Relazione per il progetto di Programmazione in Java

Erica Cau, Alice Isola

4 giugno 2021

1 La Piscina *La Sirena*

La piscina *La Sirena* ha aperto nel 2015 e nel 2021 ha deciso di rimodernare il sistema di gestione degli ingressi.
Il seguente progetto propone una possibile soluzione a questa necessità, creando un programma in Java per la registrazione, la gestione degli ingressi e per permettere l'inserimento a posteriori degli ingressi degli anni precedenti.

2 Il package *Piscina*

Il nucleo del programma realizzato è contenuto nel `main` Piscina, che permette all'utente di accedere ai diversi metodi contenuti nella classe `GestionePiscina`. Le informazioni sugli ingressi sono racchiuse all'interno di un `Vector` contenente oggetti della classe `Ingressi`.
La scelta è ricaduta sulla classe `Vector`, in quanto essa è una struttura dati dinamica che permette di conservare un numero flessibile di elementi.
Per la gestione delle **date** degli ingressi si è scelto di utilizzare il package `java.time` per sfruttare facilmente l'interfaccia `Comparator` e utilizzare metodi per il trattamento delle date.

2.1 Il Main Piscina

All'interno del main viene inizializzato il vettore di ingressi che sarà poi manipolato con i metodi appositi della classe `GestionePiscina`.
Tale vettore, se presente un file di ingressi salvato in precedenza[1], verrà riempito con gli ingressi precedentemente salvati; in caso contrario, verrà notificato all'utente la mancanza del file salvato e gli verrà data la possibilità di inserire un nuovo ingresso prima di accedere al menù principale.

Il menù permette di eseguire le seguenti operazioni:

1. **aggiungere un ingresso:** a seconda della tipologia di utente, viene inserito l'ingresso con le informazioni correlate;
2. **visualizzare gli ingressi di uno specifico giorno:** l'utente inserisce una data specifica e gli verranno restituiti tutti gli ingressi effettuati in quel giorno;
3. **visualizzare gli ingressi di uno specifico mese:** l'utente inserisce il numero di un mese e l'anno e gli verranno restituiti tutti gli ingressi effettuati in quel mese ordinati cronologicamente;
4. **visualizzare gli ingressi di uno specifico utente abbonato:** l'utente inserisce il nome e il cognome di un utente abbonato e gli vengono restituiti tutti gli ingressi di quell'utente ordinati cronologicamente;
5. **visualizzare gli incassi dei biglietti venduti in uno specifico mese:** l'utente inserisce il numero di un mese e l'anno e può visualizzare gli incassi giorno per giorno di quel mese inserito;

[1] Nel nostro caso è stato chiamato "ingressiPiscina"

6. **visualizzare il numero di ingressi di persone abbonate effettuati in uno specifico mese**: l'utente inserisce un mese e un anno e visualizza il numero di ingressi effettuati da soli utenti abbonati;

7. **visualizzare gli ingressi presenti al momento nel vettore**: il programma stampa gli ingressi presenti nel vettore;

8. **salvare gli ingressi inseriti sul file**: comando che salva su file le modifiche eseguite sul vettore;

9. **uscire dal programma:** comando che termina l'esecuzione del programma; è presente un salvataggio su file prima della chiusura.

2.2 La modellazione degli ingressi

Gli ingressi vengono modellati all'interno della superclasse `Ingressi` che contiene la variabile d'istanza `Localdate data` e viene estesa attraverso le due sottoclassi `IngressiAbbonati` e `IngressiNonAbbonati` che modellano le differenti tipologie di ingressi.
La classe `IngressiAbbonati` è descritta dalla data ereditata dalla superclasse `Ingressi` e da un oggetto della classe `UtenteAbbonato`. Allo stesso modo, la classe `IngressiNonAbbonati` contiene la data e un oggetto della classe `UtenteNonAbbonato`.
Queste due ultime sottoclassi contengono i metodi `get` e `set` per sfruttare la proprietà di incapsulamento. Inoltre, viene sfruttato il meccanismo dell'*overriding* per sovrascrivere il metodo `toString()` per stampare le informazioni sugli ingressi.

2.3 La modellazione degli utenti

Il concetto di utente è stato modellato attraverso le classi `UtenteAbbonato` e `UtenteNonAbbonato`.

Nel primo caso, è stato necessario inizializzare come variabili d'istanza il nome e il cognome dell'utente; per confrontare i diversi utenti abbonati è stato sovrascritto il metodo `equals()` di Java per confrontare la corrispondenza tra il nome e il cognome inseriti dall'utente, come richiesto da alcuni metodi della classe `GestionePiscina`.

