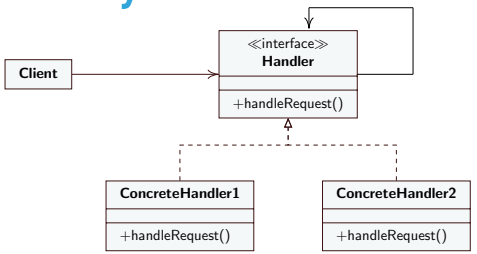
Chain of Responsibility

- **Intento**: evitare di accoppiare il mandante di una richiesta col suo ricevente, dando così a più di un oggetto la possibilità di gestire la richiesta. Concatena gli oggetti riceventi e passa la richiesta lungo la catena fino a che un oggetto la gestisce
- Motivazione
  - Consideriamo un'interfaccia utente in cui l'utente può chiedere aiuto su parti dell'interfaccia. Per es. un bottone può fornire informazioni di aiuto. Se non esiste un'informazione specifica allora si dovrebbe fornire il messaggio di aiuto del contesto più vicino (per es. la finestra)
  - Problema: l'oggetto che fornirà il messaggio d'aiuto non è conosciuto dall'oggetto che effettua la richiesta iniziale
  - Disaccoppiare mandante e ricevente

Chain Of Responsibility: Soluzione

- Struttura



- Partecipanti
  - **Handler** definisce l'interfaccia per gestire le richieste
  - **ConcreteHandler** gestisce la richiesta di cui è responsabile, può accedere al suo successore, inoltra la richiesta se non può gestirla
  - **Client** inizia effettuando la richiesta a un ConcreteHandler
- La richiesta si propaga lungo la catena finché un ConcreteHandler la gestisce

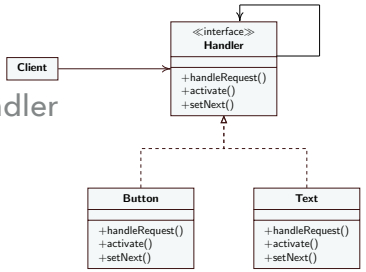
Chain Of Responsibility: Conseguenze

- Riduce l'accoppiamento: chi effettua una richiesta non conosce il ricevente
- Un oggetto della catena non deve conoscere la struttura della catena
- Gli oggetti handler mantengono solo il riferimento al successore (non a una lista di oggetti)
- Si aggiunge flessibilità nella distribuzione di responsabilità agli oggetti. Si può cambiare o aggiungere responsabilità nella gestione di una richiesta cambiando la catena a runtime
- Non vi è garanzia che una richiesta sia gestita, poiché non c'è un ricevente esplicito, la richiesta potrebbe arrivare alla fine della catena senza essere gestita

Chain Of Responsibility: Implementazione

- Handler o ConcreteHandler possono definire i link per avere il successore della catena
- Una gerarchia già esistente può essere usata, anziché ridefinire i link (il design pattern Composite è usato insieme a Chain)
- Ciascun tipo di richiesta può corrispondere a un'operazione, in questo caso il set di richieste è definito dall'Handler, oppure
- Ciascun tipo di richiesta corrisponde a un codice e solo un'operazione è definita dall'Handler. Il set di richieste non è fissato: richiedenti e riceventi prendono accordi sulla codifica delle richieste; si hanno più controlli a runtime

Chain: Esempio



- Button e Text sono ConcreteHandler

```
public interface Handler {
    public void handleRequest();
    public void activate();
    public void setNext(Handler next);
}

public class Text implements Handler {
    private Handler next;
    public Text(Handler h) {
        next = h;
    }
    @Override public void handleRequest() {
        if (next != null) next.handleRequest();
        else System.out.println("Text: assistenza");
    }
    @Override public void activate() {
        System.out.println("sono un frammento di testo");
    }
    @Override public void setNext(Handler next) {
        this.next = next;
    }
}
```

```
public class Client {
    public static void main(String[] args) {
        Text t = new Text(null);
        Button b = new Button(t);
        t.activate();
        b.activate();
        b.handleRequest();
        b.handleRequest();
    }
}
```