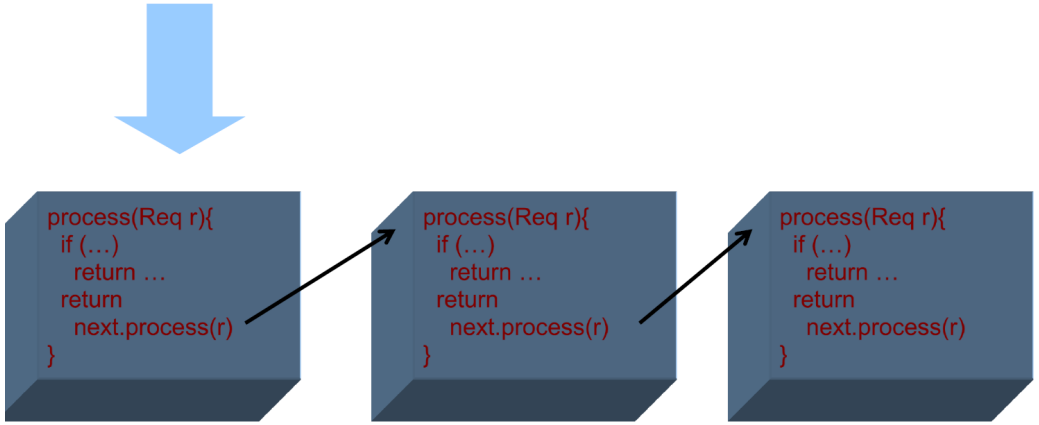
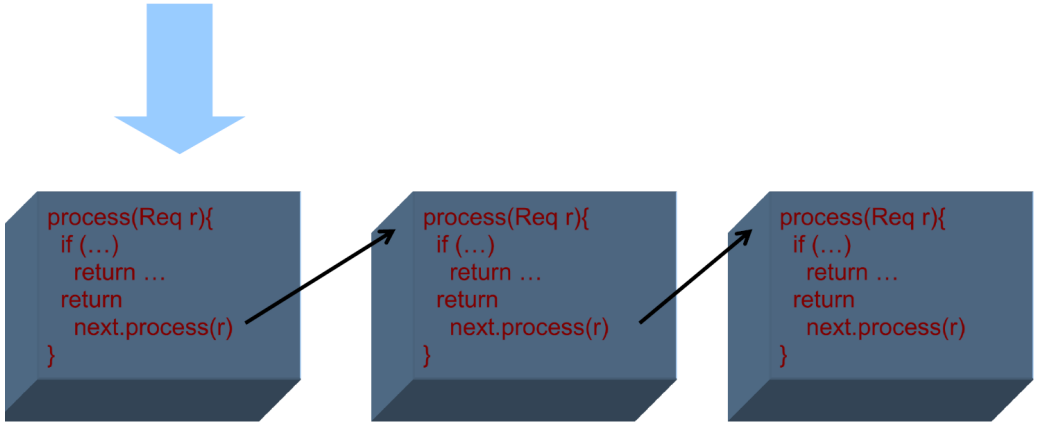
Łańcuch odpowiedzialności

