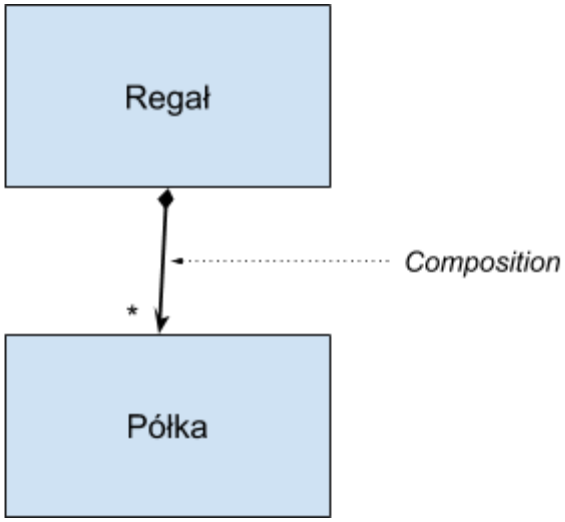
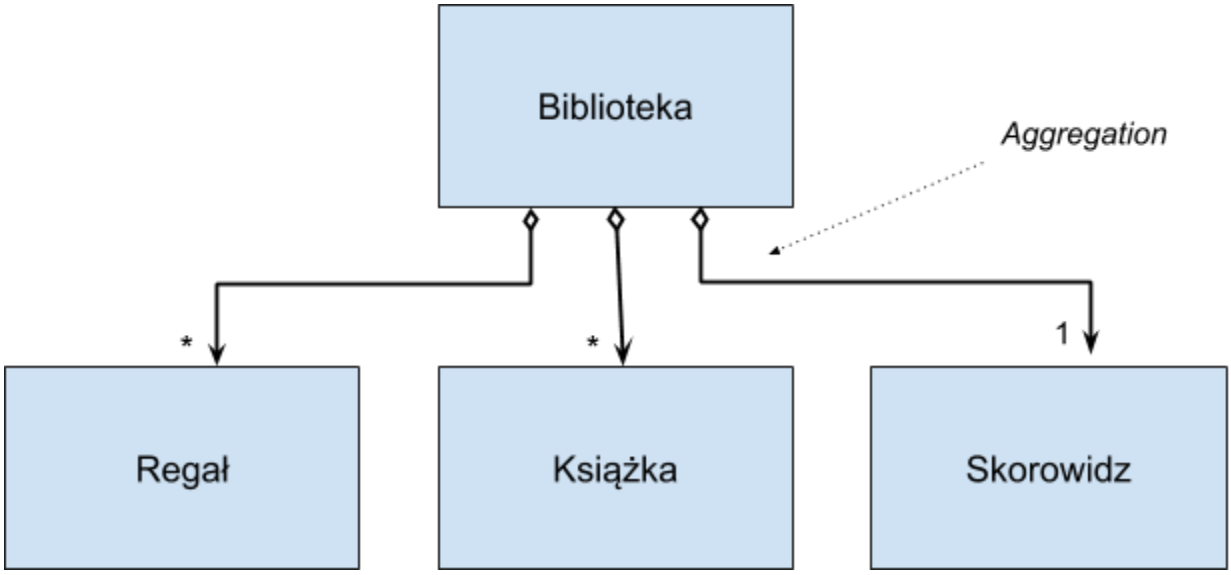
Design Patterns with UML (Unified Modeling Language)

1. Aggregation and composition




```
public Regal(int num) {

  this.polki = new Polka[NUM_POLKA];
    for (int i = 0; i < NUM_POLKA; i++) {
        this.polki[i] = new Polka(i);
    }

}
```

Regał składa się z półek. Regał nie składa się z książek.



