COESIONE E ACCOPPIAMENTO DELLE CLASSI E PRINCIPI DI BUONA PROGETTAZIONE ADOTTATI

Coesione

Misura quanto le parti che sono incluse nello stesso modulo sono legate tra di loro (è una caratteristica intra-modulo). Per ogni classe definiremo il tipo di coesione individuata.

- Classe Contact: coesione funzionale, la classe gestisce i metodi e gli attributi necessari a gestire un singolo contatto.
- Classe astratta ContactList: coesione funzionale, la classe gestisce i metodi e gli attributi necessari a gestire un insieme di contatti.
- Classe Email: coesione funzionale, la classe gestisce i metodi e gli attributi necessari a gestire un insieme di e-mail.
- Classe PhoneNumber: coesione funzionale, la classe gestisce i metodi e gli attributi necessari a gestire un insieme di numeri di telefono.
- Classe EmailChecker: coesione funzionale, la classe implementa l'interfaccia Checker per gestire il metodo isValid per validare un e-mail.
- Classe PhoneNumberChecker: coesione funzionale, la classe implementa l'interfaccia Checker per gestire il metodo isValid per validare un numero di telefono.
- Classe Rubrica: coesione funzionale, la classe estende la classe astratta ContactList per gestire un insieme di contatti e salvarli/caricarli su/da file.
- Classe Group: coesione funzionale, la classe estende la classe astratta ContactList per gestire un insieme di contatti con nome e descrizione.
- Classe Groups: coesione funzionale, la classe gestisce i metodi e gli attributi necessari a gestire un insieme di gruppi.

Accoppiamento

Misura il grado di interdipendenza tra moduli diversi (è una caratteristica inter-modulo). Per ogni relazione tra le classi espliciteremo il tipo di accoppiamento individuato.

- Classe Contact→ Classe PhoneNumber: la classe Contact mantiene un riferimento di tipo PhoneNumber. quindi utilizza un'associazione 1:1. Il tipo di accoppiamento è per dati poichè la classe PhoneNumber passa solo le informazioni necessarie al funzionamento della classe Contact.
- Classe Contact→ Classe Email: la classe Contact mantiene un riferimento di tipo Email. quindi utilizza un'associazione 1:1. Il tipo di accoppiamento è per dati poichè la classe Email passa solo le informazioni necessarie al funzionamento della classe Contact.
- Classe ContactList→ Classe Contact: la Classe astratta ContactList gestisce una lista di studenti, tuttavia i contatti possono essere creati ed esistere al di fuori della ContactList, quindi possiamo dire che il legame che li unisce è un'associazione. Il tipo di accoppiamento è per dati poichè la classe Contact passa sole le informazioni necessarie al funzionamento della classe ContactList.

- Classe Email → Classe EmailChecker: la Classe EmailChecker verifica la validità di un indirizzo email passato dalla Classe Email. Il tipo di accoppiamento è per dati, poiché EmailChecker riceve solo la stringa dell'indirizzo da verificare.
- Classe PhoneNumber → Classe PhoneNumberChecker: la Classe PhoneNumberChecker verifica la validità di un numero di telefono passato dalla Classe PhoneNumber. Come per il caso precedente, anche qui il tipo di accoppiamento è per dati, dato che PhoneNumberChecker preleva solamente il numero di telefono come informazione necessaria.
- Classe Rubrica → Classe ContactList: Rubrica eredita da ContactList solo le informazioni necessarie per la gestione dei contatti ribadendo un tipo di accoppiamento per dati.
- Classe Groups → Classe Group: La Classe Groups rappresenta un insieme di oggetti Group, con un tipo di accoppiamento per dati.
- Classe Group → Classe ContactList: La Classe Group estende la Classe ContactList al fine di gestire un insieme di contatti con metodi e attributi ereditati definendo quindi un tipo di accoppiamento per dati.

PRINCIPI DI BUONA PROGETTAZIONE ADOTTATI

- SINGLE RESPONSIBILITY PRINCIPLE: Abbiamo cercato di rendere il più modulare possibile il sistema, definendo classi semplici che svolgono un compito ben preciso. Ad esempio abbiamo creato le classi Email e PhoneNumber per gestire le email e i numeri di telefono di ogni contatto, quindi verificarne la correttezza mediante il metodo isValid delle classi EmailChecker e PhoneNumberChecker. La modifica di queste due classi rimarrà "interna ad esse" (ad esempio se vogliamo aggiungere più di tre numeri di telefono o più di tre email si dovrebbero modificare solo le classi Email e PhoneNumber). Lo stesso vale per le altre classi identificate.
- OPEN/CLOSED PRINCIPLE: Abbiamo cercato di rendere le classi chiuse alla modifica ma aperte all'estensione, dichiarando gli attributi privati e rendendoli accessibili tramite i getter e i setter o anche sfruttando l'ereditarietà. Un chiaro esempio si può individuare nelle classi Rubric e ContactList, dove la classe Rubric estende la classe ContactList aggiungendo due metodi.
- PRIVILEGIARE L'ASSOCIAZIONE RISPETTO ALL'EREDITARIETÀ: Abbiamo preferito, ove possibile, l'associazione rispetto all'ereditarietà per ridurre la forza di accoppiamento tra di esse. Ad esempio vi è associazione tra la classe Contact e le classi Email e PhoneNumber, o ancora aggregazione tra la classe Groups e la classe Group.
- INTERFACE SEGREGATION PRINCIPLE: abbiamo cercato di utilizzare interfacce piccole e specifiche per consentire alle classi di dipendere da un insieme minimo di metodi. Un esempio è l'interfaccia funzionale Checker.
- PRINCIPIO DI ROBUSTEZZA: abbiamo introdotto molte eccezioni per garantire che l'utente non possa portare il sistema in uno stato inconsistente e che non vi siano perdite di dati.