Łańcuch odpowiedzialności



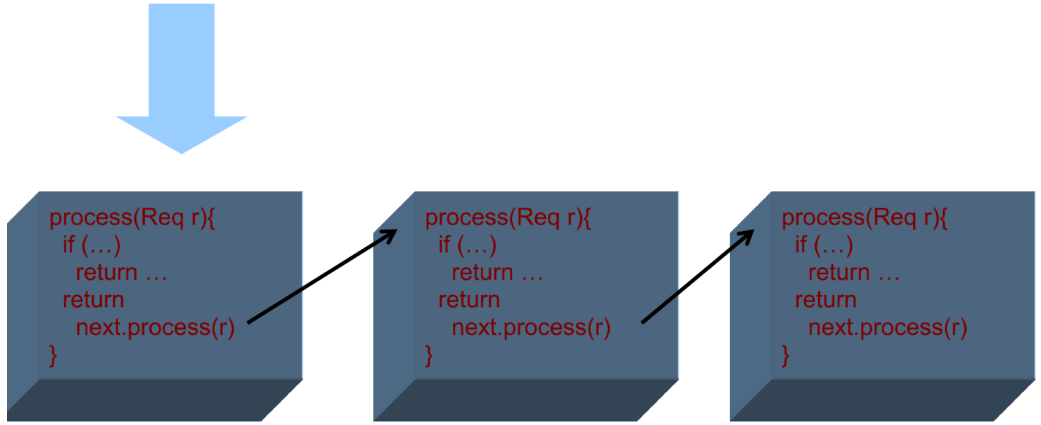
- formujemy *łańcuch odpowiedzialności* składający się z "bloczków" (ogniw)
  - metoda *SetNext* do łączenia ich ze sobą
- *nadawca* zaczyna przetwarzanie
- każdy bloczek stwierdza czy chce obsłużyć żądanie (*CanHandle*)
  - jeśli tak to przetwarza (*Process*) i kończy łańcuch (*nadawca* dostaje odpowiedź)
  - jeśli nie to przekazuje dalej

Łańcuch odpowiedzialności



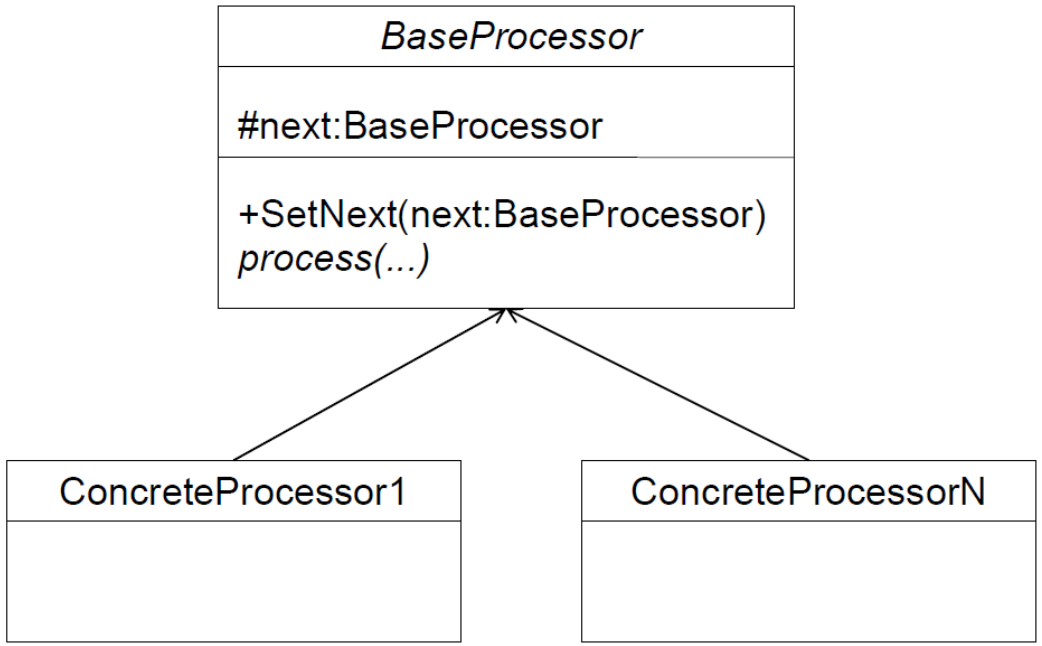
- warto podkreślić:
  - pierwszy odbiorca, który obsłuży żądanie, **przerywa łańcuch**
  - **kolejność ogniw ma znaczenie**