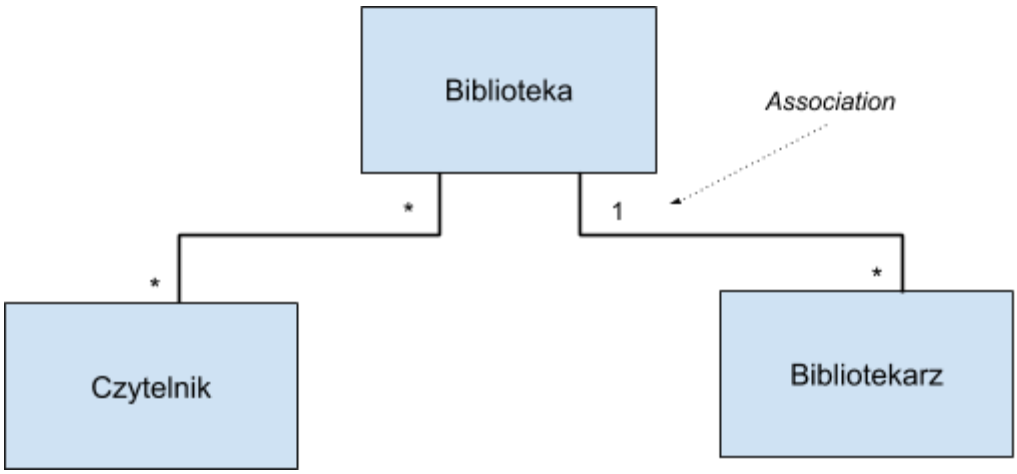
public class Biblioteka {

```
public Biblioteka() {

    this.regaly = new Regal[NUM_REGAL];
    for (int i = 0; i < NUM_REGAL; i++) {
        this.regaly[i] = new Regal(i);
    }
    this.skorowidz = new Skorowidz(this);
    this.ksiazki = new Ksiazka[MAX_KSIAZKA];
}

boolean addKsiazka(String tytul, String [] autorzy, String id, String wydawca, int
numStron)

boolean addRegal(int numPolek, int numPozycji) ...
```

}

Inicjujemy agregowane obiekty używając domyślnych wartości. Dodajemy metody do zmiany atrybutów (set*, add*, etc).

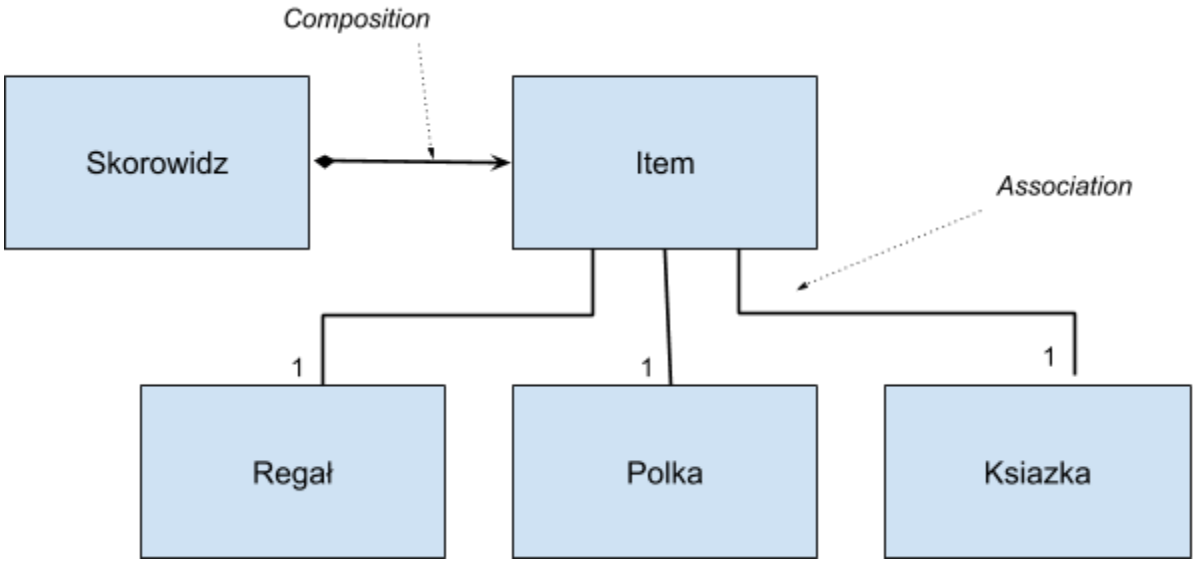
2. Associations




```
public class Biblioteka {
…
        private List<Czytelnik> listaCzytelnikow;
        private List<Bibliotekarz> listaBibliotekarzy;
        private Bibliotekarz bibliotekarz;
…

        public Czytelnik zapisz(Osoba o) ...

        public Bibliotekarz getBibliotekarz(Czytelnik c) ...
...

}

public class Czytelnik {
…
        private Biblioteka mojaBiblioteka;
        private List<Ksiazka> mojeKsiazki;
…

        public void setBiblioteka(Biblioteka b) ...

        public boolean wypozycz(String tytul, String [] autors) …

        public boolean oddaj(String tytul, String [] autors) ...
```

