Università di Catania

Progetto per il Corso

Ingegneria del Software

Autore:
Giuseppe Pitruzzella

Professore:
Emiliano Tramontana
Andrea Fornaia

Il seguente progetto, affronta la problematica basata sull'implementazione di una classica struttura ad albero, questa necessaria alla rappresentazione di una struttura aziendale.
Insieme a questo, notiamo la volontà di poter trattare elementi semplici (ossia uno Sviluppatore o un Manager) ed elementi composti in egual modo; a questo si aggiunge l'esistenza di una composizione ricorsiva, per cui elementi composti possono formare elementi composti più complessi. Con elementi composti, intendiamo un insieme di elementi semplici.
Il pattern che risolve i problemi definiti precedentemente è Composite.
Composite propone una soluzione che si compone dei seguenti elementi.

- **Dipendente**, ossia l'interfaccia Component (quale può essere ed è una classe astratta) che rappresenta elementi semplici e composti.
  Al suo interno vengono dichiarate le **operazioni** che possono essere eseguite sugli elementi della composizione, come l'accesso e la gestione degli elementi semplici.
  Inoltre, sarà all'interno di Dipendente che definiremo i metodi per aggiungere o rimuovere un elemento semplice da un elemento composto, favorendo quindi la trasparenza.

  Di seguito, descriviamo i metodi definiti all'interno di Dipendente:

  - **aggiungiDipendente(Dipendente D)**
    Il metodo aggiungiDettagli() ha il compito di aggiungere un elemento semplice all'interno di una lista, ciò permetterà la loro gestione.
    Si noti che il seguente metodo, insieme al suo inverso, forniscono un implementazione base per chi non farà override dei metodi, ovvero le leaf, le quali non avranno bisogno di quest'ultimi metodi.
    Se chiamati a partire da una Leaf, sarà generata un exception.
  - **rimuoviDipendente(Dipendente D**
    Il metodo rimuoviDipendente() ha il compito di rimuovere un elemento semplice all'interno di una lista; ciò permetterà la loro gestione.
    Come abbiamo già scritto, anche rimuoviDipendente() fornisce un implementazione base per chi non farà override dei metodi, ovvero le leaf, le quali non avranno bisogno di quest'ultimi metodi.
    Dunque se chiamati a partire da un elemento semplice, sarà generata un exception.
  - **displayDettagli()**

Il metodo displayDettagli() all'interno di Dipendente, viene implementato dalle classi Sviluppatore, Manager e Dipartimento, queste sottoclassi di Dipendente.

  - **getStipendio**
    Infine il metodo getStipendio(), anch'esso implementato dalle classi Sviluppatore, Manager e Dipartimento, sottoclassi di Dipendente.

- **Sviluppatore**, ovvero la classe Leaf, rappresentante gli elementi semplici.
  Implementa l'interfaccia (Component) Dipendente, quindi il comportamento degli oggetti semplici.
  Per quanto riguarda i metodi definiti all'interno di Sviluppatore, oltre all'implementazione dei metodi di Dipendente, getStipendio() e displayDettagli(), troviamo al suo interno un metodo privato targetBonus(), un bonus rispetto lo stipendio.

- **Manager**, ossia una seconda classe Leaf, rappresentante gli elementi semplici.
  Implementa l'interfaccia (Component) Dipendente, quindi il comportamento degli oggetti semplici.
  Per quanto riguarda i metodi definiti all'interno di Sviluppatore, oltre all'implementazione dei metodi di Dipendente, , troviamo al suo interno un metodo privato targetBonus(), un bonus rispetto lo stipendio. I metodi implementati al suo interno sono getStipendio(), displayDettagli() ed infine targetBonus(), quest'ultimo maggiore rispetto a Sviluppatore.

- **Dipartimento**, ossia la classe Composite che rappresenta e definisce il comportamento per l'insieme di elementi semplici, tenendo un riferimento per ognuno di essi. Anch'essa è sottoclasse di Dipendente.
  Si noti che Dipartimento fa override delle operazioni per la gestione degli elementi semplici.
  Di seguito, descriviamo i metodi per cui viene fatto override, all'interno di Dipartimento:

  - **aggiungiDipendente(Dipendente D)**
    Il primo metodo implementato all'interno di Dipartimento è aggiungiDipendente(), quale ha il compito di inserire all'interno di una lista di elementi di tipo Dipendente, i vari elementi che compongono l'oggetto Composite.

- **rimuoviDipendente(Dipendente D)**
  Il secondo metodo implementato è rimuoviDipendente(), quale ha il compito di rimuovere un elemento semplice dalla lista di elementi di tipo Dipendente.
- **displayDettagli()**
  Il metodo displayDettagli() ha il compito di stampare a video le informazioni principali rispetto ogni Dipendente di un certo Dipartimento.
- **getStipendio()**
  Il metodo getStipendio() ha il compito di ritornare la somma degli stipendi rispetto ogni Dipendente di un certo Dipartimento.

- **App**, usa la classe astratta Dipendente per interagire con la struttura di elementi semplici e composti.