Esercizi di Ingegneria del software

2023/2024

UNIVERSITÀ
DI PARMA

Nome: **Di Agostino Manuel**, **Simone Colli**
Insegnamento: Ingegneria del software
Anno: 2023/2024

1 Design patterns

1.1 Composite pattern

Esercizio 1.1.1

Si vuole modellare la mappa di un videogioco *open-world* utilizzando una struttura dati ad albero; a seconda del livello di zoom dell'interfaccia è infatti possibile osservare una versione più dettagliata della mappa, corrispondente ad un particolare nodo dell'albero. Esistono tre categorie di nodi:

- **foglia**: non ulteriormente espandibile, corrisponde ad un preciso punto fisso sulla mappa, importante per la logica del gioco;
- **distretto**: identifica zone più ampie, ad esempio un quartiere cittadino formato dalla composizione di nodi foglie;
- **regione**: aggrega più distretti contigui;
- **stato**: il livello più alto nella gerarchia considerata.

Ad ogni nodo sono inoltre sempre associati un *nome* e una *tipologia* conforme alla seguente interfaccia:

```
public enum NodeType {
    LEAF,
    DISTRICT,
    REGION,
    COUNTRY
}
```

Realizzare un class diagram UML che descriva la struttura della soluzione e fornire un'implementazione in Java che ne soddisfi i requisiti.

1.2 Decorator pattern

Esercizio 1.2.1 (*Ispirato ad un esempio del libro GoF*)

I flussi sono un'astrazione fondamentale nella maggior parte delle strutture di I/O. Un flusso può fornire un'interfaccia per convertire oggetti in una sequenza di byte o caratteri. Questo ci consente di trascrivere un oggetto su un file o su una stringa in memoria per il recupero successivo. Un modo diretto per farlo è definire una classe astratta `Stream` con le sottoclassi `MemoryStream` e `FileStream`. Il corpo della classe `Stream` è il seguente:

```
public abstract class Stream {
    // private data section

    public void PutInt() {
        // impl
    }
    public void PutString() {
        // impl
    }
    public abstract void HandleBufferFull();
}
```

Le classi `MemoryStream` e `FileStream` si occupano di ridefinire il metodo astratto `HandleBufferFull()` per scrivere direttamente sulla memoria RAM e rispettivamente su file.

Supponiamo che la classe astratta `Stream` manipoli il buffer di caratteri del flusso tramite codifica `UTF-8`; si aggiunga la funzionalità di conversione di codifica del testo in `ASCII` standard a `7 bit` senza intaccare le classi/interfacce sopra citate, utilizzando dunque il *Decorator pattern*.

Fornire anche un class diagram UML della soluzione.

1.3 Command pattern

Esercizio 1.3.1

Si vuole realizzare la logica applicativa di un editor di testo che permetta di manipolare file testuali in codifica `ASCII` standard a `7 bit`. È previsto che l'interfaccia `EditableFile` esponga una serie di funzionalità:

- **creazione** del file, a partire da un nome scelto dall'utente
- **eliminazione** del file
- **lettura** completa del file
- **lettura** parziale del file, specificando punto di inizio e fine (numeri riga)
- **concatenamento** di testo alla fine del file
- **modifica** parziale del file, specificando punto di inizio e fine (numeri riga)
- **ridenominazione** del file
- **salvataggio** del file

Bisogna inoltre prevedere la possibilità di annullare fino a *256 modifiche* effettuate dall'utente.

Scrivere un'implementazione in Java del sistema descritto fornendone una descrizione tramite class diagram UML.

Esercizio 1.3.2

Si consideri la seguente interfaccia Java che descrive una struttura dati *queue*:

```
public interface Queue<T> {
    public void enqueue(T elem);
    public T dequeue();
    public void clear();
    public boolean isEmpty();
}
```

L'interfaccia in questione deve essere implementata dalla classe **SimpleQueue**, che offre un costruttore senza paramentri per la creazione di una coda vuota. La coda vuole permettere l'operazione di *undo* per l'ultima modifica effettuata. Descrivere la classe descritta mediante un class diagram UML e fornirne un implementazione in Java.

1.4 Visitor pattern

Esercizio 1.4.1

[Richiede l'esercizio 1.1.1]

Sulla struttura dati implementata in 1.1.1 sono richieste le seguenti azioni:

- possibilità di creare elenchi dei nodi suddivisi per categoria (`LEAF`, `DISTRICT`, `REGION`, `COUNTRY`)
- visita in ampiezza per livelli, a partire da un determinato nodo e con una profondità massima, utile al motore di rendering della mappa

1.5 Abstract factory pattern

Esercizio 1.5.1

[Richiede l'esercizio 1.1.1]

Sulla struttura dati implementata in 1.1.1 è richiesto di delegare la creazione dei vari `Component` ad un'abstract factory.

1.6 Proxy pattern

Esercizio 1.6.1

[Richiede l'esercizio 1.3.1]

A partire dalla logica applicativa dell'editor di testo in 1.3.1 è richiesto di delegare la creazione dei vari `Component` ad un'abstract factory.