

## Diagramma delle classi

Ogni classe è impegnata alla finalizzazione di un'unica funzione: si vedano ad esempio

- le classi Libro, Utente e Prestito (gestiscono lo stato dei dati);
- le varie classi che implementano l'interfaccia Gestore (gestiscono le operazioni su collezioni);
- le varie classi controller (gestiscono l'interazione con l'interfaccia utente).

Grazie a tale progettazione, è possibile dunque raggiungere un livello di coesione elevato.

In generale, sono state evitate dipendenze che non fossero essenziali.

## Principi SOLID:

### S- Single Responsibility

Nelle entità (Libro, Utente, Prestito), nei gestori e nelle classi controller, ciascuno svolge funzionalità distinte (o parti che concorrono a realizzare una singola funzionalità). La validazione è separata in classi dedicate (ValidaDatiLibro, ValidaDataUtente, ValidaDataPrestito) e la persistenza è isolata nella classe Archivio.

### O - Open/Closed

Abbiamo definito interfacce generiche come Gestore<T> e ValidaData<T>, permettendo elevata modularità e testabilità nel codice.

### L - Liskov Substitution

Tutte le implementazioni di Gestore<T> (GestioneLibri, GestioneUtenti, GestionePrestiti) rispettano il contratto delle interfacce.

### I - Interface Segregation

ValidaData<T> presenta solo il metodo *isValido()*. Gestore<T>, invece, ha tre metodi, di cui uno potrebbe essere troppo generico (*cerca()*) e non utilizzabile da una delle classi che lo implementano (GestionePrestiti), violando di fatto il principio di segregazione delle interfacce. Per migliorarlo si potrebbe creare una nuova interfaccia.

### D - Dependency Inversion

Il principio è stato introdotto tramite l'uso delle interfacce Gestore<T> e ValidaData<T>, in modo da rendere possibile il disaccoppiamento in futuro delle classi di alto livello (Controller) dalle implementazioni concrete, sebbene in questa fase i controller istanzino ancora direttamente le classi gestori.