Łańcuch odpowiedzialności

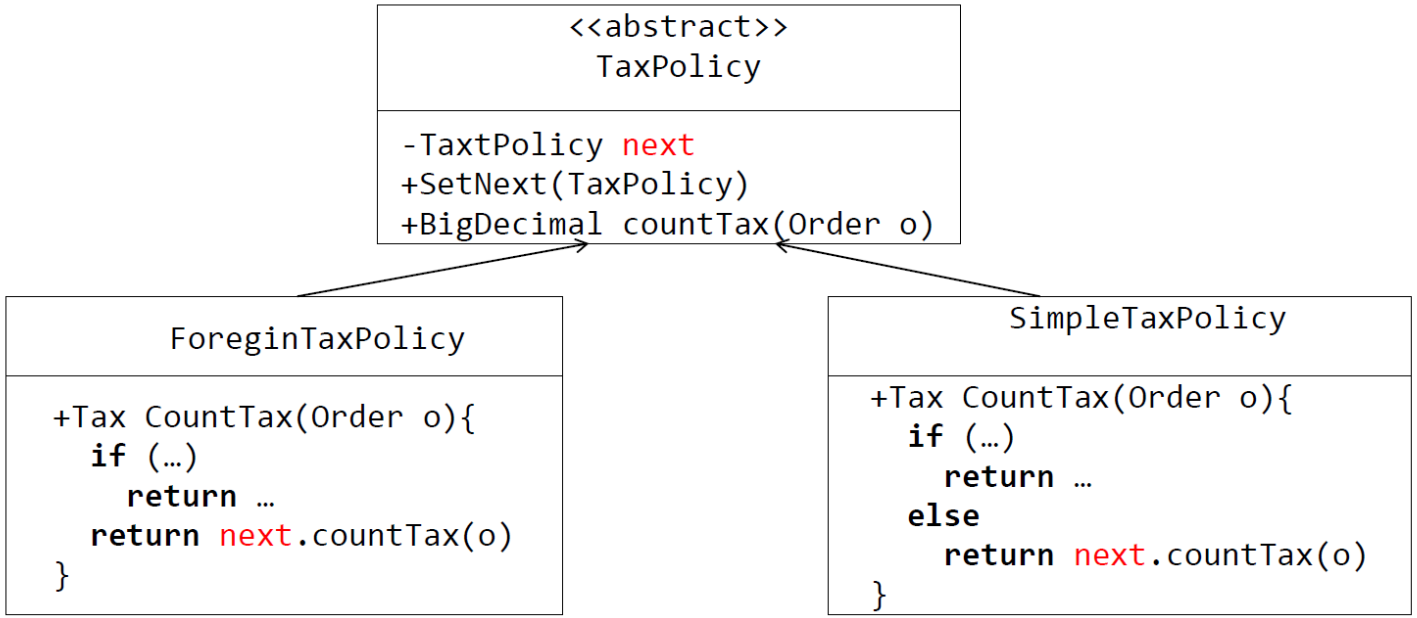


- *low coupling*:
  - nadawca wie tylko o jednym odbiorcy
  - każdy odbiorca wie tylko o kolejnym odbiorcy
  - nadawca nie wie kto obsłużył żądanie
    - nie powinno go to interesować - SRP
    - nie powinien mieć logiki sterującej

Łańcuch odpowiedzialności - struktura



Łańcuch odpowiedzialności - przykład biznesowy




```
TaxPolicy tp = new ForeginTaxPolicy();
tp.SetNext(new SimpleTaxPolicy());
...
tp.CountTax(order);
```

Łańcuch odpowiedzialności - przykład

```
public class SolutionExplorer
{
   private object _currentItem;
   public void ItemClicked()
   {
      if(_currentItem is FolderItem)
      {
         FolderHandler.Handle(_currentItem as FolderItem);
      }
      else if((_currentItem is ProjectItem)
      {
         FolderHandler.Handle(_currentItem as ProjectItem);
      }
      else if(_currentItem is FileItem)
      {
         FolderHandler.Handle(_currentItem as FileItem);
      }
   }
}
```

