Progetto GIÀ
-
Ingegneria del software

Tura Marco - s0000658671 - marco.tura4@studio.unibo.it

Link Utili

 Repository GitHub

 Regolamento G.I.A.

 Relazione Progetto

I link che compaiono qui a fianco sono le risorse utili per approfondire il contenuto del progetto:

- Il repository che contiene il codice del prototipo.
- Il regolamento a cui si ispira tutto il progetto.
- La relazione descrittiva del progetto.

Specifiche del progetto - 1

1. Gestire l’inserimento ed il caricamento dei dati delle società e degli atleti.
2. Gestire l’inserimento dei dati delle prestazioni di gara.
3. Garantire la persistenza per i dati

Specifiche del progetto - Caso d’Uso

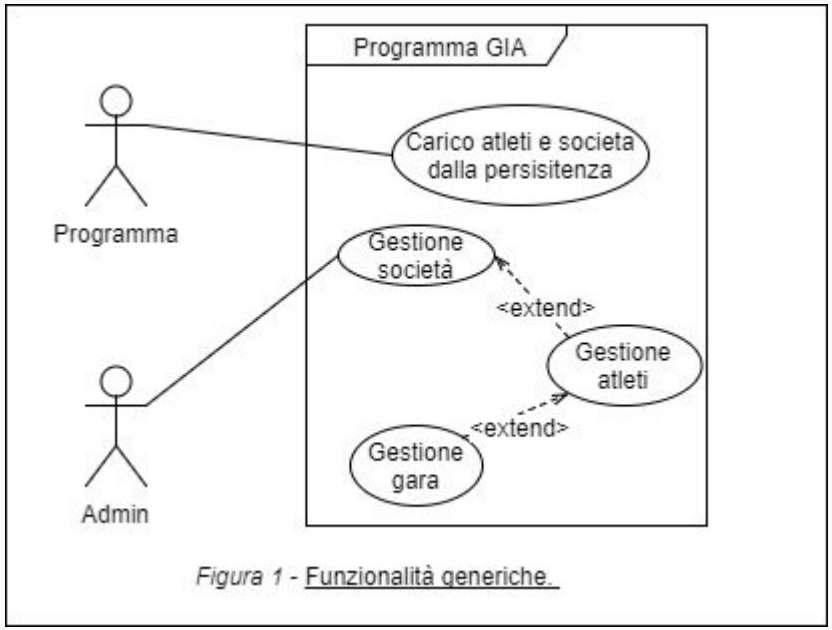


Figura 1 - Funzionalità generiche.

1. Il programma carica società ed atleti dal dispositivo di persistenza.
2. L’admin inserisce, rimuova o edita le società
3. L’admin inserisce, rimuove o edita gli atleti e li associa alle società di appartenenza (non possono esserci atleti senza società).
4. L’admin, iscrive o disiscrive gli atleti alla gara.
5. L’admin gestisce la gara.

Specifiche Progetto - Gestione Società

Il programma deve **caricare** i dati di ogni società all'avvio, i dati in questione possono essere:

- **Modificati**
- **Eliminati**

Nel caso non vi siano dati il programma si attiverà senza alcuna società al suo interno.

Darà in ogni caso la possibilità di **inserire nuove società**.