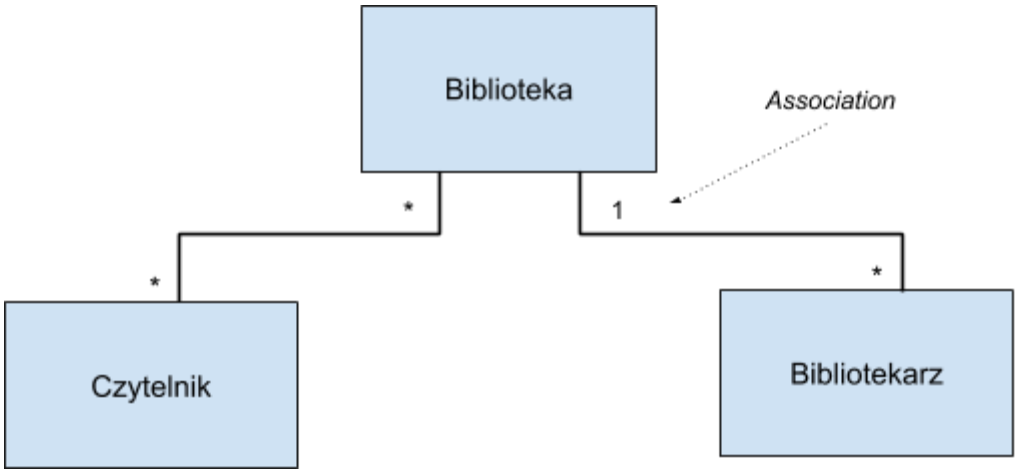
```
}

public class Bibliotekarz {
…
        private Biblioteka praca;
…

        public Bibliotekarz(String imie, String nazwisko, String pesel ) {
        }

        void pracujw(Biblioteka b)

        public Ksiazka wypozycz(String tytul, String [] autors) …

        public boolean oddaj(Ksiazka k) ...
...
}
```

Jak zaprojektować Skorowidz?



3. Konstruktor

   Parametry konstruktora: to co nieodzowne do istnienia obiektu (Bibliotekarz nie potrzebuje Regałów do istnienia). W konstruktorze obiekt tworzy/inicjuje swoje atrybuty. Czy potrzebna jest informacja z zewnątrz? Nie - tworzymy obiekt w konstruktorze, Tak - tworzymy metodę która ustawia atrybut (tzw. setter).



Biblioteka as a **singleton** (private constructor):

```
public class Biblioteka {

   private static oneBiblioteka = ...;

   private Biblioteka(...) {

...

   }

   static public Biblioteka getBiblioteka () {

       return oneBiblioteka;

   }

  public Czytelnik zapisz(Osoba o) {...}
```

```
}
```

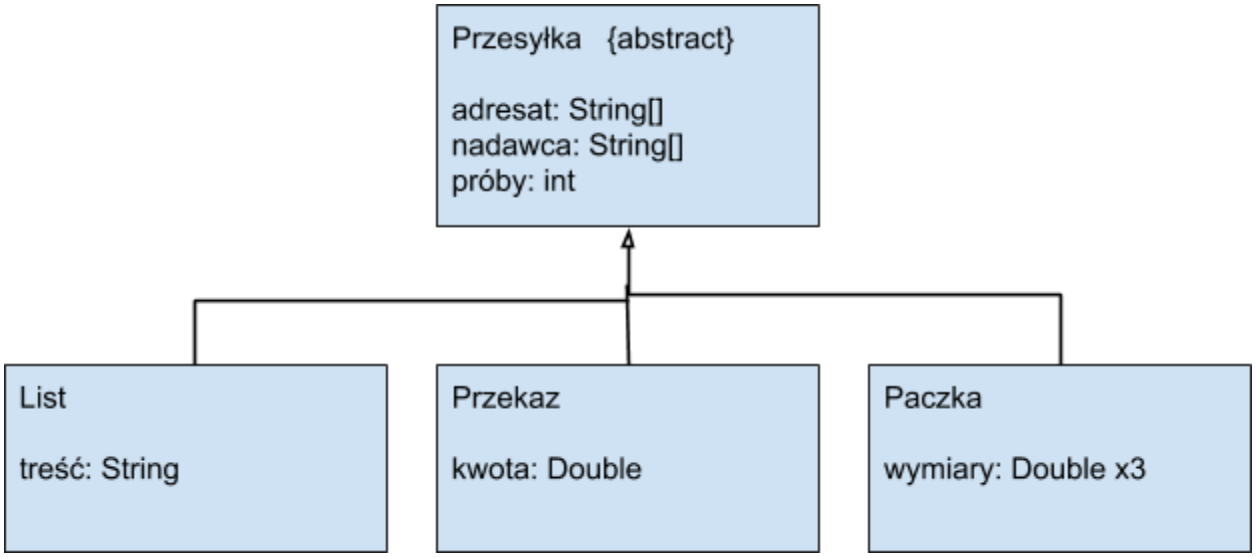
Wiele Bibliotek z kontrolą konstrukcji:

```
   static public Biblioteka znajdzBiblioteka (String adres) {
        ….
        return konkretnaBiblioteka;

   }
```