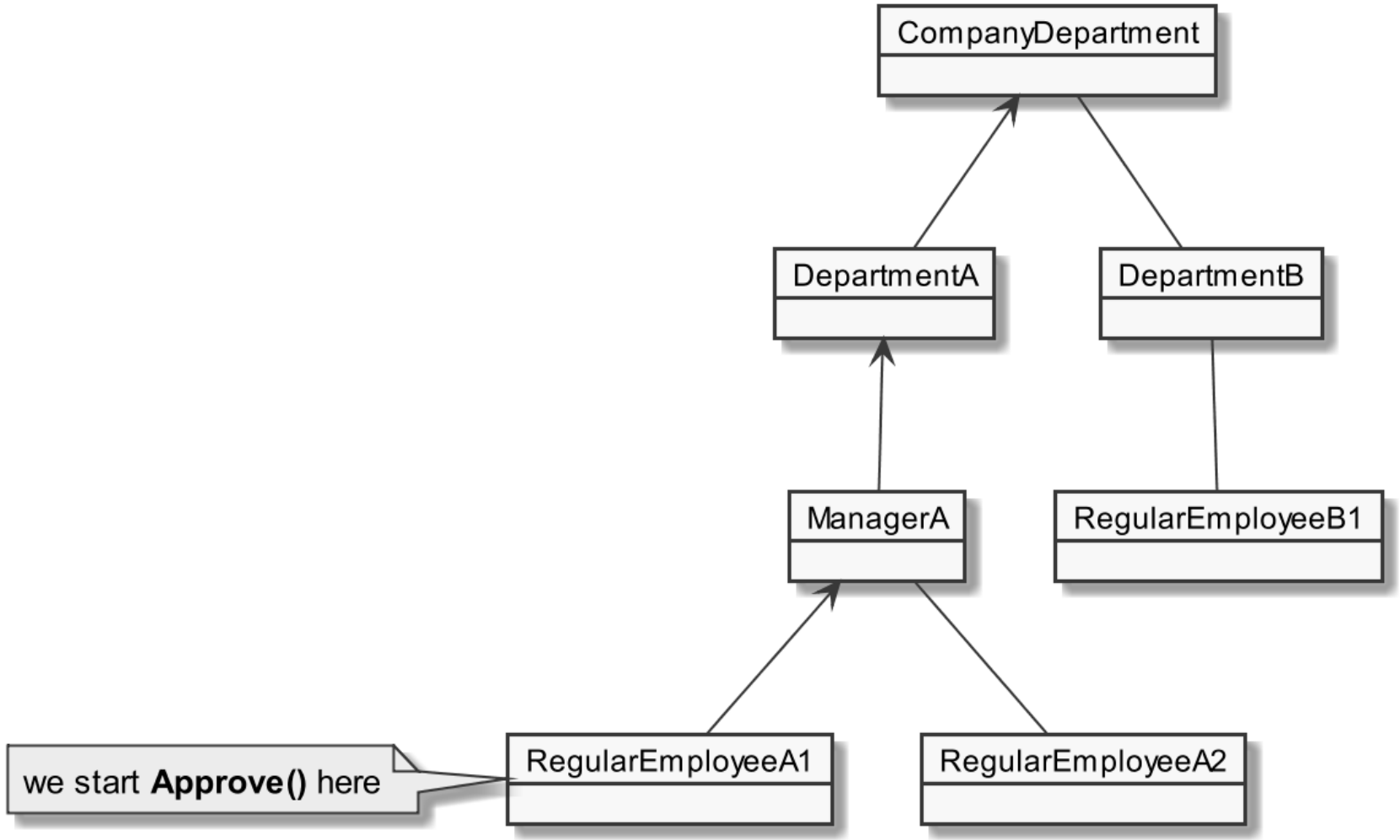
Łańcuch odpowiedzialności - przykład

```
public class SolutionExplorer
{
   private IHandler _handler;
   private object _currentItem;
   private void AddHandler(IHandler newHandler)
   {
      newHandler.SetNext(_handler);
      _handler = newHandler;
   }
   public void ItemClicked()
   {
      _handler.Handle(_currentItem);
   }
}
```

