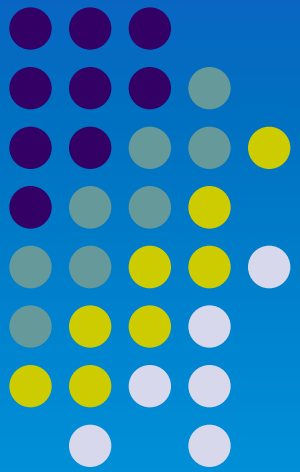
Projektowanie obiektowe
Wzorce projektowe



Gang of Four
Wzorce rozszerzeń

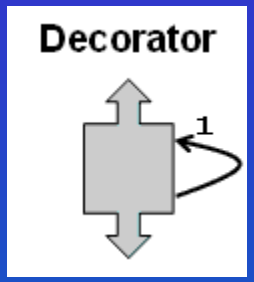
Roadmap

- Decorator
- Iterator
- Visitor

Wzorce rozszerzeń

- Mają na celu uczynić proces rozszerzania kodu bardziej czytelnym, prostym i dostępnym

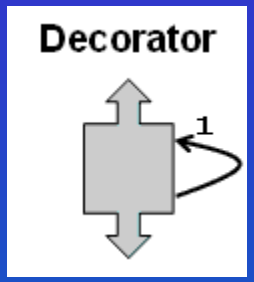
Decorator*



- Umożliwia programistom dynamiczne tworzenie zachowania obiektów.

*GoF - wzorzec strukturalny

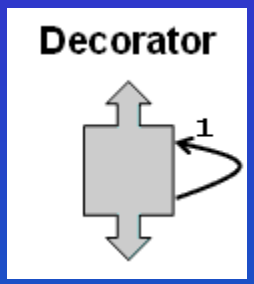
Decorator – problem



- Potrzebujemy rozszerzyć zachowanie klasy, ale istnieją powody, dla których nie chcemy korzystać z dziedziczenia.

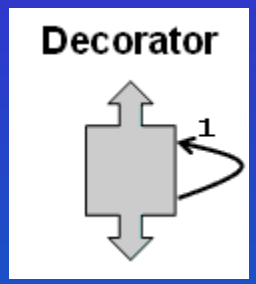
- Potrzebujemy dodać zachowanie lub stan do wybranych obiektów w czasie-wykonania, a dziedziczenie jest nieodpowiednie z powodu statyczności i faktu, że dotyczy całej klasy.

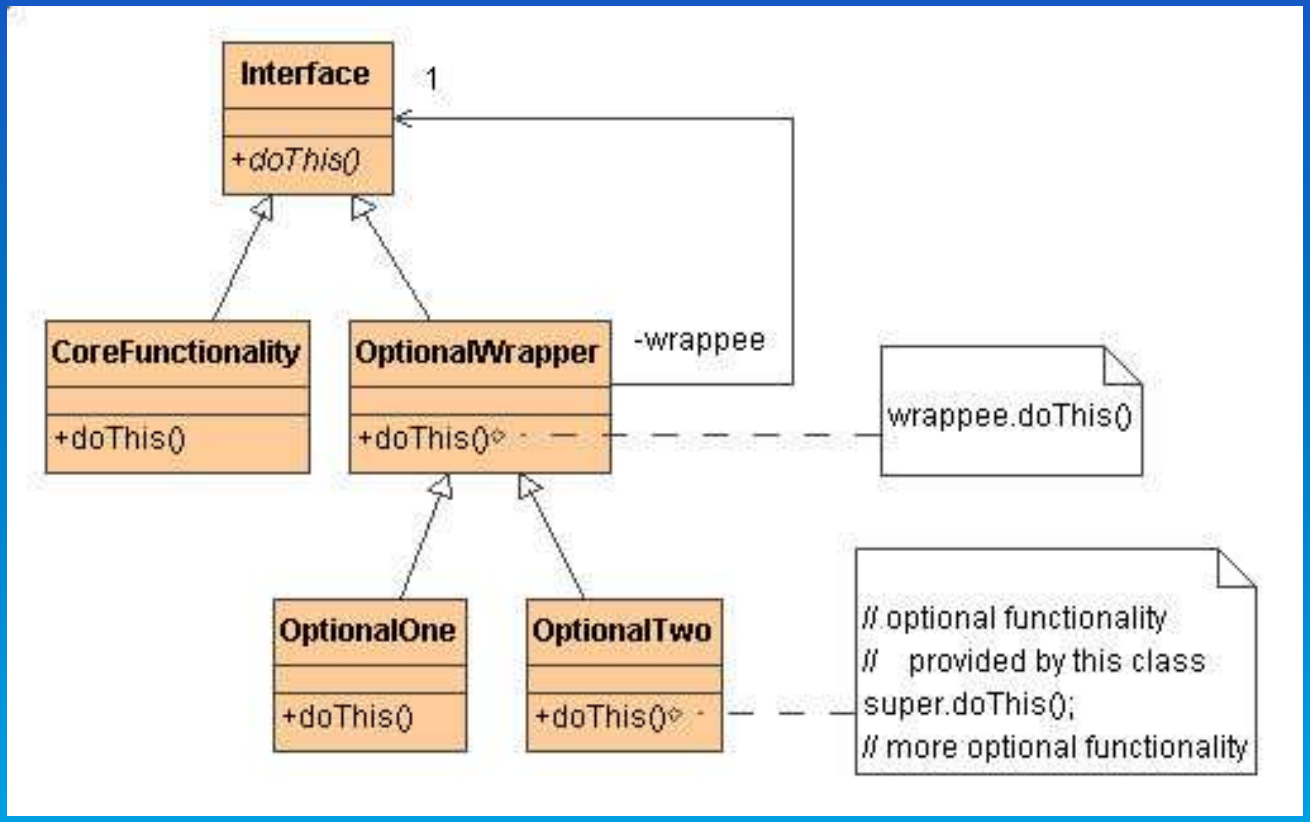
Decorator – rozwiązanie



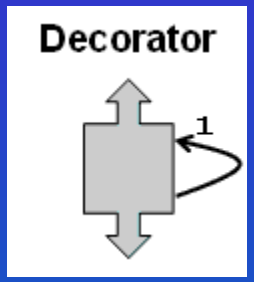
- Stosujemy rekursywną kompozycję.
- Agregacja 1 do 1: „składa się” w górę hierarchii dziedziczenia.
- Wprowadzamy pojedynczy rdzeniowy obiekt osłonięty przez możliwie wiele opcjonalnych obiektów.