4. Inheritance

Szukamy wspólnych cech w obiektach, pamiętając że subclass is-a base class:

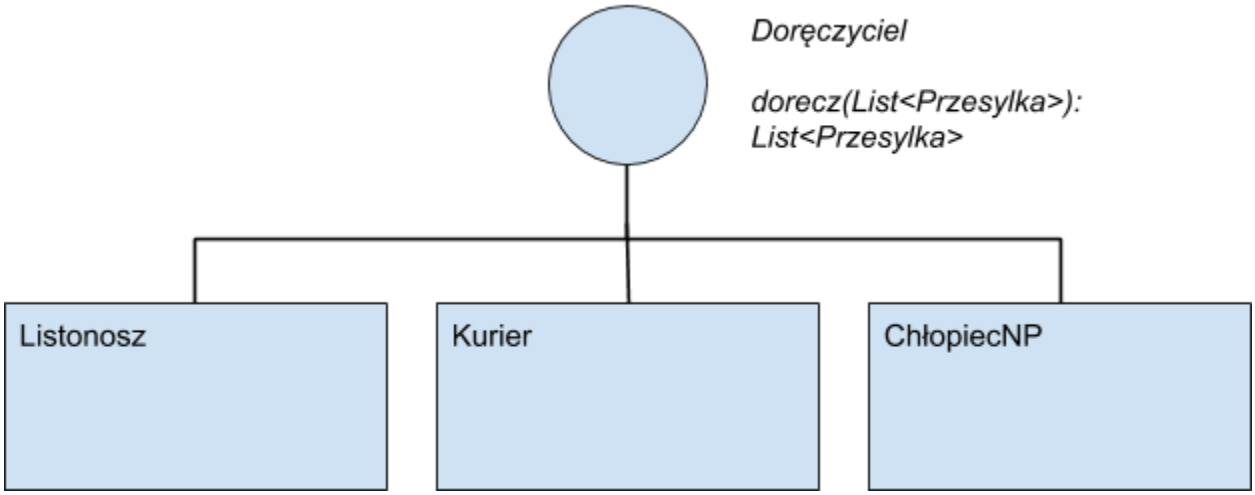
1) Wspólne atrybuty - abstract class:
List jest Przesyłką, Przekaz jest Przesyłką, itd



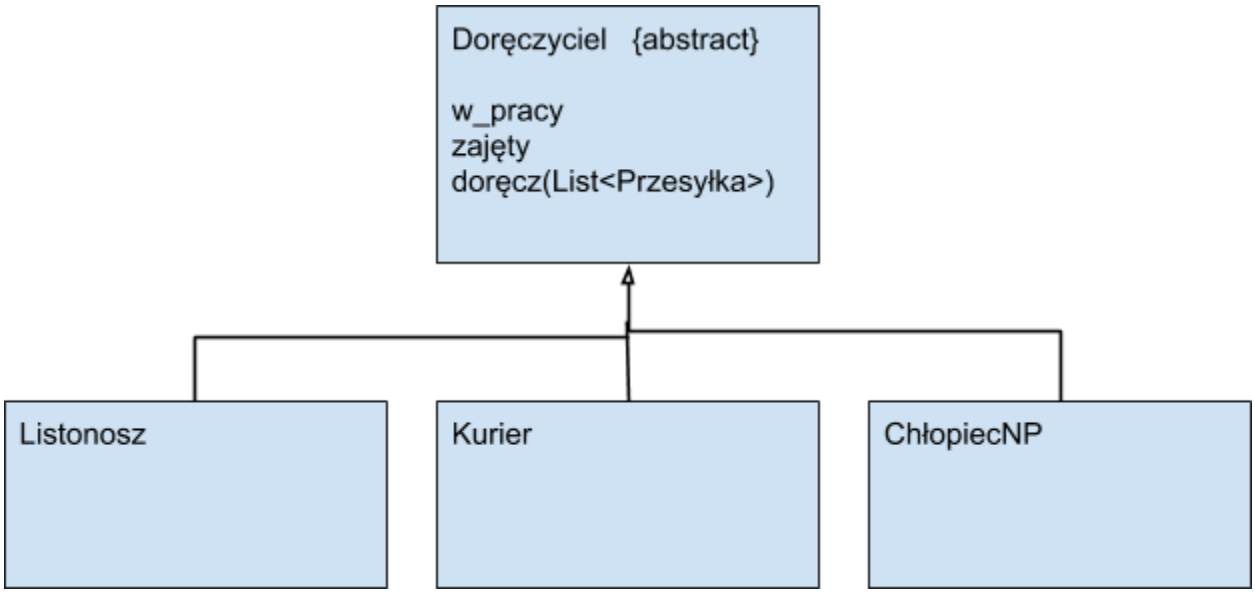
2) Wspólne metody (głównie **sygnatury,** ale implementacja też dozwolona od Java 8): interface