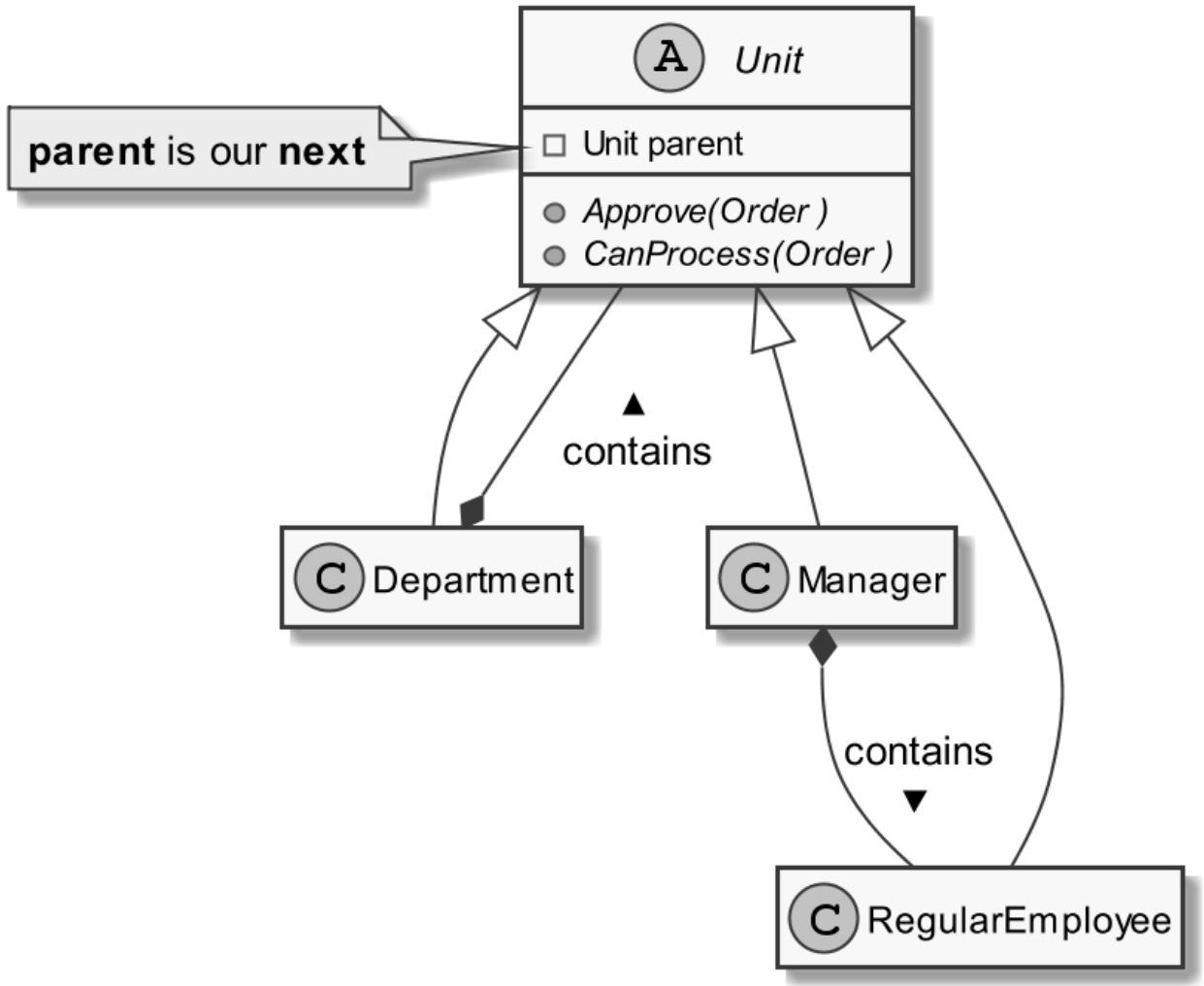
Łańcuch odpowiedzialności

- w pierwotnej wersji
  - **nie jest** to przetwarzanie procesowe (krok po kroku)!
  - **tak, jest** to po prostu "dynamiczny switch"
    - bez wiedzy/logiki u wołającego
    - otwarty na rozszerzenie - dodawanie/usuwanie kroków
    - możliwość/konieczność priorytetyzacji
  - *"żądanie jest przesyłane wzdłuż łańcucha obiektów, aż któryś z nich je obsłuży"*
- warianty
  - użycie wraz z *Composite* - efekt: operacja dla całego drzewa
  - metoda per operacja czy "obiekt żądania"
  - zewnętrzny zarządca o wysokiej kohezji
  - *multi handler* - zamiast jeden *Process*, każdy blok wykonuje *Process* lub przerywa
  - drzewo odpowiedzialności

