Programmazione Object Oriented

Esercitazioni aggiuntive

Esercizio – Scuola di Sci

Scrivere l'interfaccia `Course` inserendola nel pacchetto `skischool`. L'interfaccia contiene i metodi:

- `String getCourseName();`
- `int getMinimumNumberOfParticipants();`
- `int getMaximumNumberOfParticipants();`
- `int getNumberOfParticipants();`
- `boolean addParticipant();`
- `boolean isCourseActivated();`

Scrivere le classi `SkiCourse` e `SnowboardCourse` che implementano l'interfaccia `Course`.
Inserire le classi nel pacchetto `skischool`.

Le classi devono essere implementate in modo da memorizzare un identificativo, il numero di partecipanti, la data di inizio corso e la durata (il numero di giorni). Lanciare un'eccezione non controllata `BadArgumentException` se la durata è negativa al momento della creazione.

Il corso di sci viene attivato solo se si raggiungono almeno quattro partecipanti e può avere al massimo otto partecipanti.

Il corso di snowboard viene attivato solo se si raggiungono almeno sei partecipanti e può avere al massimo dieci partecipanti.

Il metodo `addParticipant` restituisce `true` se è stato possibile aggiungere il partecipante al corso (il numero totale di partecipanti è inferiore al numero massimo consentito) e `false` altrimenti.

Il metodo `isCourseActivated` restituisce `true` se è stato raggiunto il numero minimo di partecipanti e quindi il corso è attivato.

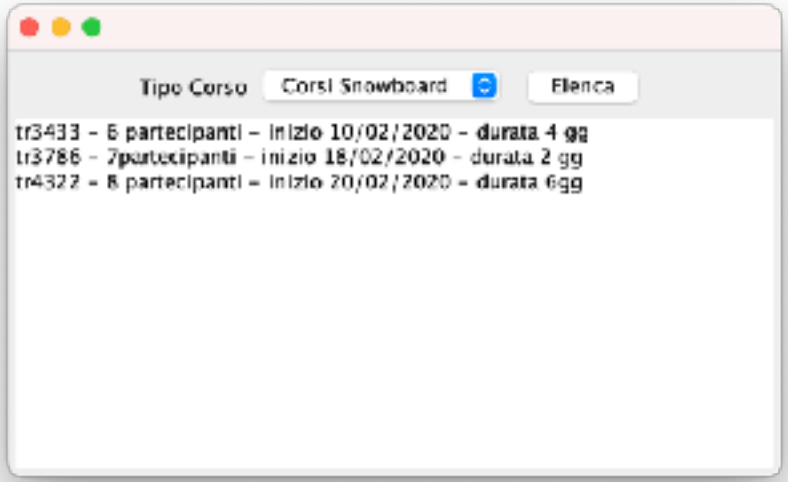
Esercizio – Scuola di Sci

Scrivere la classe `CourseList` (nel pacchetto `skischool`) che modelli i corsi offerti da una scuola di sci attraverso i seguenti metodi:

- `void addCourse(Course c)` che inserisca un corso nell'archivio in modo ordinato rispetto alla data.
- `Course getCourse(int i)` che restituisca l'i-esimo corso nell'archivio;
- `Course removeCourse(int i)` che rimuova l'i-esimo corso dall'archivio restituendolo.
- `ArrayList<Course> getCourses(String type)` che restituisca tutti i corsi in base al tipo (`SkiCourse` o `SnowboardCourse`).

Aggiungere poi il codice per

- creare una `CourseList` con almeno cinque oggetti `SkiCourse` e cinque oggetti `SnowboardCourse` con valori casuali (tutti i corsi devono essere attivi).
- Realizzare un'interfaccia grafica che consenta di visualizzare tutti gli `SkiCourse` o tutti gli `SnowboardCourse` utilizzando una `JCombobox`.



Esercizio – Mensa

Implementare la classe `Card` che modelli i vari tipi di tesserino utilizzati per una mensa universitaria. Ogni tesserino è caratterizzato da codice, nome, cognome, e dalla proprietà `active` (quest'ultima è una variabile booleana indicante se il tesserino è utilizzabile). Fornire i metodi:

- `activate()` che setti a `true` lo stato della variabile attivo (se `active` è già `true` lancia l'eccezione `RunTimeException`).
- `deactivate()` che setti a `false` lo stato della variabile attivo (se `active` è già `false` lancia l'eccezione `RunTimeException`).

Implementare poi due sottoclassi:

- `StudentCard` caratterizzata da matricola, scadenza, saldo, fascia, bonus con i metodi
  - `double calcolaPrezzo()` che calcoli il prezzo del pasto di uno studente in base alla fascia: coloro che appartengono alla fascia A pagano 2.50€, mentre coloro che appartengono alla fascia B pagano 1.50€. Inoltre gli studenti vincitori di borse di studio (i cui tesserini hanno la variabile `bonus` settata a `true`) hanno lo sconto di 1€.
  - `boolean isBonus()` che restituisca `true` se lo studente ha vinto una borsa di studio.
  - `double simulatePayment()` che simuli il pagamento di un pasto. Se il tesserino è scaduto lancia l'eccezione controllata `ExpiredCardException`, altrimenti sottrae al saldo il costo del pasto. Nel caso in cui il saldo è insufficiente per pagare il pasto viene lanciata l'eccezione non controllata `InsufficientBalanceException`. Il valore restituito corrisponde all'importo pagato dallo studente.
  - `void pay(double x)` che aggiorni il saldo con la somma x passata come argomento. Se la somma da versare è negativa lancia l'eccezione `RunTimeException`.
- `StaffCard` caratterizzata dalle variabili `department`, `spentAmount`, `category` (che può essere `professor` or `administrative`). Corredare la classe con i metodi
  - `void pay(double x)` che aggiunga a `spentAmount` il costo del pasto che dipende dal valore di categoria. Nel caso di `professor` l'importo del pasto è 1.60€, nel caso di `administrative` l'importo è 4.00€. Il metodo restituisce l'importo pagato dal personale.
  - `void changeCategory()` che modifichi la categoria del tesserino.

Esercizio – Mensa

Scrivere la classe `MealPayment` che modelli una collezione di `Card` e fornisca i seguenti metodi:

- `void addCard(Card t)` che inserisca un tesserino nell'archivio.
- `boolean useCard(int code)` che simuli il pagamento di un pasto per la persona in possesso del tesserino con codice `code`. Il metodo non gestisce le eventuali eccezioni lanciate. Restituisce `true` se il codice è presente nell'archivio, `false` altrimenti.
- `double computeTotal()` che restituisca la somma incassata fino a quello istante.
  N.B. *Utilizzare una variabile che tenga traccia di tale somma durante i pagamenti.*
- `MealPayment getCardByType(int x)` che restituisca l'elenco dei tesserini di una certa tipologia (`x=0` indica la tipologia `StudentCard` e `x=1` una `StaffCard`). Per valori di x diversi da `0` e `1` viene la lanciata l'eccezione controllata `InvalidParameterException`.
- `double getSpentAmount()` che restituisca la somma spesa da tutti i possessori di `Card`.

Implementare un'interfaccia grafica per

- caricare da un file una lista di tesserini;
- effettuare il pagamento di un pasto (chiedere di inserire solo il codice del tesserino) e visualizzare la somma incassata.