Gestione Società - Casi d’Uso

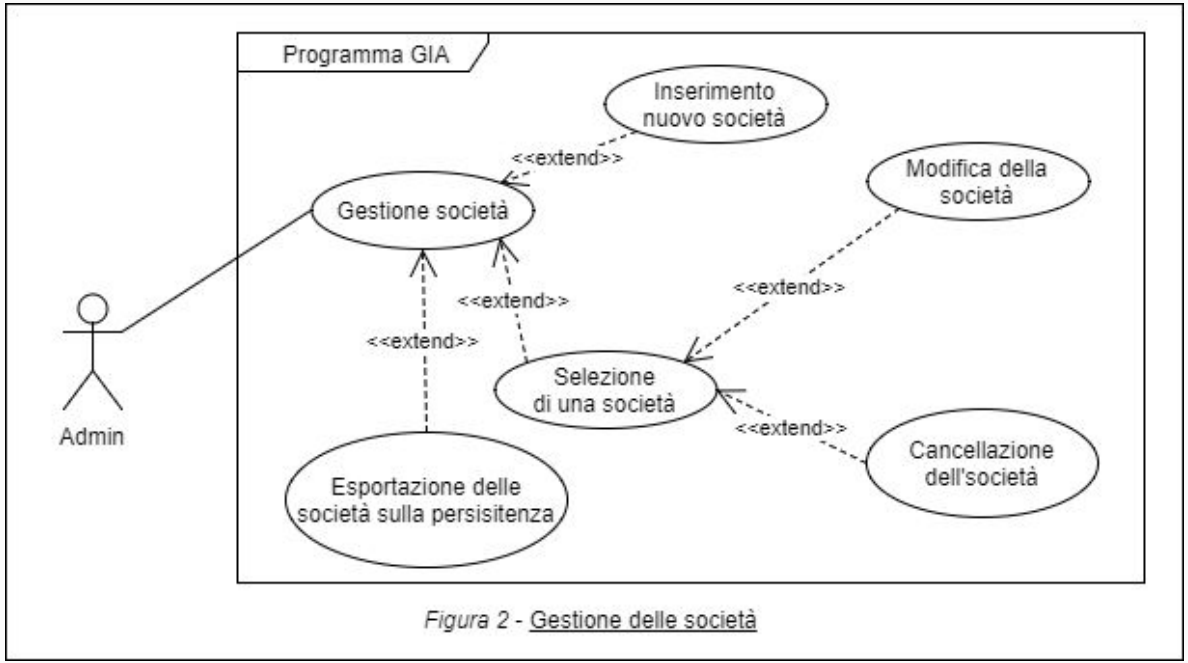


Figura 2 - Gestione delle società

Specifiche Progetto - Gestione Atleti

Il programma deve **caricare** i dati di ogni atleta all'avvio, i dati in questione possono essere:

- **Modificati**
- **Eliminati**

Nel caso non vi siano dati il programma si attiverà senza alcuna atleta al suo interno.

Darà in ogni caso la possibilità di **inserire nuovi atleti**.

Gestione Atleti - Casi d’Uso

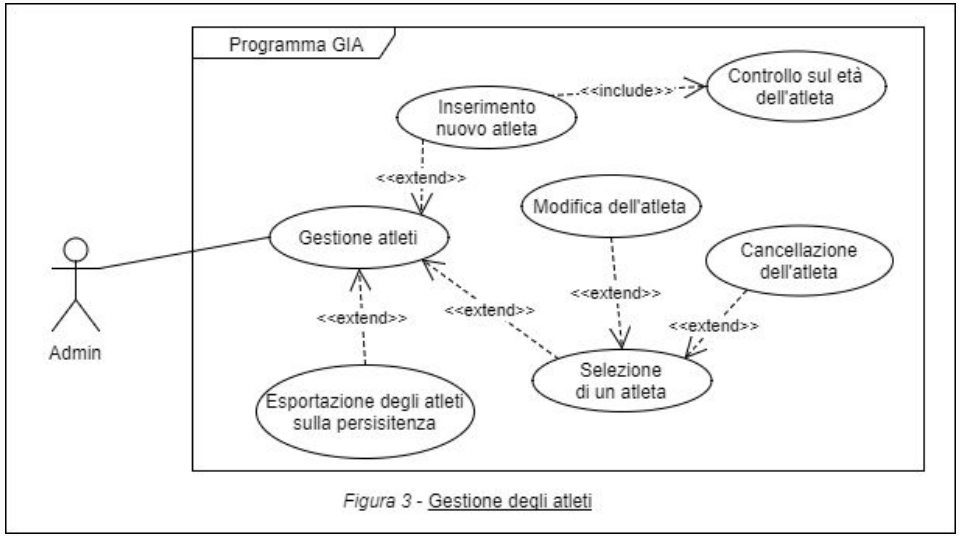


Figura 3 - Gestione degli atleti

1. L’admin seleziona una società da associare al nuovo atleta.
2. L’admin inserisce i dati associati al nuovo atleta.
3. il sistema controlla che l’età dell’atleta inserita sia inferiore a 18 anni e inferiore a 14 anni (in caso contrario lancia una notifica).
4. L’admin completa l’inserimento di un nuovo atleta.

Specifiche Progetto - Gestione Gara

Il programma permette di **creare una gara** inserendo le specialità presenti.

In seguito alla creazione di una gara permette di **iscrivere gli atleti**, separandoli per sesso.

Infine permette di **associare** ad ogni atleta, per ogni specialità di gara, **una prestazione**, il programma **calcola in modo automatico il punteggio** senza dover aggiungere altri dati oltre quelli della prestazione.

Gestione Società - Casi d’Uso 1

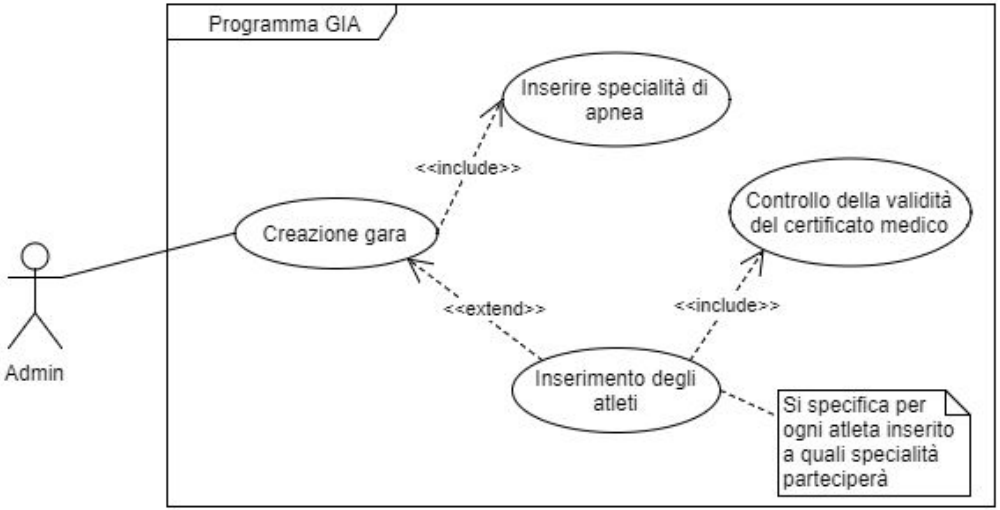


Figura 5 - Creazione gara

Gestione Società - Casi d’Uso 2

1. L’admin inserisce le specialità che saranno presenti in gara.
2. L’admin inserisce i partecipanti alla gara (passo ripetuto per ogni atleta che si vuole iscrivere).
3. L’admin seleziona un atleta da quelli presenti nel sistema
4. L’admin specifica per ogni atleta a quali specialità della gara parteciperà.
5. Il programma controlla la validità del certificato
6. L’admin conclude la creazione di una nuova gara.

Gestione Società - Casi d’Uso 3

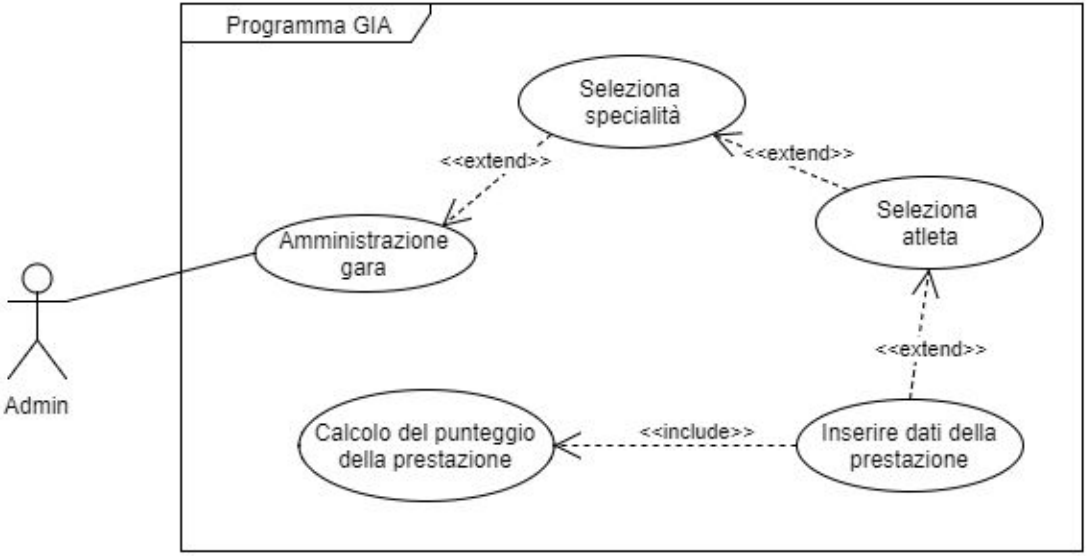
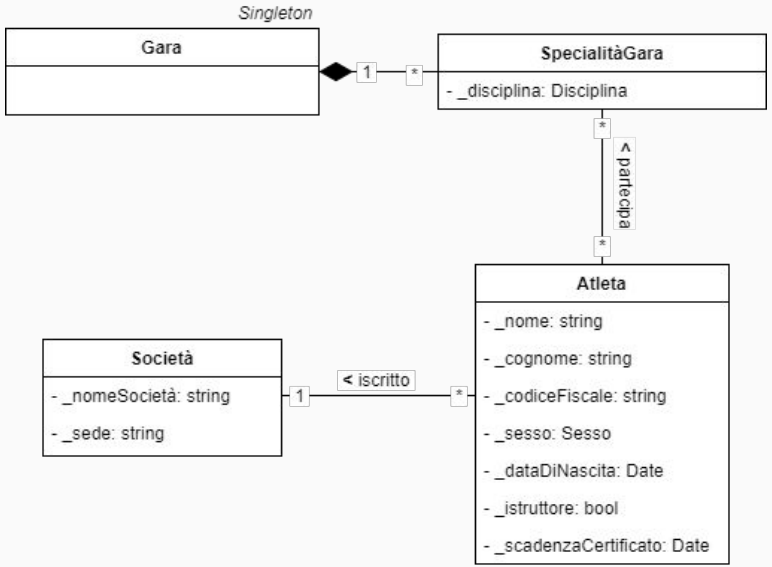


Figura 6 - Amministrazione gara

Uml di Analisi 1



Dalle specifiche sono stati ricavati i seguenti componenti:

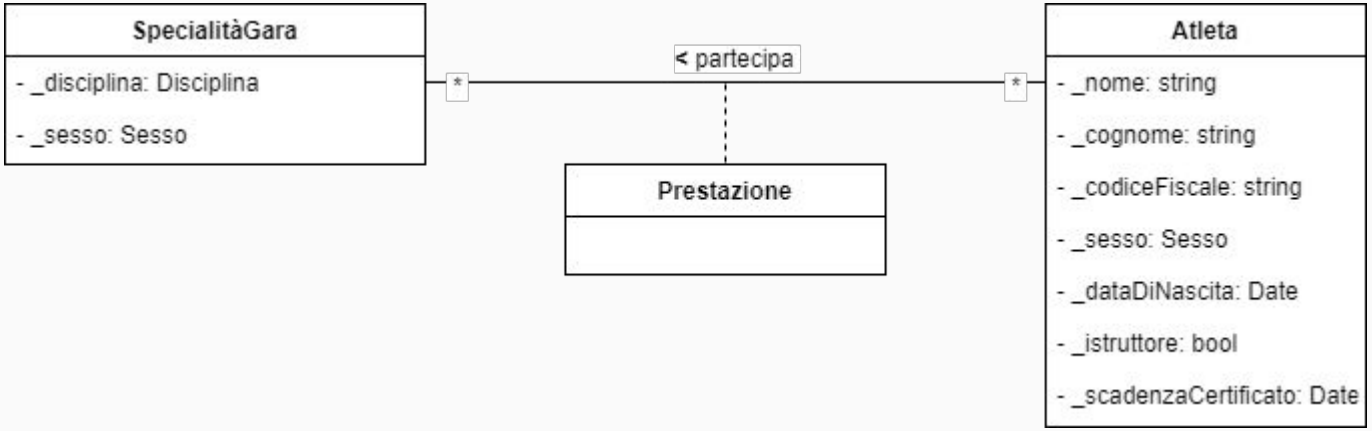
Gara - Un singleton che immagazzina tutti i dati inseriti e ne gestisce la persistenza attraverso classi con cui collabora.

Societa - Classe che astrae una società

Atleta - Classe che astrae un atleta

Specialità gara - Classe che astrae una disciplina all'interno di una gara

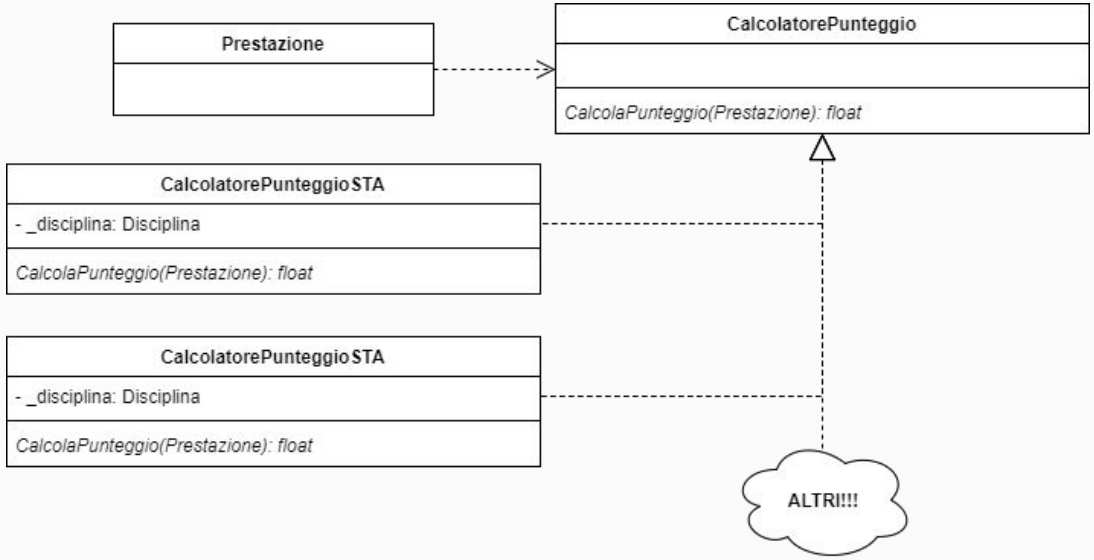
Uml di Analisi 2



Essendo necessario associare ad ogni **atleta** una **prestazione**, per ogni **disciplina** a cui partecipa nella gara, si è dovuta definire una **classe di associazione**.

La classe **Prestazione** contiene tutti i dati necessari al calcolo del punteggio della prova.

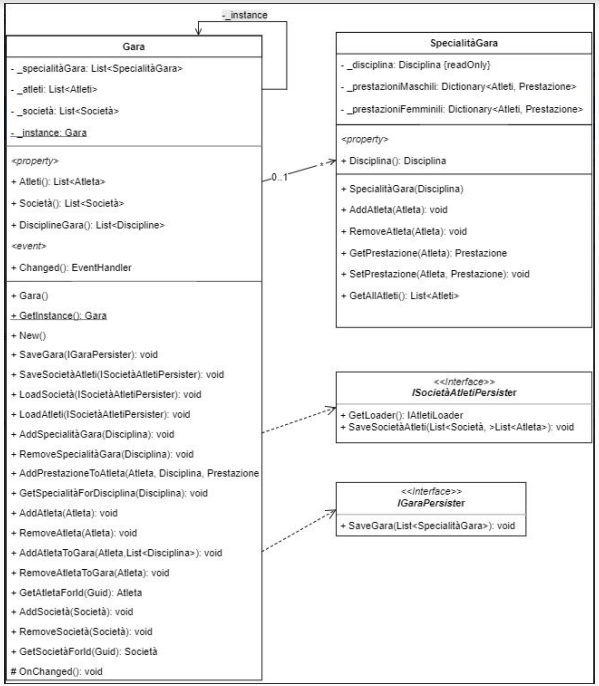
Uml di Analisi 2



Il calcolo del punteggio viene eseguito dalla classe prestazione dopo aver istanziato un **calcolatore punteggio** adeguato alla **disciplina** della **prestazione**.

Gli algoritmi di calcolo dipendono dal tipo di disciplina che si sta gestendo in quel momento, sia dal punto di vista dell'algoritmo che da quello di parametri specifici.

