UNIVERSITÀ
DEGLI STUDI
FIRENZE

Esercitazione

Refactoring exercise

Obiettivo

- Il progetto CLion fornito contiene un progetto di un software per un gestore di noleggi di film

- Scopo dell'esercitazione è:

  - Applicare tecniche di refactoring viste a lezione

  - In particolare usando il polimorfismo per evitare il codice di if e switch per il calcolo del costo di noleggio e di assegnazione dei punti di fidelizzazione

  - Lo scopo è fare in modo che usando il puntatore alla classe base `Price` si possa calcolare costo e punti da dare sulla base della classe concreta che estende `Price`.

  - Nel futuro basterà aggiungere nuove varianti di costo e il sistema continuerà a funzionare senza bisogno di altri cambiamenti.

Schema del codice

- Il programma è composto da 3 file:
- `Main.cpp` che mostra l'uso del sistema. Lo si tratti come una specie di unit test (non fornito)
- `Customer` implementa un utente ed ha il metodo statement() che fornisce un riassunto delle attività, costi e punti di fidelizzaizone ottenuti. Il metodo può essere usato come base per creare un metodo aggiuntivo con output HTML…
- `Rental` implementa un noleggio di film.
- `Movie` rappresenta un film.
- `Price` e sue classi derivate rappresentano uno schema di costi di noleggi. Movie è composta con `Price`.

Attività 1

- Fare in modo che sia responsabilità di `Movie` calcolare il costo e l'assegnazione dei punti di fidelizzazione
  - Questo è motivato dal fatto che il codice dipende dai tipi dei film. Si dovrà passare il numero di giorni di noleggio dalla classe `Rental` al metodo.
- Fare altrettanto per il calcolo dei punti di fidelizzazione.
- Usare il refactoring **Move method**. Dato che l'interfaccia di `Rental` ha questi metodi come pubblici si deve fare una delega all'oggetto movie presente in `Rental`.
  - Attenzione: Il refactoring di CLion rimuove il metodo dalla classe di partenza, questo andrà quindi ri-aggiunto a mano.

Attività 2

- Per avere il polimorfismo i metodi di calcolo costo e punti fedeltà vanno ulteriormente spostati in Price (**Move method**), rendendoli virtuali:
  - Dato che nel passo precedente i metodi erano pubblici in `Movie` si deve continuare a delegare a `Price` la loro implementazione
  - E' finalmente possibile eliminare lo switch in `getCharge()`: si faccia l'override del metodo nelle classi derivate di `Price` e nell'implementazione di ogni metodo si copi (con opportuno riadattamento) il frammento di codice del ramo di switch
    - Si renda puramente virtuale il metodo `getCharge` in `Price`
- Il metodo `getFrequentRenterPoints()` può invece essere implementato in `Price` con una semplice `return 1`, mentre il codice che tiene conto anche dei giorni di noleggio viene messo nel metodo in override in `NewReleasePrice` (con opportuno aggiustamento)
  - Le altre 2 classi di costo non hanno necessità di fare override e si limitano a ereditare l'implementazione base

UNIVERSITÀ DEGLI STUDI FIRENZE

Attività 3

- Rinominare il membro `CHILDRENS` nell'enumerazione `Constants` in `Movie.h`:

- Usare il refactoring **Rename** per rinominare il membro come `CHILDREN`