Łańcuch odpowiedzialności + Composite



Łańcuch odpowiedzialności + Composite

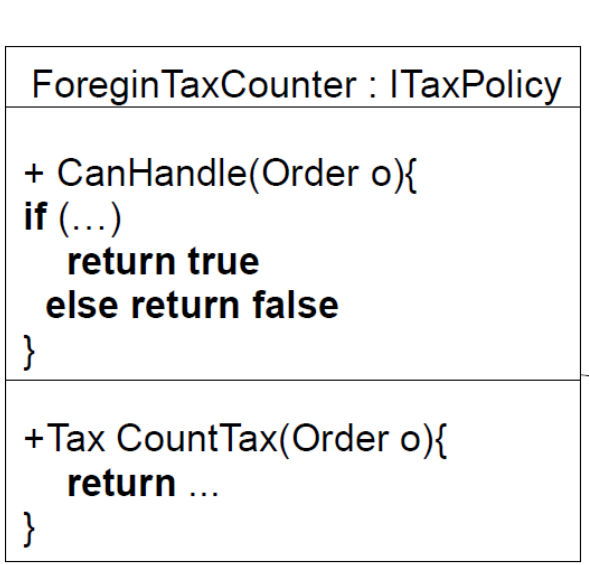


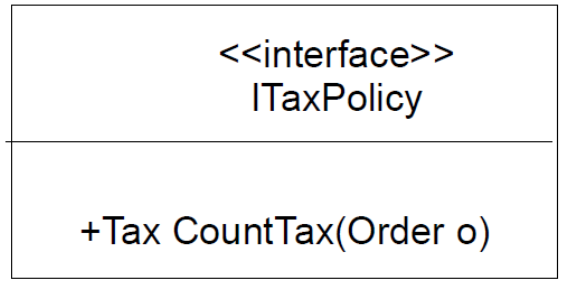
Łańcuch odpowiedzialności - obiekt żądania

```
public override void Process(IOrder order)
{
   if (CanProcess(order))
   {
      ProcessInternal(order);
      return;
   }
   Parent.Process(order);
}

private void ProcessInternal(IOrder order)
{
   switch (order)
   {
      case ApproveOrder approveOrder: ProcessApproveOrder(approveOrder);
         break;
      case CancelOrder cancelOrder: ProcessCancelOrder(cancelOrder);
         break;
      default:
         Parent.Process(order);
         break;
   }
}
```

