Il pattern Decorator

a cura di **Angelo Furfaro**

da “Design Patterns”, Gamma et al.

“Patterns in Java”, Grand

Dipartimento di Ingegneria Informatica Elettronica Modellistica e Sistemistica
Università della Calabria, 87036 Rende(CS) - Italy
Email: a.furfaro@dimes.unical.it
Web: http://angelo.furfaro.dimes.unical.it

Decorator

Classificazione

- Scopo: strutturale
- Raggio d'azione: basato oggetti

Altri nomi

Wrapper

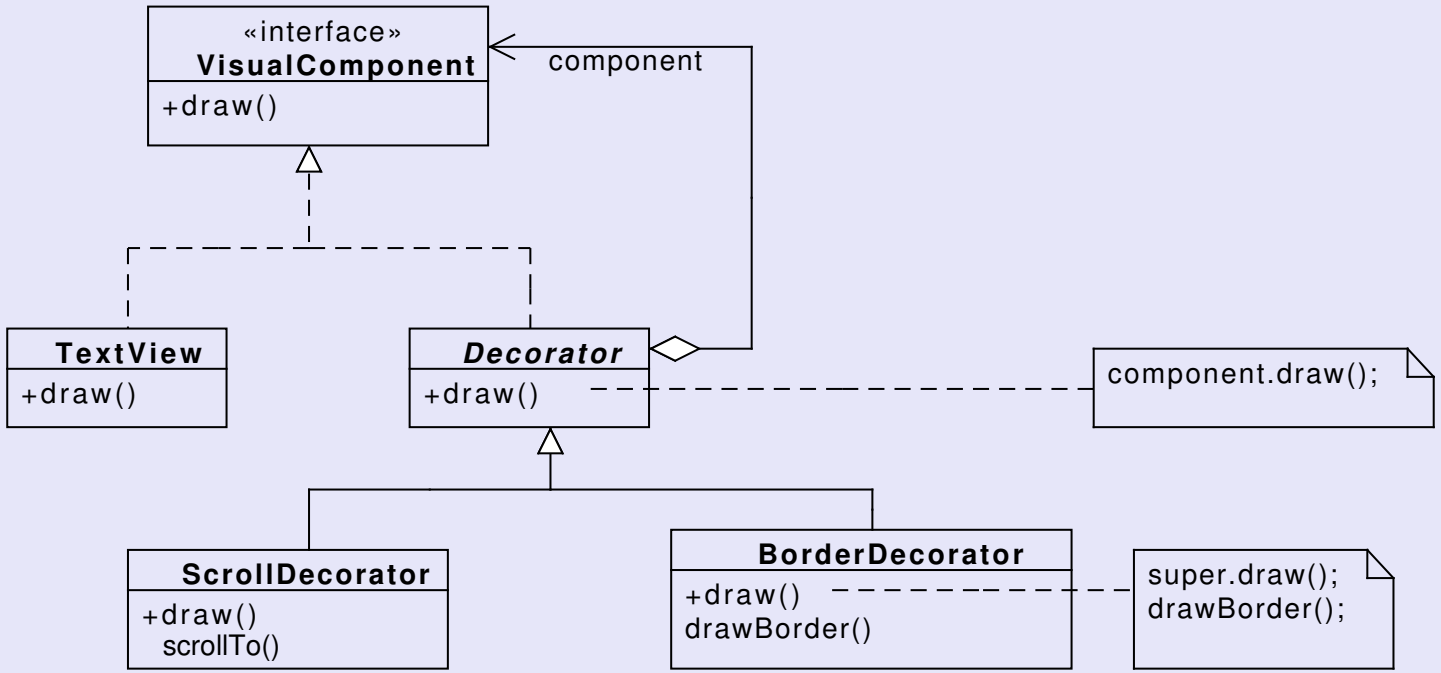
Scopo

- Aggiungere dinamicamente responsabilità ad un oggetto
- Decorator fornisce un'alternativa flessibile all'uso dell'ereditarietà come strumento per l'estensione delle funzionalità

