





Per velocità, le specializzazioni di ElementoSingolo non sono riprodotte in questa seconda parte di design, ma si dà per scontato che ogni classe definisca il proprio metodo di Accept(in visitor: ReportVisitor).

L'ipotesi principale che si fa per questa parte di progetto è che uno spettacolo abbia lo stesso titolo del film che proietta, e che ogni spettacolo proietti uno ed un solo film. Per questo motivo il titolo del film da inserire nel report sarà il titolo dello spettacolo che lo proietta.

Per soddisfare le richieste relative alla generazione di un report riguardante i biglietti venduti è stata aggiunta una classe Biglietto che, associata a OrarioSpettacolo e ad una nuova classe DataProiezione definisce i biglietti venduti in una determinata giornata per un determinato orario di inizio dello spettacolo. Questa associazione permette di definire univocamente il singolo spettacolo che viene proiettato.

La classe BigliettiVenduti è una classe contenitore che raggruppa per ogni data il numero di biglietti di ogni tipo (biglietto intero, ridotto, omaggio) distribuiti per lo spettacolo selezionato ed è collegata alla classe Spettacolo in modo che possa rimanere traccia dello storico della vendita dei biglietti.

In seguito all'introduzione delle suddette classi, sono stati introdotti nelle classi superiori i metodi relativi al raccoglimento dei dati così generati.

E' stato utilizzato il pattern Visitor per scorrere la gerarchia del cinema e per poter visitare ed utilizzare i metodi delle classi della gerarchia.

Il Visitor è strutturato mediante l'utilizzo di un'interfaccia che dichiara tutti i metodi di visita e la stampa del report, ed è implementato da un ConcreteVisitor che definisce i metodi di visita e si avvale dell'ausilio di un pattern Strategy per la gestione della stampa del report nei differenti formati.

Nella gerarchia del cinema invece sono stati aggiunti per ogni classe i metodi di Accept del visitor per permettere la visita e l'utilizzo dei metodi della classe.

Il pattern Visitor permette di scorrere velocemente la gerarchia del cinema e di definire la nuova operazione senza dover modificare le classi interessate. L'utilità di questo pattern risiede nella possibilità di effettuare altre operazioni senza stravolgere ulteriormente le classi della gerarchia.

Il pattern Strategy incapsula l'algoritmo di stampa del report in una gerarchia, e delega le richieste di stampa in dipendenza del formato desiderato. Le attuali implementazioni previste sono quelle relative alla generazione dei report nei formati xml, html e txt, ma l'utilità di questo pattern risiede nella facilità con cui è possibile creare una nuova classe che implementi i metodi astratti di ReportVisitor ottenendo in questo modo a tutti gli effetti un nuovo metodo di stampa.

Questa struttura complessa di Visitor&Strategy si rapporta con la gerarchia del cinema mediante un cliente che può essere il cliente vero e proprio, o un controller, tramite l'utilizzo di un applicativo grafico.

Nell'implementazione di questa parte di progetto si è cercato di rispettare l'*Open/Close Principle* e il *Liskov Substitution Principle*, anche grazie all'attuazione dei 2 pattern Visitor e Strategy, che richiedono comunque l'utilizzo di un'interfaccia o di una classe astratta e che permettono di sostituire "mentalmente" la classe astratta o l'interfaccia con la relativa classe concreta richiesta.

Un'eventuale soluzione simile a quella proposta prevede l'appiattimento delle classi ConcreteStrategy nel Visitor, e cioè, anziché un solo ConcreteVisitor, n differenti ConcreteVisitor per gli n differenti formati di stampa. Non è stata scelta questa alternativa, perché richiederebbe che tutte le classi implementino allo stesso modo l'algoritmo di raccoglimento dei dati del report.

Una altra soluzione alternativa più laboriosa alla generazione del report tramite l'utilizzo del Visitor potrebbe essere quella di inserire una entry che associa Biglietto a DataProiezione e un dictionary che associa ad ogni chiave DataProiezione un oggetto di tipo Biglietto.

Questa soluzione, è molto più laboriosa poiché richiede l'inserimento di almeno due nuove classi per la generazione del report e potrebbe richiedere un'eccessiva richiesta di risorse per tenere traccia di tutto lo storico nel database. D'altro canto però questa soluzione permetterebbe, in seguito all'utilizzo di query differenti, di effettuare differenti e molteplici statistiche.