Il caso dell'utente non abbonato è stato più delicato da trattare in quanto esso viene descritto attraverso il costo del biglietto per entrare nella piscina.
Infatti, il prezzo del biglietto intero, impostato a 3 euro (con una costante `double`), può essere sottoposto a riduzioni.
Sono state previste due diverse categorie di riduzioni: in base all'età e in base allo status di studente[2].
La riduzione di prezzo per età ha reso necessaria la creazione di una variabile d'istanza eta e di metodi ausiliari per il controllo della stessa. Tali operazioni sono state effettuate per lo status di studente, in cui viene richiesta all'utente l'età anagrafica (minore di 24 anni) e la conferma di essere o meno studente.

2.4 La gestione della Piscina

La classe `GestionePiscina` contiene i metodi per la gestione del vettore in cui sono conservati gli ingressi.
Per permettere l'ordinamento degli ingressi contenuti nel vettore, si è sfruttata l'interfaccia `Comparator` di `java.util`.

Di seguito, i metodi presenti:

1. metodo `AggiungiIngresso`: chiede all'utente di inserire la data dell'ingresso che può essere quella attuale o una precedente. Se la data rientra nell'ultimo periodo, viene richiesto un controllo della temperatura della persona che vuole accedere alla piscina.
   Viene poi data all'utente la possibilità di specificare se l'ingresso da inserire è quello di un abbonato (per cui verranno richiesti nome e cognome) oppure di un non abbonato (a cui verrà chiesta l'età e/o lo status di studente).

[2]Il prezzo è stato impostato a 2.5 euro per gli studenti e a 2.0 euro per età pari o inferiore a 12 anni e pari o superiore a 65 anni

2. metodo `IngressiGiornalieri`: metodo che chiede all'utente di inserire una data e restituisce la lista di tutti gli ingressi effettuati in quel giorno (ordinati con il `Comparator`) e il numero totale di ingressi nello stesso.

3. metodo `IngressiMensiliOrdinati`: metodo che chiede all'utente di inserire il mese e l'anno, controlla le date di tutti gli ingressi nel vettore e, se questa corrisponde viene stampato l'ingresso. Nel caso in cui non siano presenti ingressi nella data inserita, questo viene notificato all'utente.

4. metodo `IngressiUtenteAbbonato`: metodo che chiede all'utente di inserire il nome e il cognome di un abbonato, controlla che un ingresso sia stato effettuato da un abbonato e, sfruttando l'overriding del metodo `equals()` (effettuato nella classe utente abbonato), restituisce tutti gli ingressi di quello specifico utente abbonato.

5. metodo `IncassiMensili`: scorrendo il vettore di ingressi, si vanno a controllare i prezzi dei biglietti degli ingressi senza abbonamento di uno specifico mese; inoltre, vengono stampati gli incassi totali di quel mese e il numero di biglietti con prezzo ridotto.

6. metodo `IngressiAbbonamentoMensili`: scorrendo il vettore di ingressi, si vanno a controllare gli ingressi dei soli abbonati in uno specifico mese e si stampa infine il numero totale di ingressi. Nel caso in cui non siano presenti ingressi nel mese inserito, questo viene notificato all'utente.

Sono stati creati poi diversi **metodi ausiliari** volti al controllo delle date di chiusura della piscina (`ChiusuraPiscina()`), al controllo della temperatura dell'utente (`ControllaTemperatura()` e al controllo dell'età (`ControllaEta()`) per impedire l'inserimento di ingressi di utenti con età troppo elevate o inferiori a 0. [3]

Caso più delicato è stata la gestione delle date per evitare errori nel formato inserito dall'utente che avrebbero bloccato l'esecuzione del programma. Sono stati quindi creati i metodi `ControllaMese()` e `ControllaGiornoMese` che aggiungono in automatico uno 0 per mesi compresi tra gennaio e settembre e giorni compresi tra 1 e 9.
È stato inoltre creato un metodo (`ChiediData()`) per la richiesta della data all'interno del metodo `AggiungiIngressi()`, il quale controlla che la data non sia posteriore al giorno attuale. Per far questo è stato sfruttato il metodo `now()` del package `java.time`.

2.5 La creazione delle Eccezioni

Sono state infine create alcune **eccezioni**per la gestione di situazioni anomale particolari quali la data sintatticamente errata (`DataErrataException`), l'età errata (`EtaCorrettaException`) e la piscina chiusa (`PiscinaChiusaException`) che restituiscono all'utente messaggi ad hoc in caso di errore.

[3] L'età massima per convenzione è stata stabilità a 150 anni