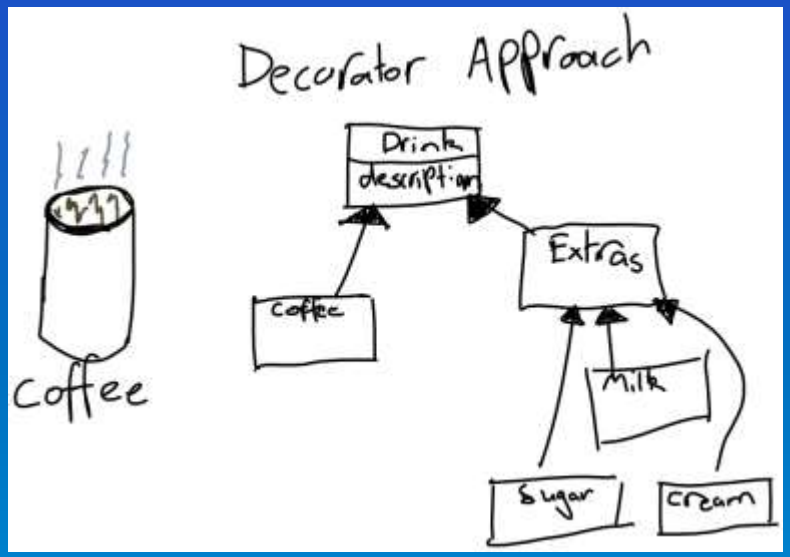
Decorator – diagram klas

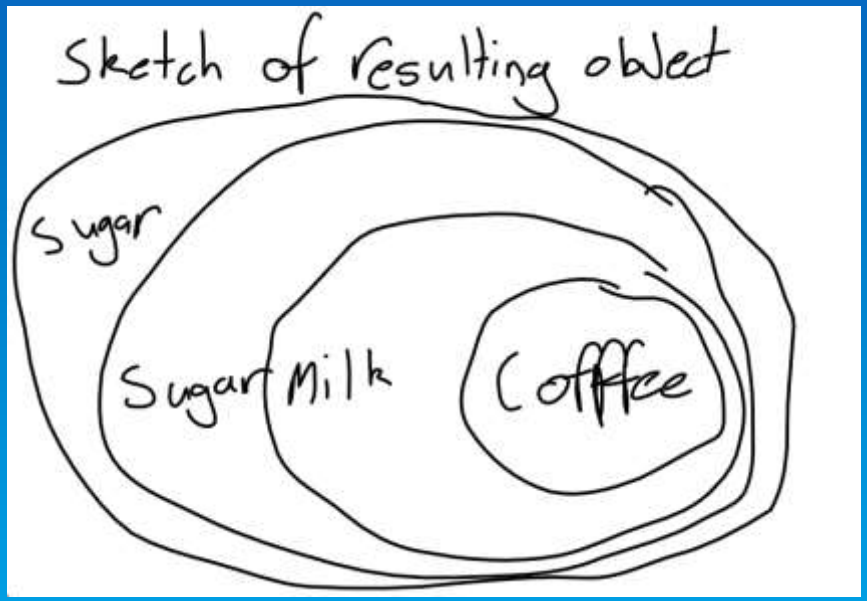




Decorator – diagram klas

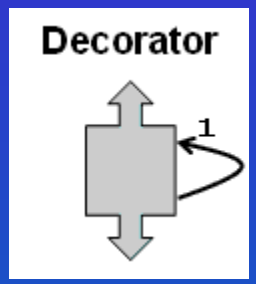


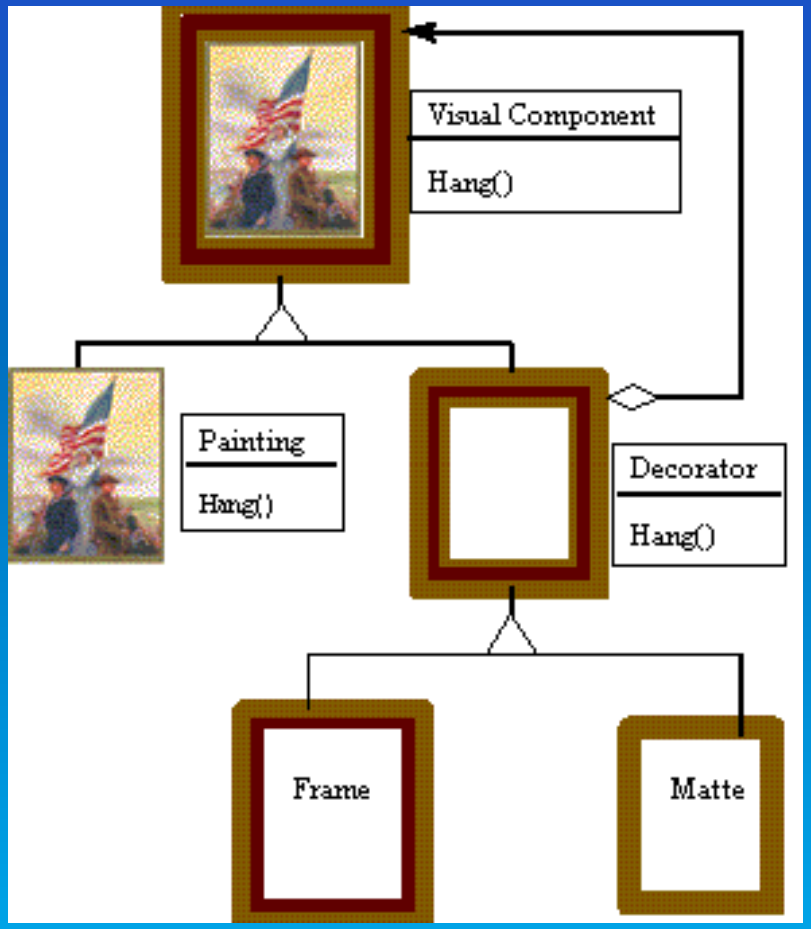




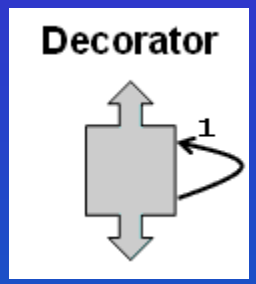
https://tylercash.xyz/post/a-look-at-decorator-patterns

Decorator – przykład





Decorator – konsekwencje



- Dostarcza większą elastyczność niż dziedziczenie.
- Wykorzystując różne kombinacje osłon, tworzy się wiele różnych kombinacji zachowań.
- Mniejsza liczba klas, kosztem większej liczby obiektów.
- Elastyczność osłon zwiększa ryzyko błędów.
- Trudno jest ustalić tożsamość obiektu, bo właściwe obiekty są za osłonami.

Zależności między wzorcami

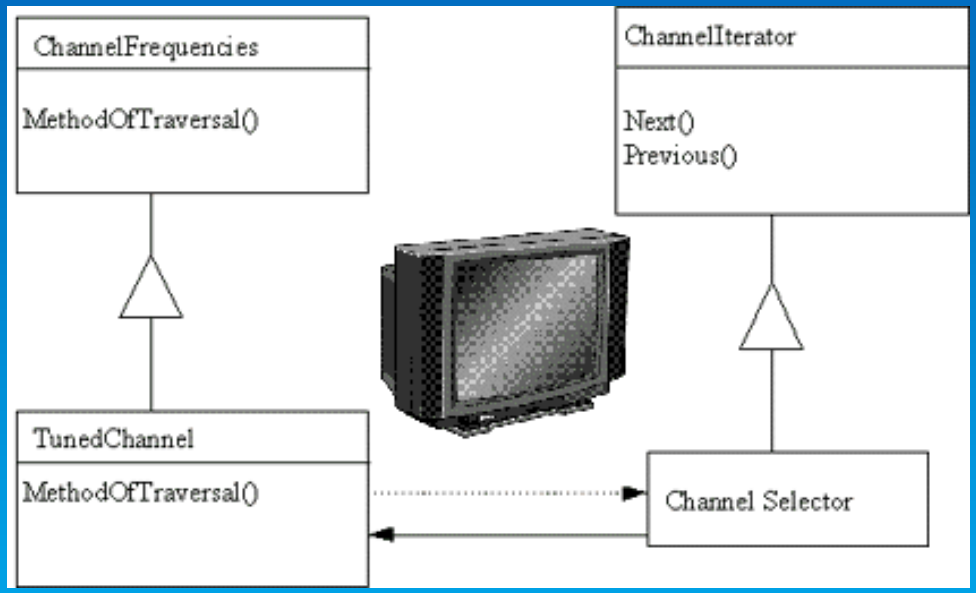
- Dostarczany interfejs jest:
  - inny – **adapter**,
  - ten sam – **proxy**,
  - rozszerzony – **decorator**.
- **Decorator** i **proxy** mają inne zastosowanie, ale podobne struktury (warstwa pośrednicząca).
- **Decorator** pozwala zmienić „skórę” obiektu, a **strategy** „bebechy”.

Zależności między wzorcami

- **Composite** i **decorator** za pomocą rekursywnej kompozycji organizują dowolną liczbę obiektów.
- **Decorator** może być postrzegany jako zdegenerowany **composite**, z jednym komponentem, ale jego celem nie jest agregacja.
- **Decorator** i **composite** są używane razem, bo się uzupełniają.

