## esercizio preparazione per 2° test in itinere 2023, 15 giugno

Si definisca un insieme di classi per rappresentare i clienti di una Piscina:

Ogni cliente è caratterizzato dai seguenti dati:*, nome, data di nascita,e-mail (chiave univoca)*

- *i clienti sono ordinati naturalmente per nome, data di nascita, e-mail*

Nel momento in cui un utente si iscrive alla piscina, viene creata una subscription.
*Subscription* è identificata univocamente da un **codice_utente**, un intero progressivo. Conterrà inoltre le informazioni sul cliente e il livello di abilità esprimibile come un intero da 0 a 4 [variante: enum per identificare il livello abilità (es. base, principiante, brevetto, esperto, istruttore)]

Risulta chiaro quindi che i clienti iscritti hanno ognuno un livello di abilità (0-4): 0 rappresenta un livello di abilità base, mentre ai livelli da 1 a 4 corrisponde un livello avanzato.

La piscina mantiene le informazioni relative alle subscription dentro un Archivio una HashMap<Integer, Subscription> con chiave il **codice_utente**.

[variante che si può provare come altro esercizio: HashMap<String, Subscription> con chiave la email utente, oppure TreeSet<Subscription> sfruttando l'ordinamento]

Operazioni di gestione dell'archivio:

1) inserimento/iscrizione di un nuovo cliente verificando che non sia già presente:
   sono da considerarsi uguali i clienti/subscription con la stessa e-mail
2) lettura di un elemento in base a codice_utente [ o email, in base alla scelta della chiave nelle varianti]
3) rimozione dall'archivio della piscina di un cliente che vuole lasciare la piscina
4) Creazione di una collezione TreeMap<Integer,subscription> [ variante da provare ArrayList<Subscription> ] con le iscrizioni di clienti con età inferiore a 18 (under 18) e di livello avanzato (livello 1-4).

**Gestione delle gare**
Nel contesto della piscina, è prevista la gestione di due gare: una riservata agli iscritti under18 con livello di abilità>=3 e un'altra per iscritti under18 con qualunque livello avanzato (1-4)

I due tipi di gara hanno ciascuna una capienza massima di 8 partecipanti. Raggiunto il limite di capienza, la richiesta di partecipazione è messa in attesa fintanto che non venga liberato un posto.

Nota: Ogni gara è ovviamente costituita da iscritti distinti ma è possibile, per uno stesso iscritto, partecipare a entrambe le gare. Ogni gara è rappresentata da un monitor RegistroGara con una collezione a scelta dello studente su cui agiscono i thread descritti di seguito.

Per simulare la gestione gare è richiesto realizzare:

- 1 Thread - GestoreGara (Produttore) estrae random un'integer che può essere una chiave presente nel treemap degli iscritti under18 e se presente inserisce nella collection dei prenotati per accedere alla singola gara, altrimenti riprova ad estrarre nuovo intero finchè non ne trova uno assegnato come chiave nella TreeMap del punto 4 [oppure più semplicemente estrae un gli indici della ArrayList se si sta usando quell al punto 4] (con ciò viene simulata la richiesta di un iscritto under18 che chiede di/si prenota per partecipare ad una delle 2 gare).

- 1 Thread - Consumatore estrae e rimuove a sorte un partecipante da una delle gar.. Il criterio

di estrazione del partecipante è a discrezione dello studente

- 1 Thread - Eraser che simula la conclusione di una gara e svuota la collection di RegistroGara. Il registro sarà comunque ripopolato dal GestoreGara successivamente.
- 1 Thread - Segreteria (lettore) che notifica a tutti gli iscritti l'attuale lista dei partecipanti iscritti in un RegistroGara, in tempo reale
- 1 Thread - Statista (lettore) valuta l'età media dei partecipanti ad una gara in tempo reale

NOTA: tutti i thread vanno in sleep con una dinamica decisa dallo studente