   Interface mówi co robi obiekt, a nie jak. Obiekty które implementują ten sam interfejs robią to samo, ale nie tak samo.

   Listonosz jest Doręczycielem

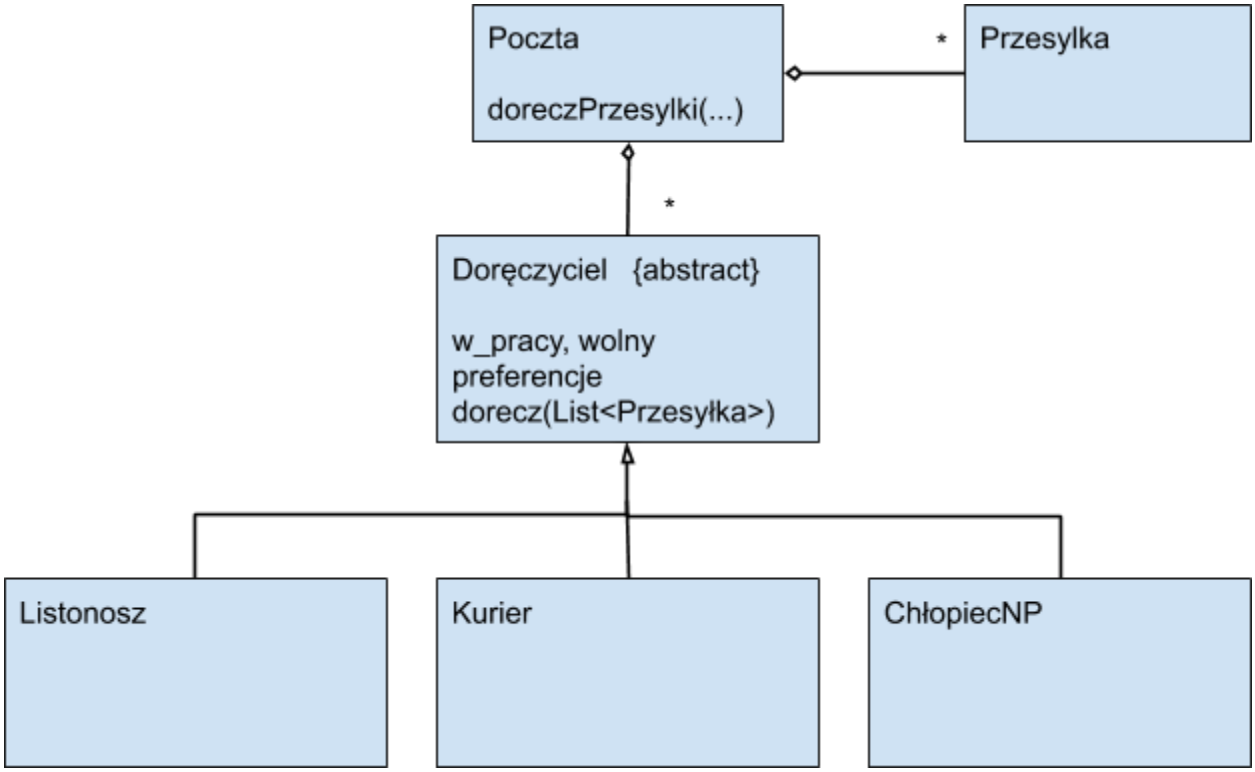


3) Wspólne metody i atrybuty, wspólne **implementacje** metod: class
   Obiekty robią to samo, i często tak samo (wspólna implementacja w klasie bazowej), metody abstract też dozwolone.