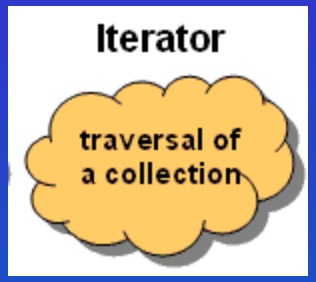
Iterator*

- Udostępnia sposób sekwencyjnego dostępu do elementów kolekcji



*GoF - wzorzec behawioralny

Iterator – problem

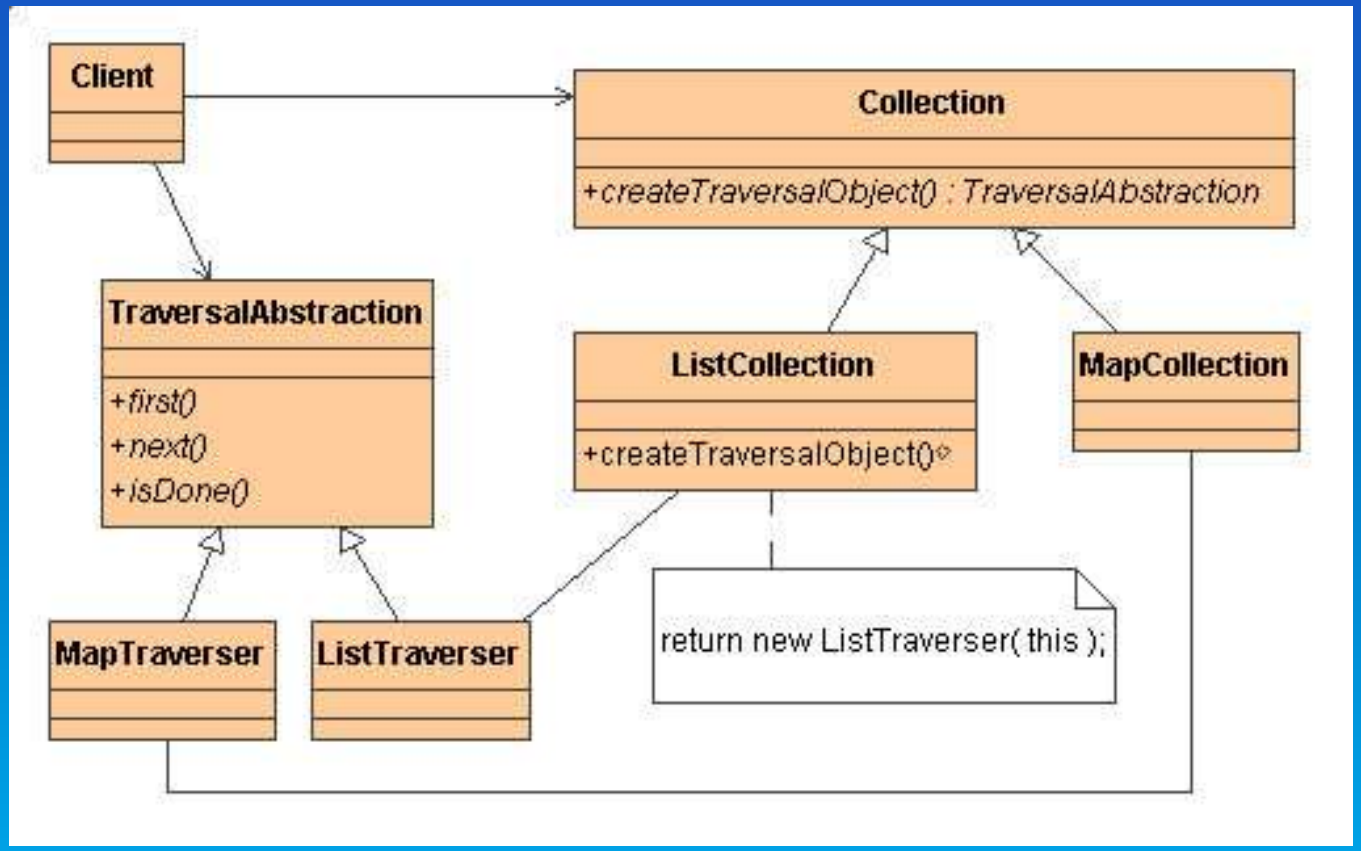


- Potrzebujemy ujednolicić mechanizm przechodzenia (przeglądania) wysoce rożnych struktur danych, tak aby można było zdefiniować algorytmy zdolne do przezroczystego wzajemnego oddziaływania z każdą z nich.

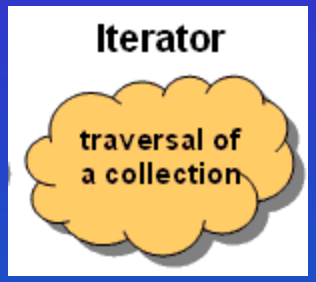
Iterator – rozwiązanie

- Stosujemy polimorfizm do przemierzania (iterowania).
- Przemierzanie kolekcji awansuje do statusu pełnego objektu.

Iterator – diagram klas

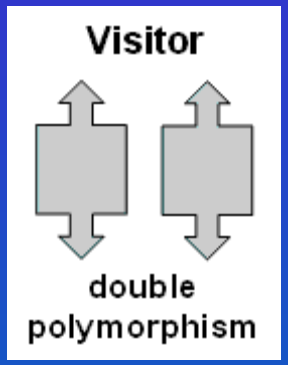


Iterator – konsekwencje



- Jest możliwy dostęp do kolekcji obiektów bez znajomości źródła obiektów.
- Używając wielu obiektów **iteratorów**, jest łatwo posiadać i zarządzać wieloma „przemierzeniami” naraz.
- Klasa kolekcji może dostarczać różne rodzaje iteratorów by przemierzać kolekcje w różny sposób. (np. klucze i wartości indeksów)

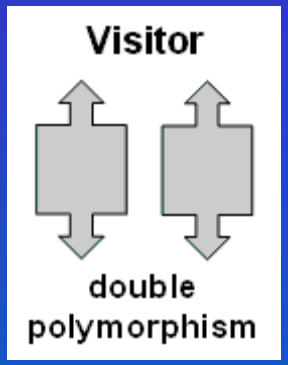
Visitor



- GoF - wzorzec behawioralny.

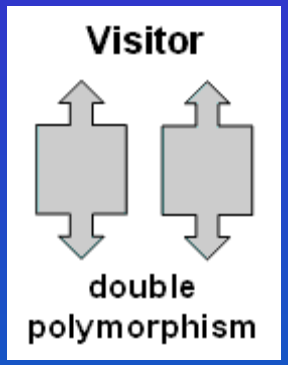
- Umożliwia programistom zdefiniowanie nowej operacji dla hierarchii bez konieczności zmiany klas zawartych w tej hierarchii.

Visitor – problem



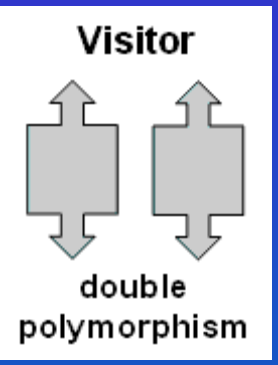
- Wiele różnych i niezwiązanych operacji trzeba przeprowadzić na obiektach węzłowych w heterogenicznej złożonej strukturze, ale:
  - Chcemy uniknąć „zanieczyszczania” klas węzłowych w te operacje.
  - Nie chcemy wykonywać sprawdzenia typu każdego węzła i rzutować wskaźnik do właściwego typu przed wykonaniem pożądanej operacji.