Uml di Progettazione 1

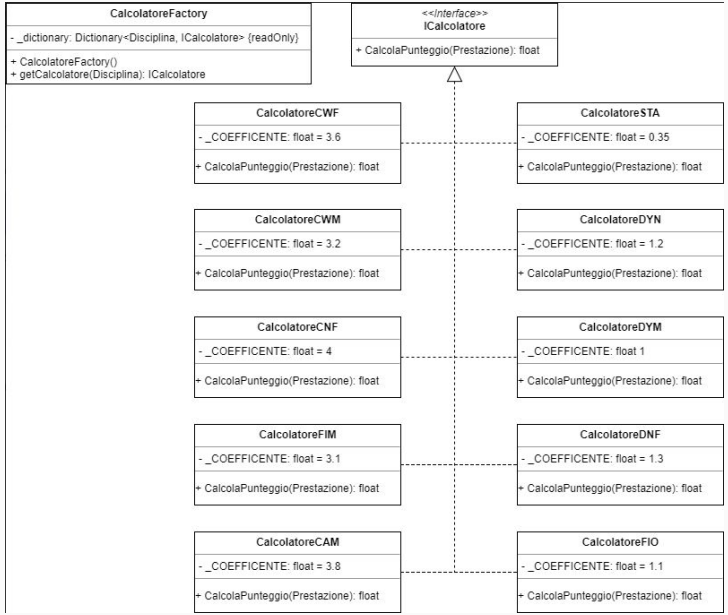


La classe **Gara** in fase di progettazione è stata modificata in modo da contenere liste di **Atleti** e **Società,** ed una lista di **SpecialitàGara** che la compongono.

Per la persistenza si avvale di istanze di **ISocietàAtletiPersister** e **IGaraPersister** in questo modo è in grado di garantire la persistenza ai dati che gestisce.

La classe **SpecialitàGara** contiene due dizionari separati, uno per gli atleti di sesso maschile ed un altro per quelli di sesso femminile, si è deciso di trasferire la classe di associazione **Prestazione** all'interno dei dizionari, così da garantire un corretto funzionamento.

Uml di Progettazione 2



Per avere un corretto funzionamento della componente di calcolo dei punteggi, si è deciso di implementare il tutto seguendo due design pattern contemporaneamente: **FLyWeight** e **Strategy**

Il **pattern strategy** ,ci ha permesso di istanziare correttamente la classe concreta di ICalcolatore, che contiene l'algoritmo specifico della disciplina.

Il **pattern FlyWeight** ci ha permesso di istanziare, attraverso una **factory**, le classi che condividono uno stato intrinseco specifico (_COEFFICIENTE) variabile da disciplina a disciplina.

Design Pattern 1 - Strategy

Il **pattern strategy** permette di definire un insieme di algoritmi tra loro correlati, incapsulandoli in una gerarchia di classi, così da renderli intercambiabili, in questo modo un cliente potrà definendo un istanza astratta, e successivamente, a necessità, istanziare una classe concreta applicando a questa il meccanismo di delega di eventuali operazioni che richiedono specifici algoritmi

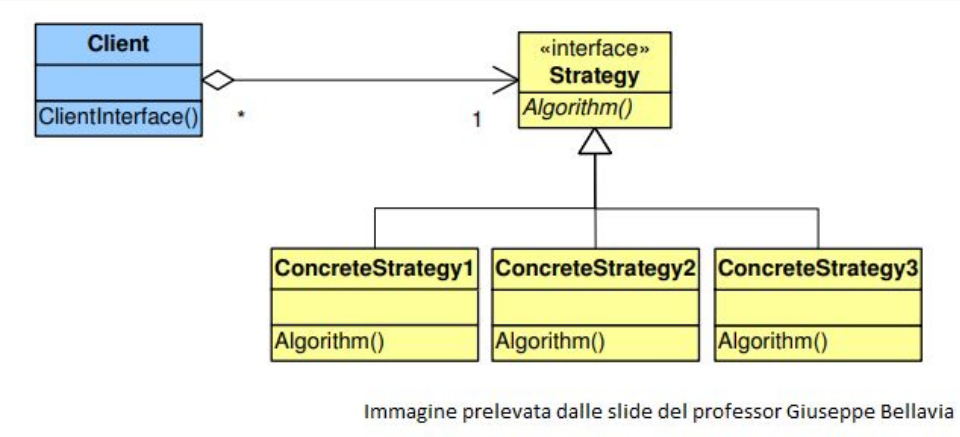


Immagine prelevata dalle slide del professor Giuseppe Bellavia

Design Pattern 2 - Flyweight

Il pattern flyweight permette di condividere oggetti “leggeri” in modo tale da minimizzare il costo di un loro impiego.

