Sviluppo Applicazioni Software 13/14 - Design Pattern Spike

A Team: Bono, Cerrato, Vinci

3 giugno 2014

Testo della consegna

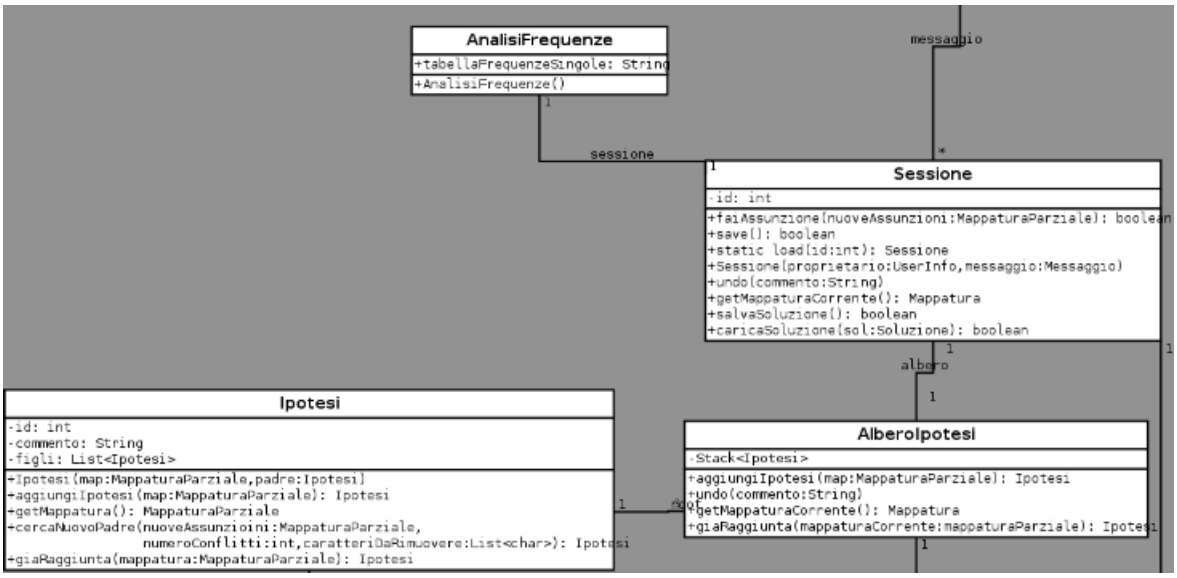
Si tratta di un esercizio di implementazione di un Design Pattern GOF a vostra scelta fra: Decorator, Composite, Observer, Visitor, State, Proxy

Dovete immaginare una situazione pratica di utilizzo dei pattern (volendo presa dal nostro progetto o da qualche sua evoluzione che immaginate possa avere nel futuro, ma anche completamente diversa se non vi viene in mente nulla). Quindi dovete disegnare lo schema delle classi nel pattern istanziato al caso che avete pensato, e realizzare uno scheletro di implementazione che faccia vedere l'utilizzo del pattern stesso. Per scheletro si intende la struttura delle classi con campi e metodi rilevanti, e un abbozzo di linee di codice all'interno dei metodi che sintetizzano l'uso degli oggetti coinvolti nel pattern. Naturalmente non è necessario che il risultato sia compilabile e/o eseguibile.
Se per caso però uno di questi pattern è entrato di diritto nel codice del vostro progetto, potete consegnarci direttamente le classi relative così come le avete implementate, corredate dalla porzione di modello del progetto che descrive la struttura del pattern.

Il pattern da noi selezionato è il pattern **Proxy**.

Descrizione dell'utilizzo del Pattern

Abbiamo definito all'interno del nostro Modello di Progetto una classe *Sessione*. Tale classe racchiude all'interno di sè le varie operazioni che la Spia vorrà svolgere durante la decifratura di un messaggio, e mantiene riferimenti agli oggetti utili a questo scopo.



Il nostro oggetto Sessione contiene, tra le altre cose, il riferimento all'oggetto *AlberoIpotesi*: tale oggetto è di tipo ricorsivo, in cui varie Ipotesi sono aggiunte via via che la Spia compone le sue assunzioni tentando di decifrare il Messaggio. Si tratta di un oggetto complicato e pesante, che non desideriamo caricare a meno che non sia strettamente necessario.
È d'altra parte necessario che l'utente possa visualizzare le Sessioni che sta effettuando, potendo scegliere quale continuare o eliminare. In questa operazione, che nei nostri contratti abbiamo chiamato *mostraSessioni*(), bisogna necessariamente caricare dal DB l'intero oggetto *Sessione*, completo di *AlberoIpotesi*. Desideriamo evitare questo scenario, scarsamente efficiente, utilizzando il pattern **Proxy**, in particolare secondo il concetto di **Virtual Proxy**.

Schema del Pattern nel contesto

I metodi che sono interessanti per le operazioni che vogliamo descrivere sono *load*(*id*) e *toString*(), di cui riportiamo le implementazioni di seguito.

```
public interface Sessione {
        public static Sessione load(int id);
        public String toString();
}
```

```
public class SessioneImpl implements Sessione {
  public Sessione(CachedRowSet crs) {
    this.id = crs.getInt("id");
    this.messaggio = Messaggio.load(crs.getInt("messaggio"));
    this.proprietario = UserInfo.load(crs.getInt("proprietario"));
    Blob bl = crs.getBlob("albero");
    byte[] buf = bl.getBytes(1, (int) bl.length())
    ObjectInputStream objectIn;
    objectIn = new ObjectInputStream(new ByteArrayInputStream(buf));
    this.albero = ((AlberoIpotesi) objectIn.readObject());
  }
  public static SessioneImpl load(int id) {
    DBController dbc = DBController.getInstance();
      CachedRowSet crs = dbc.execute("SELECT * FROM Sessione WHERE id = ?", id);
      crs.next();
      return new SessioneImpl(crs);
  }

  public String toString() {
    return new RuntimeException("errore SessioneImpl");
    //non vogliamo che toString() sia eseguito da questa classe
  }


}
```