# Progetti programmazione avanzata

2022-2023

JAVA

Il programma si presenta come un piccolo gestionale per abbonamenti in palestra.

Oltre alla gestione degli abbonamenti, il programma è in grado anche di gestire dei corsi ai cui solo gli utenti con un abbonamento plus possono iscriversi.

Grazie al software è possibile creare due tipi di abbonato, “normale” e plus aventi un costo differente. Gli abbonati plus hanno la possibilità di iscriversi a corsi come yoga e cardio (per ora sono stati implementati solo questi due).

I corsi hanno un proprio costo, il quale verrà aggiunto al costo dell’abbonato plus nel caso si iscriva.

Il corso di yoga può esistere una sola volta a settimana, in questo caso il sabato.  
Il corso cardio può essere presente più volte a settimana, ma solo una volta nello stesso giorno.

Tramite il gestionale, oltre la creazione degli abbonati e dei corsi, è possibile avere un elenco di tutti gli abbonati e dei corsi inseriti in una lista generica ordinabile.   
Inoltre, è possibile applicare uno sconto agli abbonati, con la condizione che il loro “conto” non scenda sotto i 400€. Esiste uno sconto base del 10% che può essere applicato una sola volta per abbonato, oppure è possibile inserire uno sconto a scelta.

Queste funzioni sono state implementate utilizzando le diverse caratteristiche del linguaggio Java.

Nello specifico, le classi implementate sono le seguenti:

* **Abbonato**: è possibile creare e abbonati i quali sono identificati da un id univoco (final), dati personali e un conto di base di 500€. Implementa i metodi toString, compareTo e applicaSconto (per il binding dinamico) il quale se invocato può applicare lo sconto “base” del 10% altrimenti è possibile passare uno sconto con la condizione che questo non faccia scendere il conto al di sotto di 400€.
* **Abbonato\_plus**: estende Abbonato, eseguendo overriding sul metodo toString e impostando il conto a 520€. Un abbonato plus può però iscriversi ai corsi tramite il metodo aggiungi\_corso (varargs). I corsi vengono gestiti tramite una lista ordinabile di corsi presente in ogni abbonato plus.
* **Corso**: è una classe astratta con all’interno tutti i campi e i metodi comuni a tutti i corsi.
* **Corso\_Cardio**: estende Corso, ha un prezzo di 50€.
* **Corso\_Yoga**: estende Corso, viene utilizzato il design pattern singleton affinché possa essere creato un solo corso di yoga il sabato.
* **Giorno**: enumerativo, utilizzato per gestire i giorni in cui sono svolti i corsi.
* **SequenzaOrdinabile**: sequenza di generici ordinabili. Implementa i metodi insert, toString, sort e find per sapere se un oggetto è già presente nella lista (anche se ha hashCode diverso).
* **AbbonatoIF**: interfaccia di abbonato dove vengono dichiarati i propri metodi.

Nel **main** per prima cosa vengono creati tre abbonati, andando così a testare la covarianza. Questi vengono inseriti in una SequenzaOrdinabile dove viene possibile la gestione.

Dopodiché vengono generati quattro corsi, inseriti anch’essi in una Sequenza Ordinabile.

Per provare i metodi implementati, avviene la stampa dei corsi (ordinati in base al giorno).

Dopo aver inserito i corsi ad un abbonato plus, si procede con l’output degli abbonati (in ordine di id univoco). Infine, viene applicato uno sconto ad un tipo di abbonato, e riavviene la stampa di tutti gli abbonati con tutti i loro dati.

C++

Anche in questo caso, il progetto si pone come un gestionale per abbonamenti e corsi di palestra.

Il software permette di inserire abbonati, sia normali che plus come in Java.

Anche in questo caso gli abbonati plus hanno al loro interno una sequenza di corsi, i quali però, a differenza del progetto in Java, possono essere creati inserendo un nome e un costo.

Oltre ai due tipi di abbonati è presente anche una nuova tipologia “iscritto” che rappresenta un semplice iscritto che non ha ancora effettuato un abbonamento.   
Ereditando in maniera privata la classe Abbonato, dal main è possibile accedere solo al nome, cognome e id tramite un metodo toString.

Le classi utilizzate per implementare il software sono le seguenti:

* **Abbonato**:al suo interno è presente un costruttore che prende in input nome, cognome, anno nascita, iscritto (true/false) e genera un nuovo abbonato con un conto di 500€.

L’anno di nascita, essendo costante, viene inserito tramite initializer list.

L’opzione iscritto serve nel caso che l’abbonamento sia generato da un utente già iscritto, dove quindi non è necessario generare un id univoco nuovo.  
Oltre a ciò sono presenti due metodi, ovvero toString (metodo virtuale) e applica\_sconto che effettua uno sconto del 10% ad un abbonato base.  
Come in ogni classe è presente un distruttore che, in questo caso, informa della disiscrizione dell’abbonato dalla palestra.

* **Abbonato\_plus**: eredita Abbonato in modo pubblico, ereditando così tutti i suoi campi e metodi pubblici.