Problema

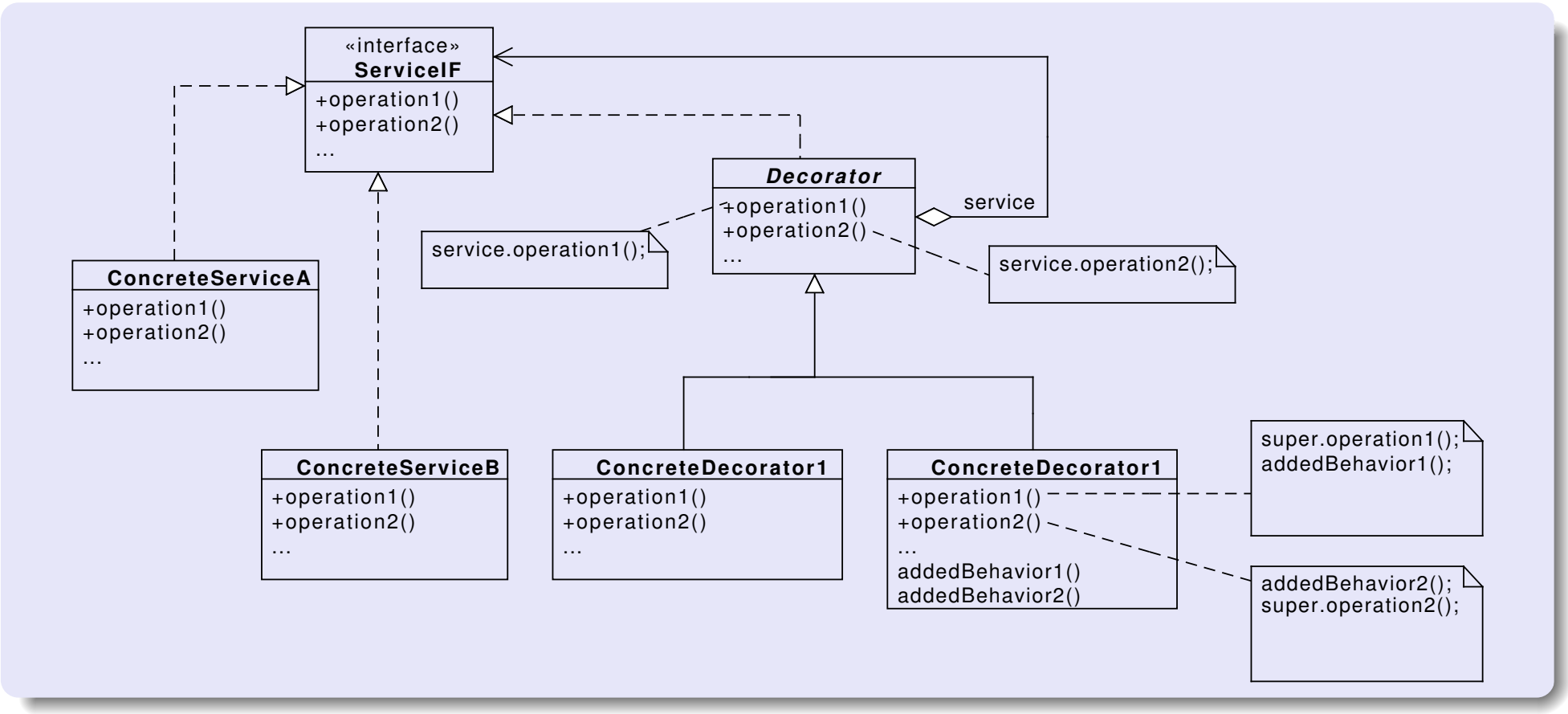
- Talvolta si vogliono aggiungere responsabilità a singoli oggetti e non a un'intera classe.
- Ad esempio una libreria per la realizzazione di interfacce utente dovrebbe consentire di aggiungere bordi o altri elementi quali barre di scorrimento a ciascun componente grafico.
- Una possibile soluzione consiste nel ricorrere all'ereditarietà creando sottoclassi che aggiungono il comportamento e/o le funzionalità aggiuntive desiderate. Quest'approccio non è flessibile ed è difficile comporre più *estensioni* di comportamento.
- Un approccio più flessibile consiste nel racchiudere il componente da decorare dentro un altro che sarà responsabile di disegnare il bordo o aggiungere le barre di scorrimento.
- L'oggetto contenitore è detto *decoratore*. Il decoratore ha la stessa interfaccia dell'elemento decorato in modo da essere trasparente al client.
- Il decorator può svolgere azioni aggiuntive prima o dopo aver trasferito la richiesta all'oggetto decorato.