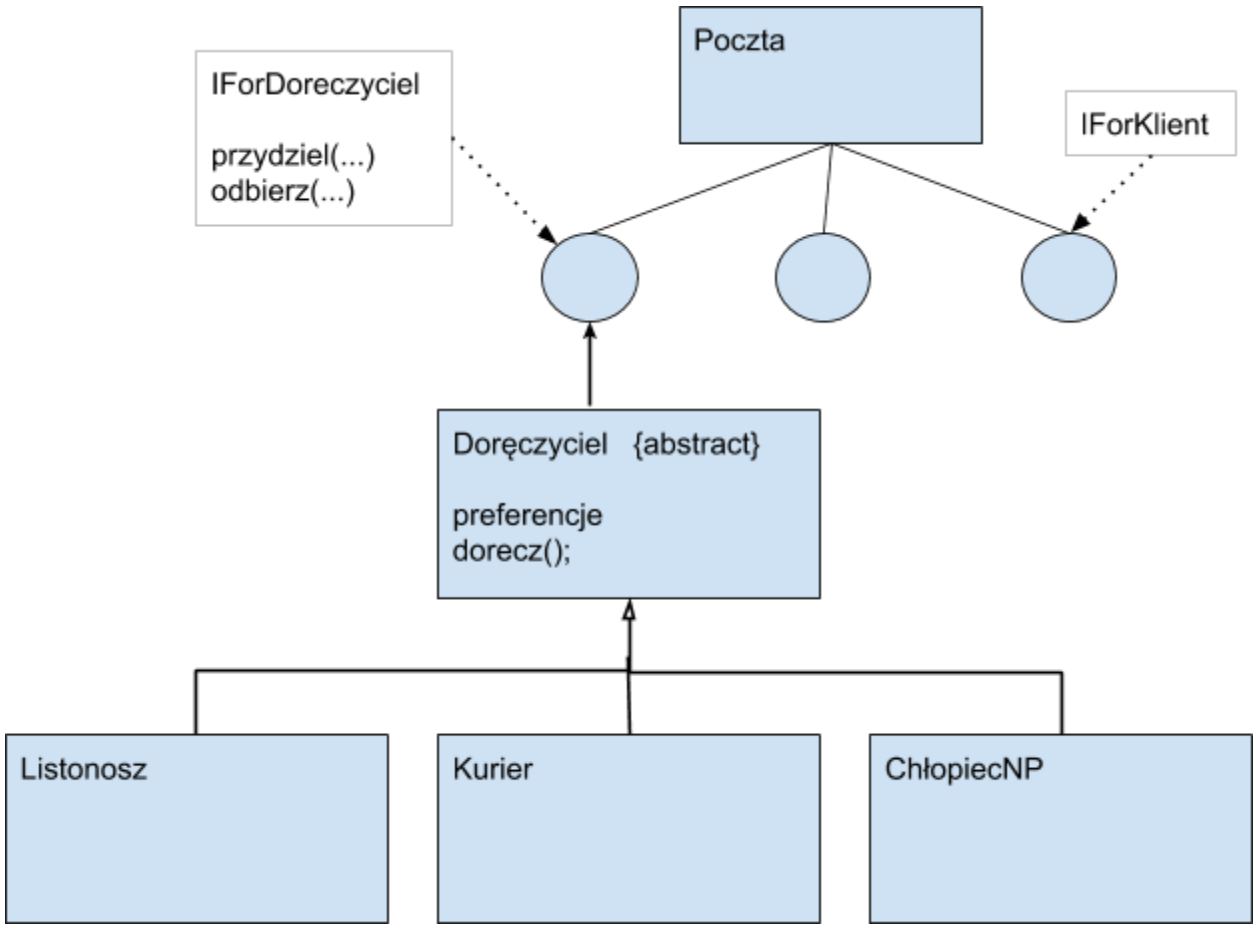


Jak połączyć te obiekty z Pocztą?

Wersja 1: Inicjatorem akcji jest Poczta



Zawsze jest więcej rozwiązań:
Wersja 2: Inicjatorami akcji są Doręczyciele:



Dostajemy obiekt typu Przesylka. Jak sprawdzić prawdziwy typ?

*If (przesylka **instanceof** List) {*

List list = (List)przesylka;

// uzywamy list

}