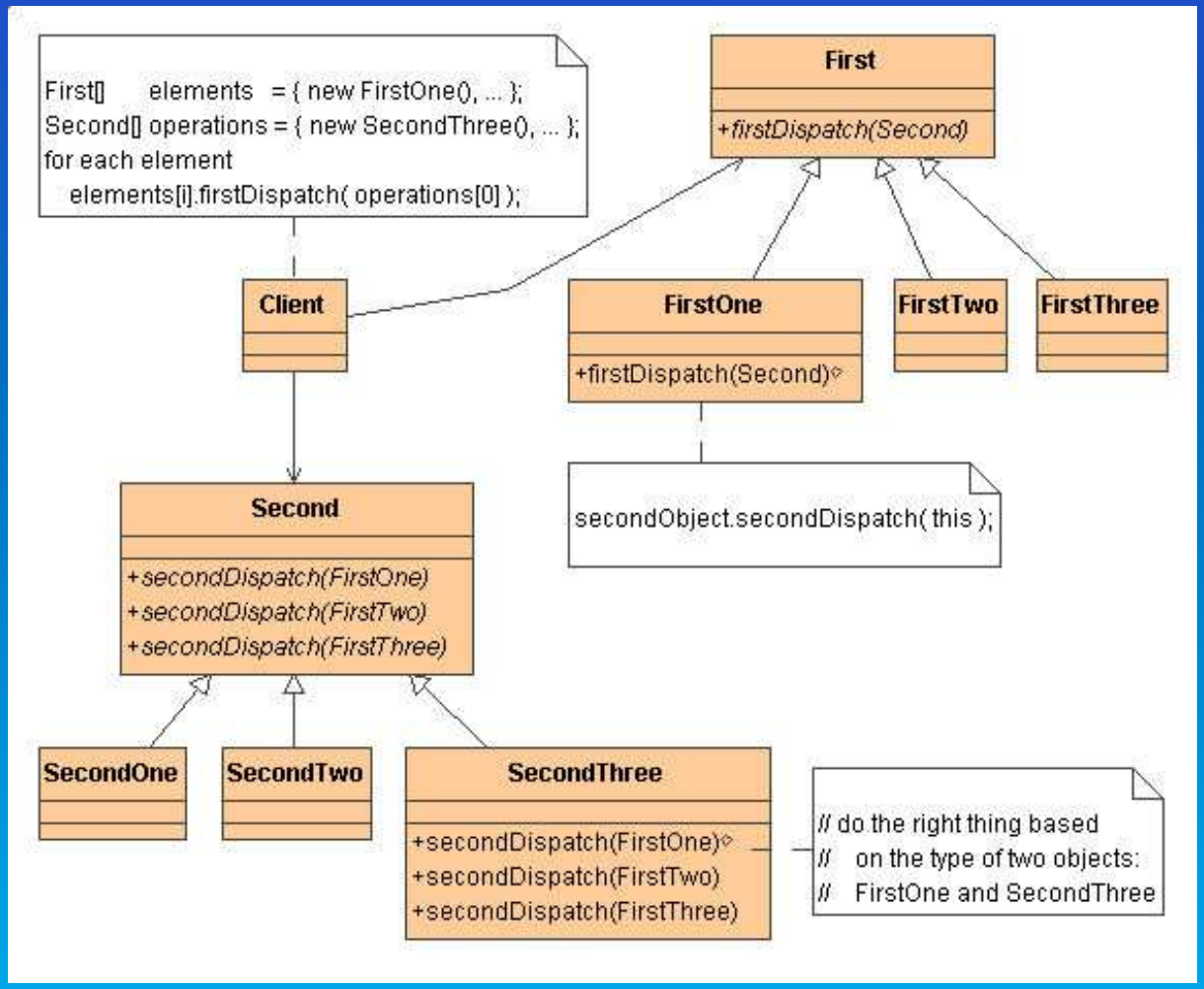
Visitor – rozwiązanie



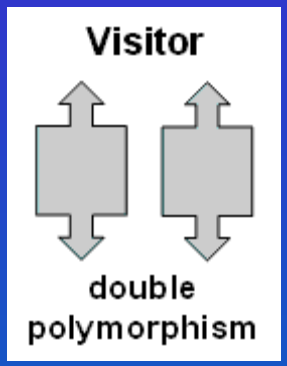
- Podwójne rozdzielenie (double dispatch).
- Wybór wykonywanej operacji należy uzależnić od typu dwóch obiektów.
- Implementacja mechanizmu umożliwiającego dodawanie operacji do istniejącej hierarchii.

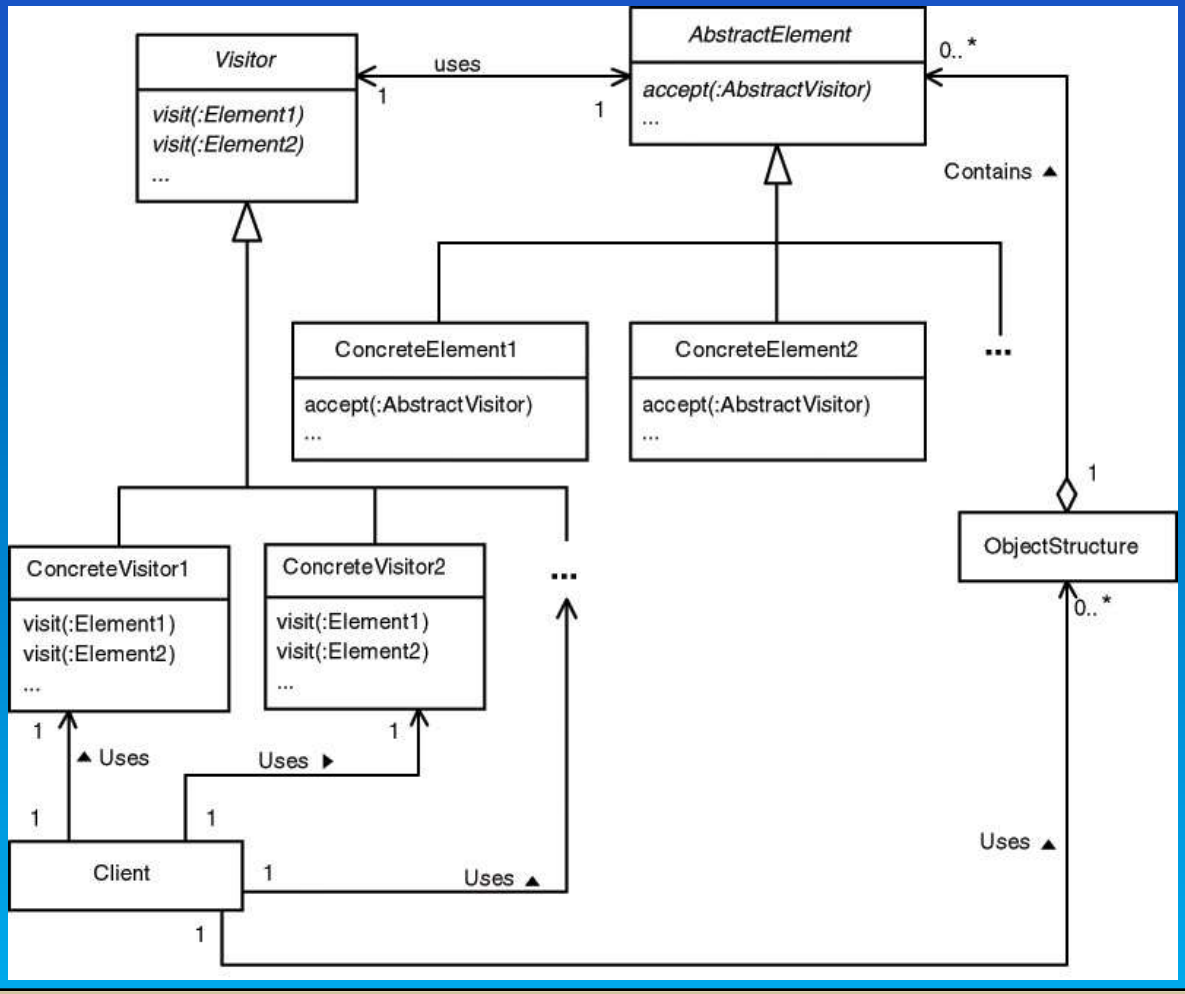
Visitor – diagram klas (double dispatch)