È presente un campo privato, ovvero un oggetto di tipo SequenzaGenerica, dove sono presenti tutti i corsi a cui un Abbonato\_plus può iscriversi.  
Per iscriversi ai corsi è quindi servito un metodo aggiungi\_corso, il quale prende in input un oggetto di tipo initializer\_list di corsi (per consentire una aggiunta di più corsi con una solo chiamata del metodo, come varargs per Java) e li aggiunge alla SequenzaGenerica di corsi.

All’interno di questa classe sono ridefiniti i metodi applica\_sconto (effettua redefining), toString (effettua overriding) e il distruttore.

* **Corso**: è una semplice classe con i campi nome e costo (privato) e i metodi getCosto, e toString che ritornano solo il nome del corso.
* **Iscritto**: eredita Abbonato in modo privato. Grazie a ciò è possibile creare un iscritto con solo nome, cognome, anno nascita ma non è possibile utilizzare i metodi di Abbonato.

Per far sì che un iscritto diventi un abbonato è stato implementato un metodo abbonati e abbonati\_plus, il quale restituisce un Abbonato (o Abbonato\_plus) con lo stesso nome, cognome, anno di nascita e id.

* **SequenzaGenerica**: simile alla SequenzaOrdinabile sviluppata in Java. Questa al suo interno presenta un vettore di elementi di tipo K generici.

All’interno di questa classe sono implementati i metodi inserisci, che attraverso un initializer list (l’oggetto) è in grado di inserire diversi elementi K nel vettore, ed il metodo toString che restituisci i toString degli elementi K presenti nel vettore.

All’interno del **main**, per testare il funzionamento del software sono stati creati i due tipi di abbonati in due modi diversi, ovvero “normalmente” e in seguito passando per la classe Iscritto.

Dopo la stampa dei quattro iscritti, vengono creati due corsi, inseriti in seguito in una lista di corsi. Per la stampa di questi è stato sviluppato un metodo toString il quale utilizza un iterator per scorrere la lista.

Questi corsi sono stati inseriti nella sequenza di corsi di un abbonato plus tramite una sola chiamata del metodo, grazie all’initializer list presente in questo.

Come ultimo punto vengono effettuati i metodi applica\_sconto sulle due tipologie di abbonati e ristampati tutti gli abbonati per verificare il corretto funzionamento delle funzioni.

A differenza del software Java, qui non avviene la gestione degli abbonati tramite sequenza generica, se si volesse implementare risulterebbe comunque molto simile a quanto già svolto in Java.

**Haskell**

Per quanto riguarda Haskell, il progetto implementato è sicuramente più elementare.

Il campo di lavoro qui non è più la palestra con gli abbonamenti. Per mettere in evidenza gli aspetti della programmazione funzionale, sono state implementate delle piccole funzioni che utilizzano i paradigmi fondamentale di questo tipo di programmazione.

Durante lo sviluppo è stato usato il compilatore online replit, ed il codice generato è stato inserito in un file di testo.

Il progetto funziona in maniera molto semplice, chiede all’utente di inserire dei nomi di corsi universitari, in seguito, viene chiesto di inserire i CFU dei corsi appena inseriti ed infine è necessario inserire il voto ottenuto.

Inserendo questi dati in input il programma eseguirà una stampa di tutti i dati e calcolerà il voto di laurea ottenuto.

Per implementare ciò sono state create 5 funzioni principali e 1 di supporto.

Le funzioni sono le seguenti:

* **aggiungiCorsi:: Int-> IO [String]**

Prende in input un intero per capire se l’utente vuole continuare a inserire nuovi corsi oppure vuole fermarsi (1/0). Questa funzione viene richiamata da se stessa finché l’utente non inserisce uno 0. Infine, restituisce una lista contenente tutti i nomi dei corsi inseriti.

* **aggiungiCFU:: [String]->IO [Int]**

Prende in input un array di stringhe, che, in questo caso, sarebbe l’array contenente i nomi dei corsi inseriti prima. In questo caso non avviene il controllo della fine perché questa funzione si fermerà quando non ci saranno più corsi a cui inserire CFU. In output viene generata una lista di interi rappresentante i CFU.

* **aggiungiVoti:: [String]->IO [Int]**

Molto simile ad aggiungiCFU, prende la lista dei corsi e chiede all’utente di inserire i voti. Qui, per utilizzare il paradigma “if then else” è stato inserito un controllo sull’inserimento del voto che dovrà per forza essere compreso tra 18 e 30. Anche qui in output viene generata una lista di interi rappresentanti i voti inseriti.

* **unisci::[a]->[Int]->[Int]->[(a,Int,Int)]**

Questa funzione viene eseguita alla fine del programma, prendendo in input le tre liste sopra crete, le unisce in modo da ottenere una lista di questo tipo: [(Corso1,CFU1,Voto1),(Corso2,CFU2,Voto2)…]. Questa lista viene utile per avere un sommario dei dati inseriti.

* **calcolaVoto:: [Int]->[Int]->Int**

Prendendo in input la lista dei voti e dei CFU, viene restituito in output il voto previsto di laurea. Essendo una media ponderata è stato necessario creare una funzione complementare per il calcolo di ogni voto moltiplicato per il suo peso.

Unendo queste funzioni è stato ottenuto il programma finale, in cui avviene sia l’inserimento dei dati sia la stampa di tutti i risultati ottenuti da queste funzioni.