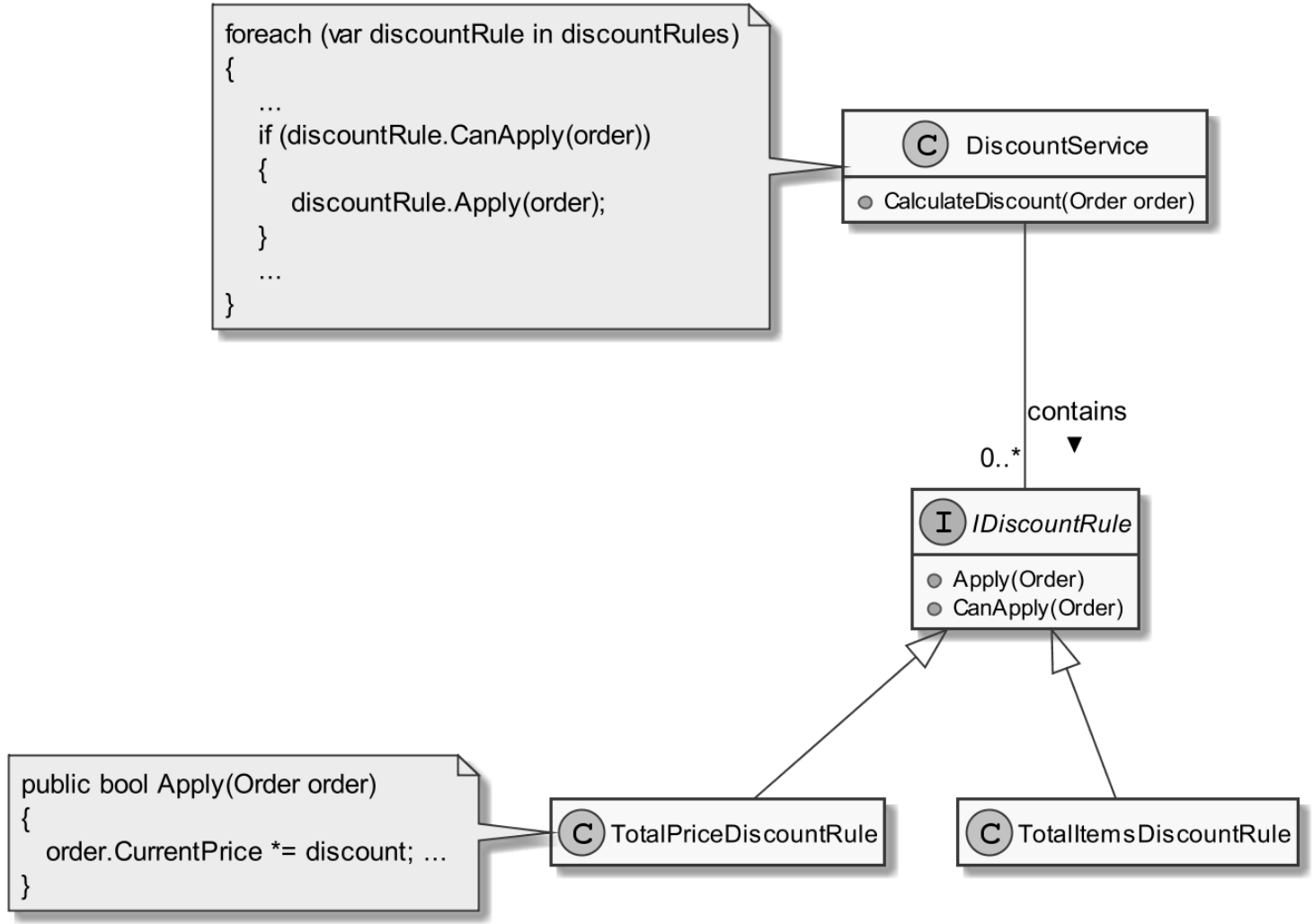
Łańcuch odpowiedzialności - zewnętrzny zarządca