Soluzione



- L'interfaccia `VisualComponent` definisce il tipo generico di componenti visuali. La classe `TextView` consente di visualizzare del testo in una finestra.
- La classe astratta `Decorator` semplicemente inoltra le richieste al componente incapsulato.
- `BorderDecorator` e `ScrollDecorator` consentono di rispettivamente di aggiungere bordi e barre di scorrimento.

Struttura


«interface»
ServiceIF
+operation1()
+operation2()
...
Decorator
+operation1()
+operation2()
...
service
service.operation1();
service.operation2();
ConcreteServiceA
+operation1()
+operation2()
...
ConcreteServiceB
+operation1()
+operation2()
...
ConcreteDecorator1
+operation1()
+operation2()
...
ConcreteDecorator1
+operation1()
+operation2()
...
addedBehavior1()
addedBehavior2()
super.operation1();
addedBehavior1();
addedBehavior2();
super.operation2();

Struttura

Partecipanti

- **ServiceIF** (`VisualComponent`):
  definisce l'interfaccia comune per gli oggetti ai quali possono essere dinamicamente aggiunte nuove responsabilità.
- **ConcreteService**X (`TextView`):
  definisce un oggetto al quale possono essere aggiunte ulteriori responsabilità.
- **Decorator**:
  mantiene un riferimento ad un oggetto di tipo `ServiceIF` e, al contempo, implementa l'intefaccia `ServiceIF`.
- **ConcreteDecorator** (`ScrollDecorator`,`BorderDecorator`):
  aggiunge responsabilità al componente.

Conseguenze

- ☺ Fornisce una maggiore flessibilità rispetto all'ereditarietà: consente di modificare dinamicamente il comportamento degli oggetti aggiungendo o rimuovendo *decoratori*.
- ☺ Utilizzando differenti combinazioni di oggetti decoratori si possono creare molti comportamenti differenti. Per ottenere lo stesso effetto con l'ereditarietà sarebbe necessario introdurre una differente sottoclasse per ciascuna combinazione di *decoratori*.
- Un progetto che usa il Decorator spesso porta a sistemi composti da molti oggetti simili interconnessi tra di loro. Sebbene tali relazioni siano semplici da comprendere per il progettista, agli altri possono risultare difficili da comprendere.
- ☹ Un decoratore e l'oggetto decorato non sono identici. Il decoratore si può utilizzare trasparentemente al posto dell'oggetto decorato dal punto di vista del comportamento ma non dell'identità.
- ☹ Occorre porre attenzione a come si compongono i decoratori. Ad esempio evitare dipendenze circolari tra di essi.

Uso nelle API di Java

Swing

- È noto che una `JTextArea` (`JComponent`) fornisce uno spazio su una GUI dove visualizzare linee di testo. Quando il testo supera una certa dimensione, può essere utile disporre di barre di scorrimento per poter vedere tutte le parti del testo (in orizzontale o in verticale)
- Nelle precedenti versioni della libreria di Java, la gestione delle barre di scorrimento era a carico della text area. Nelle versioni più recenti, questo "servizio" può essere aggiunto quando serve. In particolare, una `JScrollPane` (anch'essa un `JComponent`) può rappresentare una text area con in più le barre di scorrimento.
- Modalità d'uso:

```
JTextArea area=new JTextArea(10, 25);
JScrollPane sp=new JScrollPane( area );
```

Uso nelle API di Java

Filtri nei flussi di I/O

- Un altro esempio notevole di utilizzo del pattern decorator si ha nei *filtri* sugli stream di I/O.
- Un *filtro* elabora il flusso di dati di uno stream sottostante, in modo da facilitarne l'utilizzo
- Esempio:
  ```
  InputStream is=new BufferedInputStream( new
  FileInputStream(nomefile));
  ```
- In questo caso, la gestione dell'input stream (es. un file) è bufferizzata: si introduce un buffer così che una read() attinga ad esso.
- Solo quando il buffer è vuoto il filtro (`BufferedReader`) provvede a riempirlo leggendo un blocco di dati dallo stream sottostante.
- Si ottiene una migliore utilizzazione del flusso di ingresso e una maggiore efficienza del programma.

Pattern correlati

- Adapter
- Composite
- Strategy