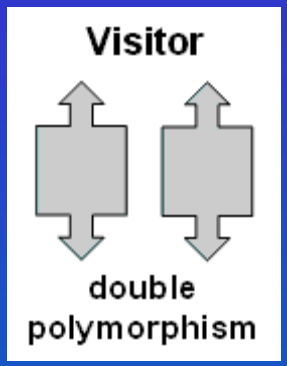


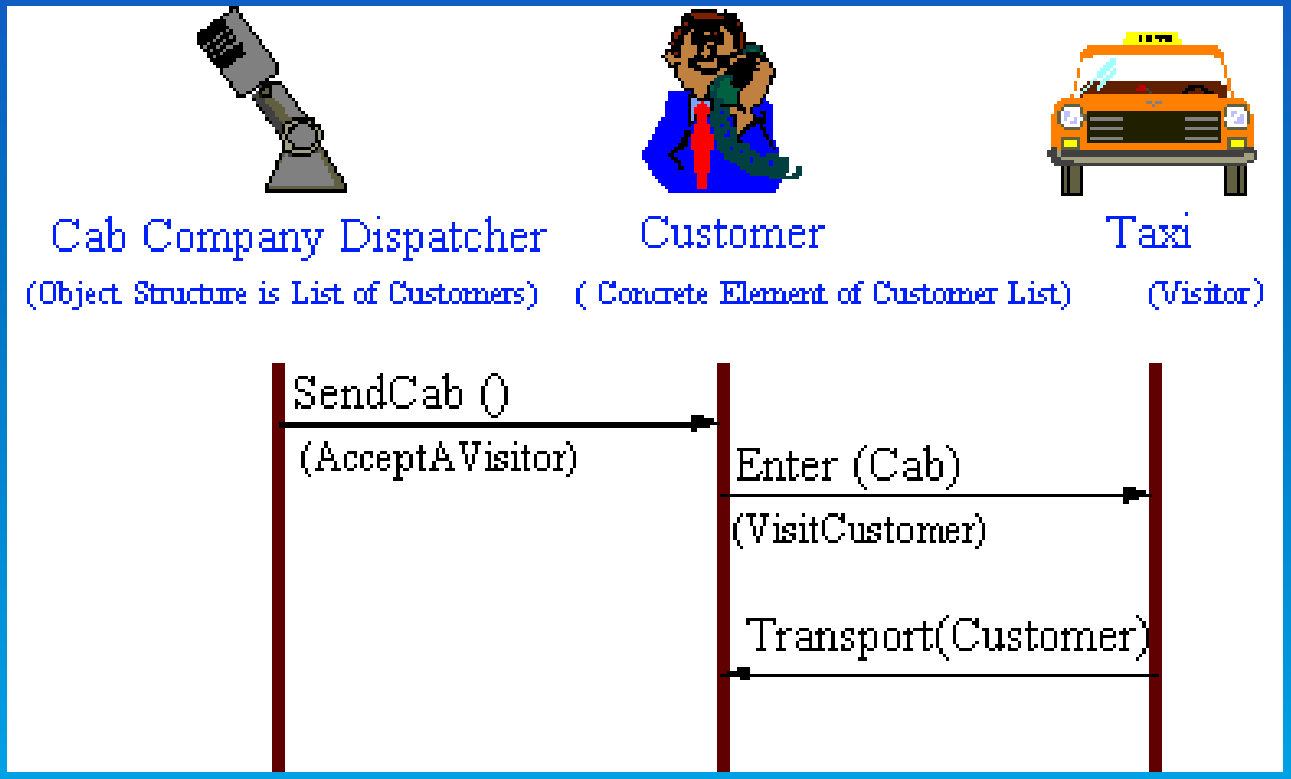
Visitor – diagram klas



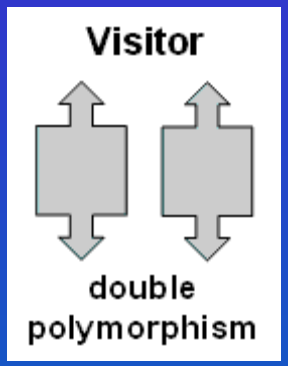


Visitor – przykład





Visitor – konsekwencje

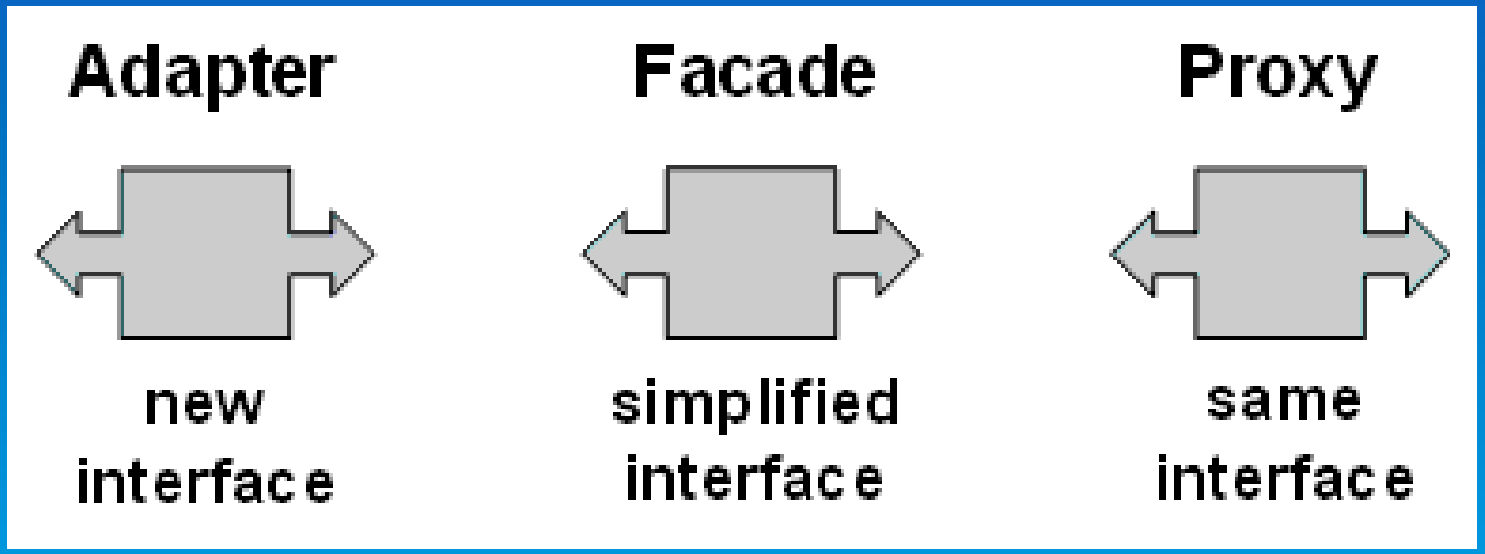


- Ułatwia dodawanie nowych operacji na strukturze obiektów. Konkretne instancje klas struktury nie mają zależności z klasami **visitora**.
- Logika nowej operacji jest umieszczona w jednej spójnej klasie tzn. konkretnej implementacji **visitora**.
- Jednie obiekt **visitora** przechowuje stan potrzebny do wykonania operacji na strukturze obiektów.