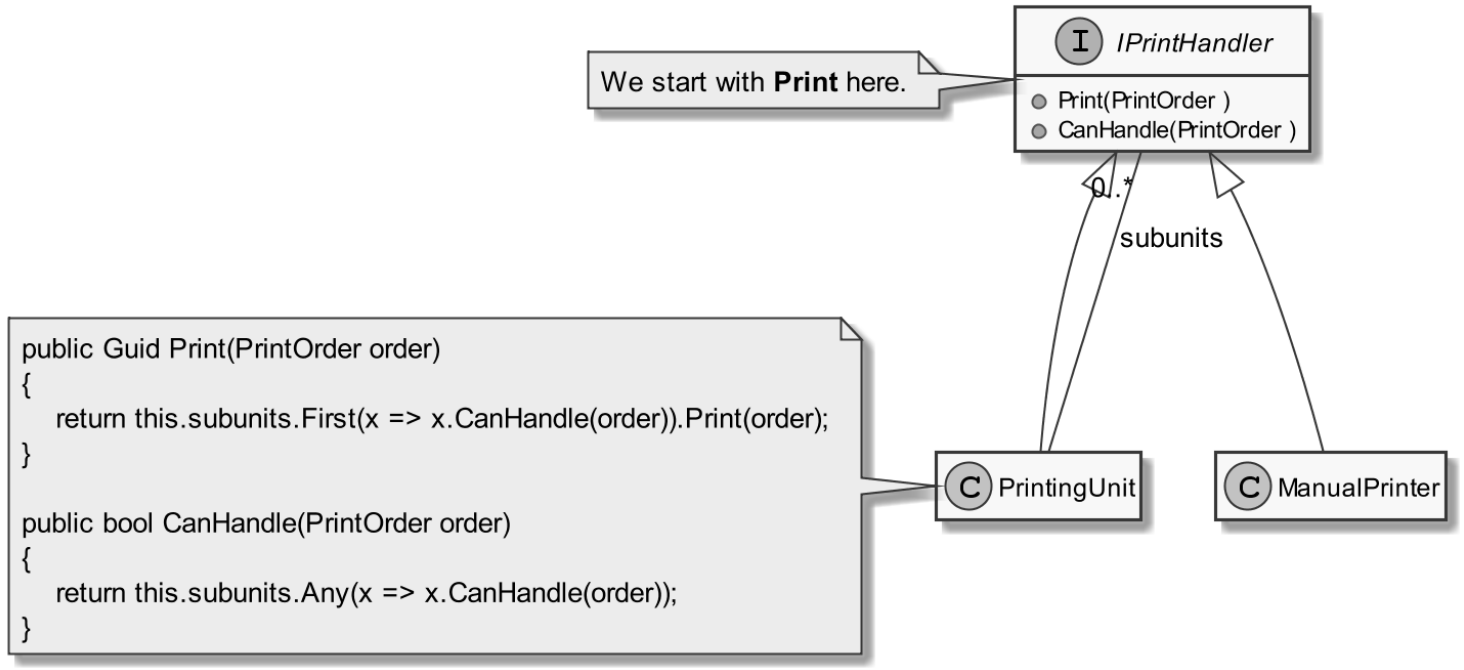



```
public class TaxPolicyChain : TaxPolicy
{
  private List<ITaxPolicy> _policies
  public Tax count(Order o)
  {
    foreach(TaxCounter in _policies)
    {
       if (tc.CanHandle(o))
        return tc.CountTax(o);
    }
    return new BigDecimal(0);
  }
}
```

Łańcuch odpowiedzialności - *multi handler*



Drzewo odpowiedzialności



Łańcuch odpowiedzialności

- Łańcuch zobowiązań vs Dekorator - bardzo podobne struktury klas
  - Oba wzorce bazują na rekursywnej kompozycji celu przekazania obowiązku wykonania przez ciąg obiektów

Łańcuch odpowiedzialności

- Łańcuch zobowiązań vs Dekorator - bardzo podobne struktury klas
  - Oba wzorce bazują na rekursywnej kompozycji celu przekazania obowiązku wykonania przez ciąg obiektów
- Istnieją jednak kluczowe różnice:
  - obsługujący Łańcuch zobowiązań mogą wykonywać działania niezależnie od siebie
  - mogą również zatrzymać dalsze przekazywanie żądania na dowolnym etapie
  - z drugiej strony, Dekoratory mogą rozszerzać obowiązki obiektu zachowując zgodność z interfejsem bazowym
  - dodatkowo, dekoratory nie mają możliwości przerwania przepływu żądania

Łańcuch odpowiedzialności - przykłady

- **`_Intro`** - switchologia
- **`Main`** - wzorzec
- **`Multihandler`** - kilka po kolei
- **`OrderObject`** - a'la Command
- **`Tree`** - Composite + ChoR