Esercizio 1

Implementare una classe `GestioneLibri` che permetta di gestire una raccolta di libri. Ogni libro è caratterizzato da un titolo (String), un autore (String) e un anno di pubblicazione (int).

La classe `GestioneLibri` deve avere i seguenti metodi:

1. Un costruttore che accetta la dimensione massima della raccolta di libri.
2. Un metodo `aggiungiLibro` che accetta un titolo, un autore e un anno di pubblicazione e aggiunge il libro alla raccolta, se c'è spazio disponibile. Restituisce true se l'operazione ha avuto successo, false altrimenti.
3. Un metodo `rimuoviLibro` che accetta un titolo e rimuove il libro dalla raccolta, se presente. Restituisce true se l'operazione ha avuto successo, false altrimenti.
4. Un metodo `cercaLibro` che accetta un titolo e restituisce l'indice del libro nella raccolta, se presente. Se non viene trovato, restituisce -1.
5. Un metodo `stampaLibri` che stampa a video tutti i libri presenti nella raccolta, con il formato "Titolo: <titolo>, Autore: <autore>, Anno: <anno>".
6. Un metodo `libriPerAutore` che accetta un autore e restituisce un array di stringhe contenente i titoli dei libri scritti da quell'autore, presenti nella raccolta.
7. Un metodo `libriPremaAnno` che accetta un anno e restituisce un array di stringhe contenente i titoli dei libri pubblicati prima di quell'anno, presenti nella raccolta.

Crea una classe `Main` separata per testare la funzionalità della classe `GestioneLibri`.

Suggerimenti:

- Puoi utilizzare un array di oggetti Libro (da creare) per memorizzare i libri nella raccolta.
- Ricorda di gestire correttamente i casi limite, come la raccolta piena o vuota.

Questo esercizio ti permetterà di praticare la programmazione orientata agli oggetti, la gestione delle raccolte di dati con gli array e l'implementazione di metodi per manipolare e accedere ai dati.

Esercizio 2

Implementare un sistema di gestione di un'agenzia di viaggi. Il sistema deve includere le seguenti classi:

1. `Destinazione`: Rappresenta una destinazione di viaggio, con attributi come nome, paese, descrizione e costo base (in euro).
2. `Viaggio`: Rappresenta un viaggio organizzato dall'agenzia, con attributi come destinazione, data di partenza, data di ritorno, costo totale (calcolato in base al costo base della destinazione e alla durata del viaggio), e un identificatore univoco.
3. `Cliente`: Rappresenta un cliente dell'agenzia, con attributi come nome, cognome, email, e un identificatore univoco.
4. `Prenotazione`: Rappresenta una prenotazione di un viaggio da parte di un cliente, con attributi come viaggio, cliente, data di prenotazione e stato (prenotato, confermato, cancellato).
5. *`AgenziaViaggi`*: Gestisce l'intera agenzia, con metodi per aggiungere/rimuovere destinazioni, creare viaggi, effettuare prenotazioni e cancellazioni, e altre funzionalità come:
    - Cercare viaggi per destinazione, data di partenza o costo
    - Elencare le prenotazioni di un determinato cliente
    - Calcolare il fatturato totale dell'agenzia in un determinato periodo
    - Generare statistiche sui viaggi più prenotati o sulle destinazioni più popolari

Inoltre, implementare le seguenti regole di business:

- Il costo totale di un viaggio è calcolato in base al costo base della destinazione e alla durata del viaggio (ad esempio, costo base + 10% del costo base per ogni giorno di durata).
- Una prenotazione può essere effettuata solo se il viaggio non è sold-out (ad esempio, con un limite massimo di 20 prenotazioni per viaggio).
- Una prenotazione può essere cancellata senza penali fino a 14 giorni prima della data di partenza. Oltre tale periodo, viene applicata una penale pari al 20% del costo totale del viaggio.
- Uno sconto del 10% sul costo totale viene applicato ai viaggi prenotati almeno 6 mesi prima della data di partenza.

Crea una classe `Main` separata per testare il funzionamento del sistema, creando istanze di destinazioni, viaggi, clienti ed effettuando operazioni di prenotazione, cancellazione e generazione di statistiche.