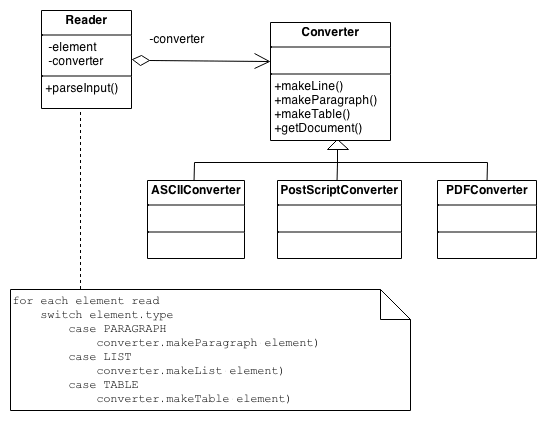
**Wzorce kreacyjne**

**Builder** (Budowniczy) – rozdziela sposób budowy obiektów od ich reprezentacji

Dokładny przebieg procesu budowania przedstawia znajdujący się obok diagram sekwencji:

1. klient używający wzorca konstruuje obiekt budowniczego,
2. klient konstruuje nadzorcę, przekazując mu referencję do obiektu budowniczego, z którego ma korzystać,
3. klient zleca skonstruowanie produktu,
4. nadzorca zleca budowniczemu wykonanie w odpowiedniej kolejności wszystkich czynności niezbędnych do stworzenia produktu,
5. klient pobiera gotowy produkt od budowniczego.



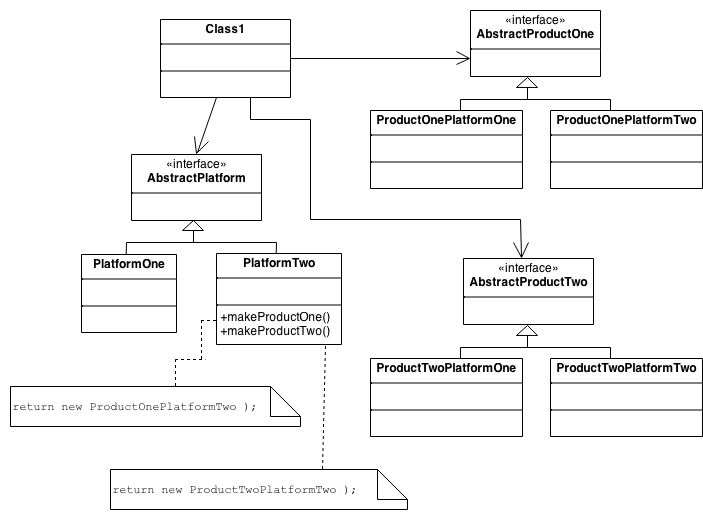
Zalety:

- Jasne oddzielenie budowy od reprezentacji obiektu

- Lepsza kontrola na procesem tworzenia obiektu

Budowniczy w UBER

**Abstract factory** (Fabryka abstrakcyjna) – interface do tworzenia rodzin jednego typu bez specyfikowania ich konkretnych klas.



Zalety:

- Możliwość ukrycia szczegółów implementacyjnych przed klientem

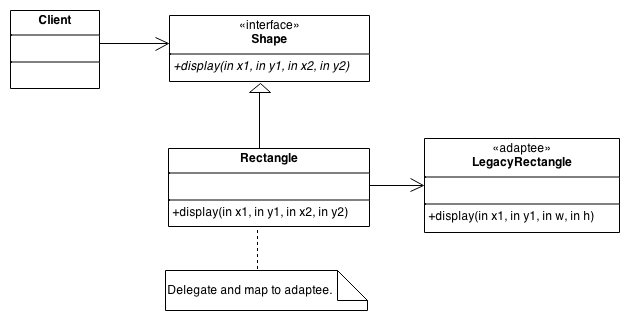
- Klient widzi tylko interface

Fabryka abstrakcyjna w UBER

**Wzorce strukturalne**

**Adapter** (wrapper) – umożliwia współpracę dwóm klasom o niekompatybilnych interfacach poprzez przekształcenie jednej z klas na interface drugiej klasy. Umożliwia też opakowanie interfacu w nowy.

Adapter:



Konsekwencje stosowania:

- brak możliwości adaptowania klasy wraz z jej podklasami,

- możliwość przeładowania metod obiektu adaptowanego.

Adapter w UBER

**Most** (bridge) - pozwala oddzielić abstrakcję obiektu od jego implementacji.

- odseparować implementację od interfejsu,

- poprawić możliwości rozbudowy klas, zarówno implementacji, jak i interfejsu (m.in. przez dziedziczenie),

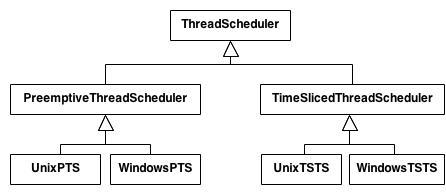
- ukryć implementację przed klientem, co umożliwia zmianę implementacji bez zmian interfejsu.

Zalety:

- decoupling interface obiektu

- poprawia możliwości rozbudowy

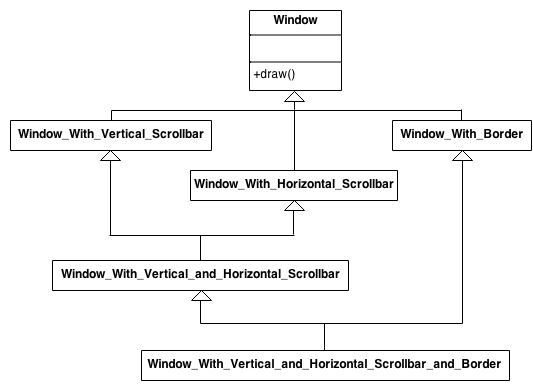
- ukrywa detale przed klientem



Most w UBER

**Dekorator** (Decorator) - pozwala na dodanie nowej funkcji do istniejących klas dynamicznie podczas działania programu. Wzorzec dekoratora polega na opakowaniu oryginalnej klasy w nową klasę "dekorującą". Zwykle przekazuje się oryginalny obiekt jako parametr konstruktora dekoratora, metody dekoratora wywołują metody oryginalnego obiektu i dodatkowo implementują nową funkcję.

Przykład użycia



Zalety:

- Zapewnia większą elastyczność niż statyczne dziedziczenie.

- Pozwala uniknąć tworzenia przeładowanych funkcjami klas na wysokich poziomach hierarchii.

- Dekorator i powiązany z nim komponent nie są identyczne.

- Powstawanie wielu małych obiektów.

Dekorator w UBER: