Esercizio 1 – Gestione Conto Corrente

Sviluppare un'applicazione Java che simuli la gestione dei conti correnti di una banca, integrando i seguenti requisiti:

1. Creare una classe astratta o concreta `ContoCorrente` con attributi come `IBAN`, `saldo` e `conteggioMovimenti`, oltre ai metodi necessari per effettuare versamenti, prelievi e stampare le informazioni del conto.
2. Realizzare due classi figlie di `ContoCorrente`: `ContoCorrentePrivato` e `ContoCorrenteGiuridico`, che estendano la classe padre e aggiungano attributi specifici come `codiceFiscale`, `nome`, `cognome` per i conti privati, e `partitaIVA`, `ragioneSociale` per i conti giuridici. Implementare costruttori, getter/setter e il metodo `stampaInfo`() in override.
3. Implementare una classe `Banca` che contenga un `ArrayList` di oggetti `ContoCorrente` (`listaConti`), consentendo di inserire, cancellare e ricercare conti sia per IBAN che per nome/cognome o `partitaIVA`/`ragioneSociale`. Includere anche metodi per calcolare il totale dei saldi e stampare le informazioni di tutti i conti.
4. Nella classe Banca, realizzare metodi per:
    - `inserisciConto`(ContoCorrente) e `inserisciConto`(String IBAN, double saldo, int movimenti) (overloading)
    - `cancellaConto`(String IBAN)
    - `modificaConto`(String IBAN) (restituisce il ContoCorrente trovato o null)
    - `cercaConto`(String IBAN) e `cercaConto`(String cognome, String nome) (overloading)
    - `totaleSaldi()`
    - `stampaInfoConti()`
5. Creare una classe `Main` per istanziare un oggetto Banca, inserire alcuni `ContoCorrente` (istanze di ContoCorrentePrivato e ContoCorrenteGiuridico) e testare i vari metodi implementati.
6. Implementare costruttori (vuoto, con parametri, copia) per tutte le classi e gestire correttamente l'ereditarietà e il polimorfismo nell'utilizzo dell'ArrayList `listaConti`.
7. Aggiungere controlli e gestione degli errori dove necessario, ad esempio per evitare prelievi superiori al saldo disponibile.

Esercizio 2 – Gestione calciatori

1. Definire una classe **Partita** che contenga due oggetti di tipo **Squadra** (una per ciascuna squadra).
2. Implementare un metodo **effettuaSostituzione(String nomeUscita, String nomeEntrata)** all'interno della classe **Partita** che consenta di effettuare una sostituzione durante la partita. Questo metodo deve cercare i giocatori nelle squadre, rimuovere il calciatore che esce dalla partita e inserire il calciatore che entra.
3. Creare una classe **MainPartita** con il metodo **main** che simuli una partita di calcio. In questa simulazione, vengono create due squadre con un numero di calciatori e portieri. Durante la partita, si effettuano diverse sostituzioni utilizzando il metodo implementato nella classe **Partita**.

Dettagli:

- Ogni squadra ha un massimo di 11 giocatori in campo, tra cui un portiere.
- Durante la partita, ogni squadra può effettuare un massimo di 3 sostituzioni.
- Ogni calciatore ha un nome, un numero di maglia e un ruolo (attaccante, difensore, centrocampista, portiere, ecc.).
- Ogni sostituzione deve essere registrata e stampata a video insieme alla formazione aggiornata delle squadre dopo la sostituzione.

Esercizio 3 – Gestione ospedaliera

Implementare un sistema software per la gestione di un ospedale. Il sistema deve consentire la registrazione di medici, pazienti, visite mediche e prenotazioni. Deve anche essere in grado di assegnare le prenotazioni ai medici disponibili in base alla loro specializzazione e alla loro agenda.

1. **Classe Persona:** Implementare una classe **Persona** che rappresenti una persona generica con attributi come nome, cognome e codice fiscale.
2. **Classe Medico:** Derivare la classe **Medico** dalla classe **Persona**. Ogni medico dovrebbe avere attributi aggiuntivi come specializzazione e lista di prenotazioni. Implementare metodi per aggiungere prenotazioni, visualizzare le prenotazioni attive e controllare la disponibilità per nuove prenotazioni.
3. **Classe Paziente:** Derivare la classe **Paziente** dalla classe **Persona**. Ogni paziente dovrebbe avere attributi aggiuntivi come diagnosi e lista di visite mediche. Implementare metodi per aggiungere visite mediche e visualizzare la cartella clinica.
4. **Classe VisitaMedica:** Implementare una classe **VisitaMedica** che rappresenti una visita medica effettuata da un medico a un paziente. Ogni visita dovrebbe includere informazioni come diagnosi, trattamento e data.
5. **Classe Prenotazione:** Implementare una classe **Prenotazione** che rappresenti la prenotazione di una visita medica da parte di un paziente. Ogni prenotazione dovrebbe includere informazioni come il paziente, il medico, il trattamento e la data.
6. **Classe Ospedale:** Implementare una classe **Ospedale** che contenga tutti i medici e i pazienti afferenti all'ospedale. Deve includere funzionalità per aggiungere medici, pazienti e gestire le prenotazioni, assegnandole ai medici disponibili in base alla loro specializzazione e alla loro agenda.
7. **Classe Principale:** Implementare una classe **Principale** che contenga il metodo **main**. In questa classe, creare un'istanza di **Ospedale** e eseguire le operazioni di test per verificare il corretto funzionamento del sistema.