I flyweight essendo oggetti condivisi, usabili simultaneamente da più clienti necessitano di essere istanziati solo attraverso una factory.

I flyweight, proprio per via dell'essere oggetti leggeri, per essere utili necessitano di due tipi di stato:

- **Stato Intrinseco -** Non dipendente dal contesto di utilizzo quindi condiviso da tutti i clienti, per cui memorizzato nel flyweight.
- **Stato Estrinseco -** Dipendente dal contesto di utilizzo è quindi non condiviso tra i clienti, ma memorizzato o calcolato dallo specifico cliente, e passato al flyweight quando viene invocata una sua operazione

Design Pattern 3 - Flyweight

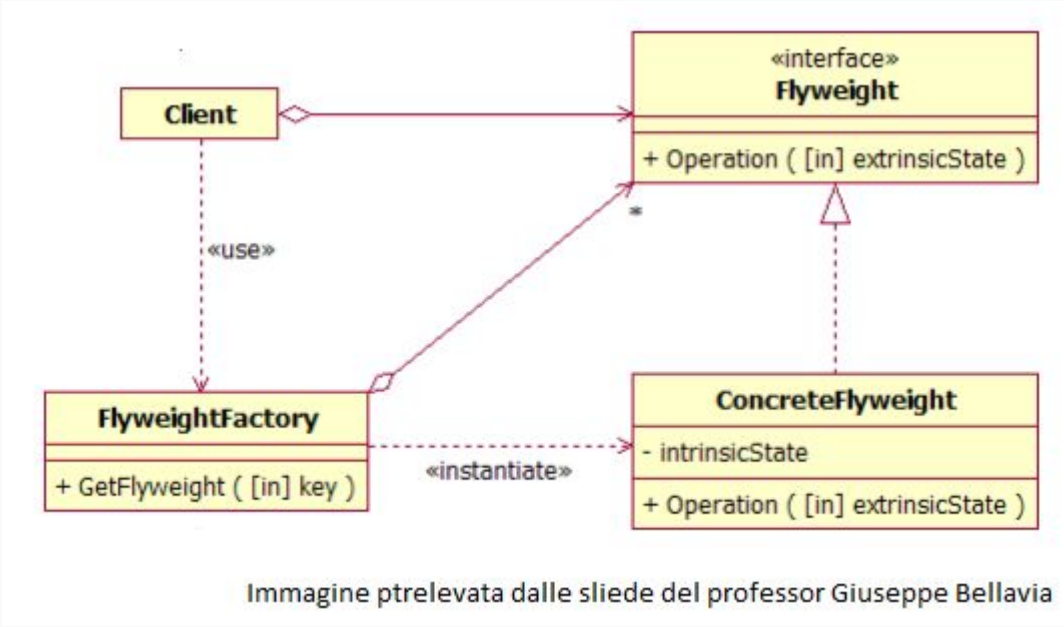


Immagine ptrelevata dalle sliede del professor Giuseppe Bellavia

Design Pattern 4 - Conclusioni

In conclusione si è voluto adottare questo specifico approccio per quanto riguarda i **design pattern** per venire incontro ad una necessità riscontrata negli anni.

Ogni anno il regolamento del G.I.A. viene aggiornato, modificando il numero di discipline o variando il coefficiente usato nel calcolo dei punteggi, l'approccio che si è seguito quindi è quello di semplificare al massimo l'aggiornamento della applicazione in questi specifici contesti di refactoring, nel caso quindi che si decida di eliminare o aggiungere una disciplina basterà modificare la classe enumerativa **Discipline** ed aggiungere o rimuovere un **CalcolatoreXXX**, apportando modifiche minime alla UI.

Nel caso invece che si desideri solo apportare una modifica ai coefficienti, basterà modificare la classe **CalcolatoreXXX**.