- Dodanie nowej konkretnej klasy do struktury obiektów wymaga rozbudowy każdego istniejącego **visitora**.
- Klasy struktury muszą udostępnić **visitorowi** odpowiedni dostęp do ich stanu do przeprowadzenia operacji.

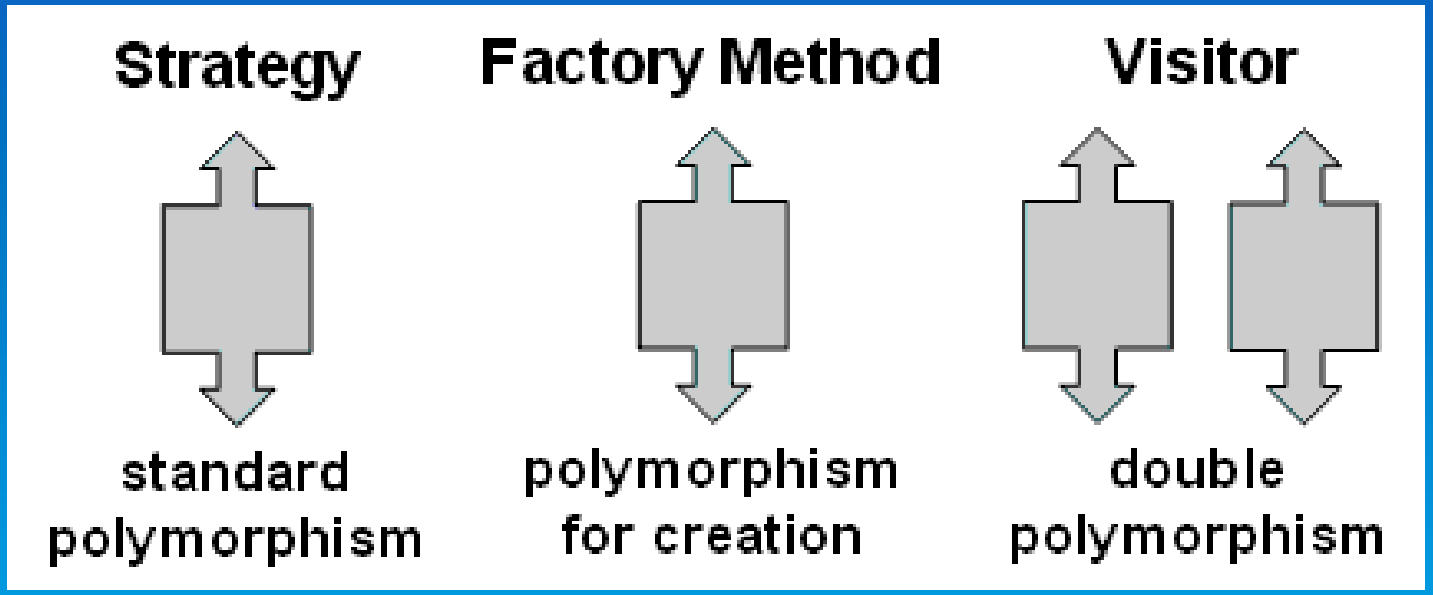
Zależności między wzorcami

- **iterator** może przemierzać **composite**
- **visitor** może wykonywać operację na **composite**
- **visitor** jest silniejszy niż **command** ponieważ może uruchamiać odpowiednią operację dla napotkanego rodzaju obiektu.

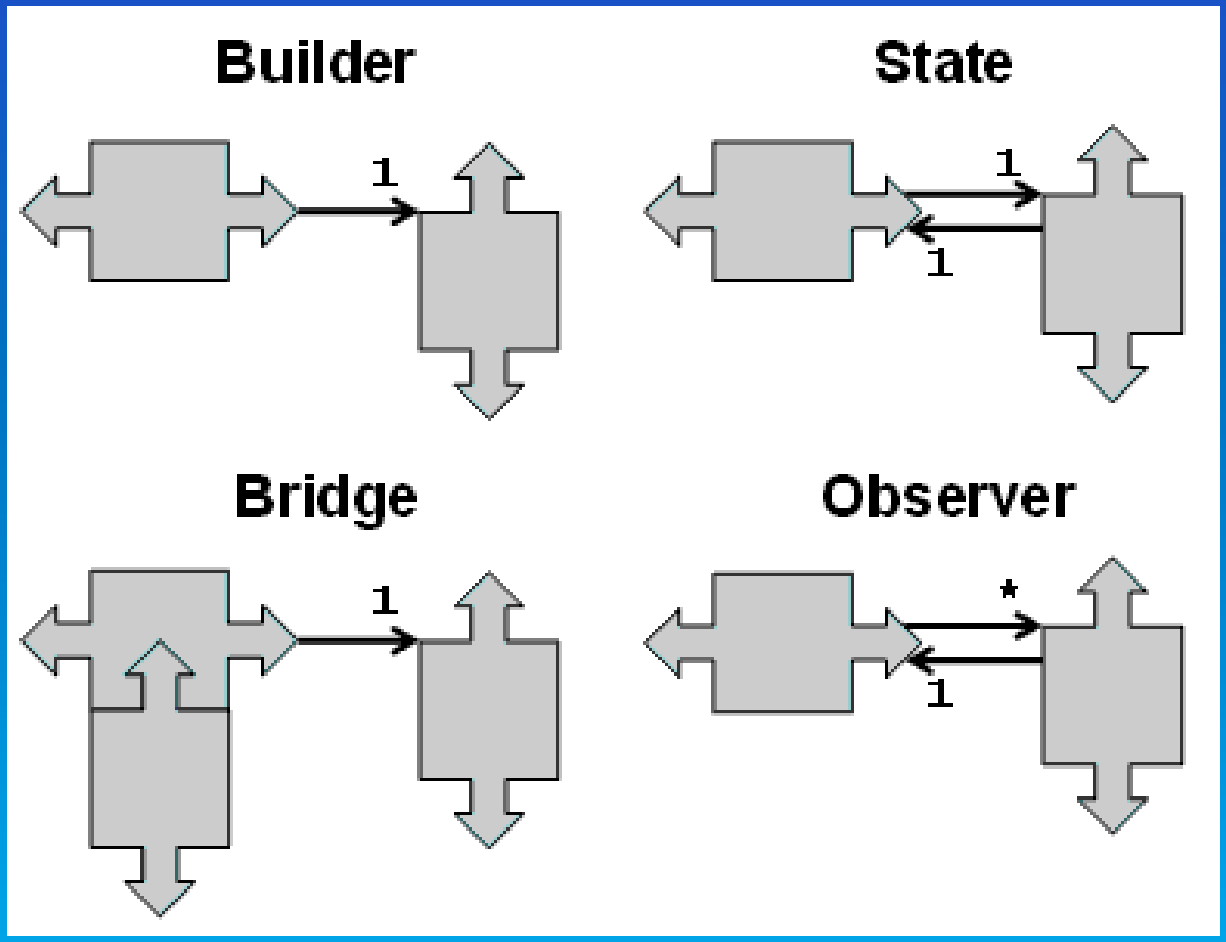
Podobieństwo strukturalne wzorców wrapper, delegacja, agregacja



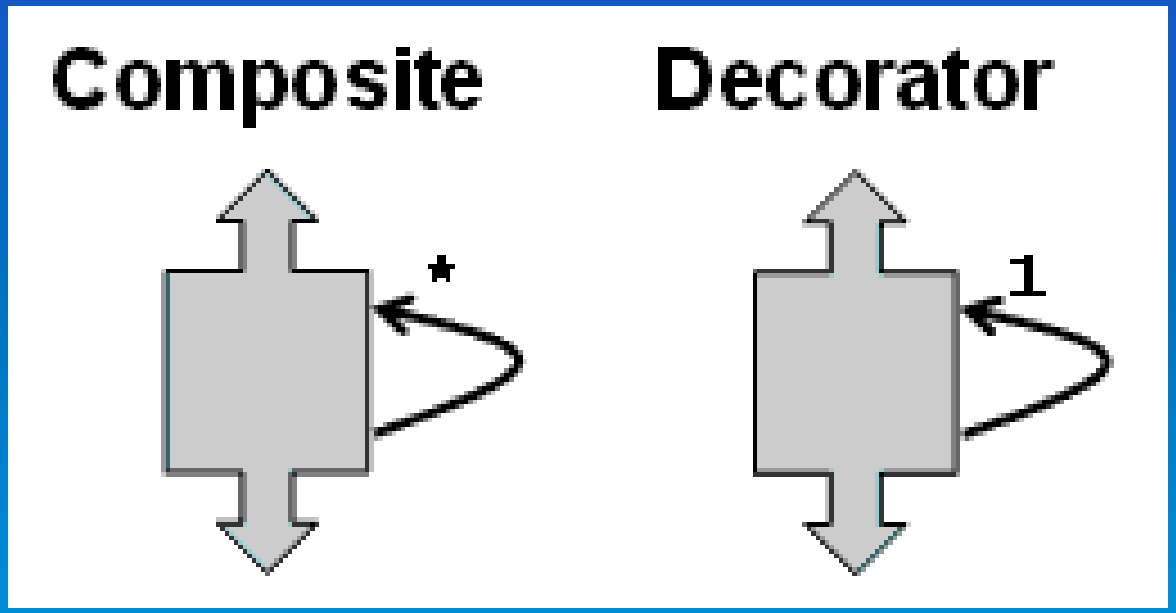
Podobieństwo strukturalne wzorców hierarchia dziedziczenia



Podobieństwo strukturalne wzorców osłona hierarchii dziedziczenia



Podobieństwo strukturalne wzorców rekursywna kompozycja



Podobieństwo strukturalne wzorców status pełnoprawnego obiektu