Laboratorio di Linguaggi e Programmazione Orientata agli Oggetti (LPO)

25 novembre 2019

a.a. 2019/2020

1. Leggere le spiegazioni su AulaWeb per compilare ed eseguire programmi Java e fare qualche prova con la classe `HelloWorld` definita nelle spiegazioni.
2. Definire la classe `Person` secondo i seguenti requisiti:
   - ogni istanza di `Person` è dotata dei seguenti attributi:
     - nome (`name`) di tipo `String`, obbligatorio e non modificabile;
     - cognome (`surname`) di tipo `String`, obbligatorio e non modificabile;
     - codice fiscale (`socialSN`), di tipo **`long`**, obbligatorio e non modificabile; tale attributo non può essere negativo e identifica ogni istanza della classe: due oggetti di tipo `Person` hanno lo stesso codice fiscale se e solo se sono lo stesso oggetto.
     - coniuge (`spouse`) di tipo `Person`, opzionale e modificabile.
   - nome e cognome devono essere stringhe valide rispetto all'espressione regolare `[A-Z][a-z]+( [A-Z][a-z]+)*`; usare il metodo di istanza `matches`[1] della classe predefinita `String`;
   - i seguenti dati devono essere accessibili in sola lettura dall'esterno della classe: nome, cognome e coniuge (**`null`**, in caso quest'ultimo non sia definito);
   - deve essere definito il metodo di query `isSingle()`, che restituisce **`true`** se e solo se la persona non ha un coniuge;
   - deve essere definito il metodo **`void`** `join(Person p1, Person p2)` che permette di sposare due persone `p1` e `p2`, con la pre-condizione che non siano la stessa persona e siano entrambi senza coniuge;
   - deve essere definito il metodo **`void`** `divorce(Person p1, Person p2)` che permette di far divorziare due persone `p1` e `p2`, con la pre-condizione che siano coniugi l'uno dell'altro;
   - decidere sull'opportunità che `join` e `divorce` siano metodi di istanza o di classe.

   Esprimere gli invarianti della classe e definire il costruttore più opportuno che garantisca la verifica degli invarianti.

   Definire metodi di classe **`private`** ausiliari per validare pre-condizioni e assicurare che gli invarianti vengano rispettati.

   Scrivere un semplice programma di test che fa uso del costrutto **`assert`** nel metodo `main` della classe separata `Test`.
3. Definire la classe `CreditAccount` (conto corrente) secondo i seguenti requisiti:
   - ogni istanza di `CreditAccount` è dotata dei seguenti attributi:
     - limite minimo di giacenza del conto (`limit`) di tipo **`int`**, obbligatorio e modificabile; tale valore è espresso in centesimi e può essere anche negativo;
     - saldo del conto (`balance`) di tipo **`int`**, obbligatorio e modificabile; tale valore è espresso in centesimi e non può essere mai inferiore al limite minimo di giacenza;
     - intestatario del conto (`owner`), di tipo `Person`, obbligatorio e non modificabile;
     - codice del conto (`id`) di tipo **`long`**, obbligatorio e non modificabile; tale attributo non può essere negativo e identifica ogni istanza della classe: due oggetti di tipo `CreditAccount` hanno lo stesso codice se e solo se sono lo stesso oggetto.
   - i seguenti dati devono essere accessibili in sola lettura dall'esterno della classe: limite di giacenza, saldo, intestatario e codice del conto;

[1] `https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/String.html#matches(java.lang.String)`

- deve essere definito il metodo `int deposit(int amount)` che permette di depositare sul conto una somma positiva `amount` (espressa in centesimi) e che restituisce il saldo totale dopo il versamento di tale somma;
- deve essere definito il metodo `int withdraw(int amount)` che permette di prelevare dal conto una somma positiva `amount` (espressa in centesimi) e che restituisce il saldo totale dopo il prelevamento di tale somma; l'operazione è consentita solo se il saldo non scende al di sotto del limite di giacenza;
- deve essere definito il metodo `void setLimit(int limit)` che permette di modificare il limite minimo di giacenza del conto; l'operazione è consentita solo se il saldo corrente non scende al di sotto del nuovo limite di giacenza;
- deve essere definito il dato `default_limit`, non modificabile, ma accessibile all'esterno della classe, che stabilisce che il limite di giacenza predefinito è pari a 0;
- deve essere definito il metodo factory

  ```
  CreditAccount newOfLimitBalanceOwner(int limit, int balance, Person owner)
  ```

  che permette di aprire un conto restituendo un nuovo oggetto di tipo `CreditAccount` con limite di giacenza `limit` (in centesimi), saldo iniziale `balance` (in centesimi) e intestatario `owner`; il conto può essere aperto solo se il saldo iniziale è positivo.
- deve essere definito il metodo factory

  ```
  CreditAccount newOfBalanceOwner(int limit, int balance, Person owner)
  ```

  che permette di aprire un conto restituendo un nuovo oggetto di tipo `CreditAccount` con limite di giacenza predefinito, saldo iniziale `balance` (in centesimi) e intestatario `owner`; il conto può essere aperto solo se il saldo iniziale è positivo.
- decidere sull'opportunità che `deposit`, `withdraw`, `setLimit`, `newOfLimitBalanceOwner` e `newOfLimitBalanceOwner` siano metodi di istanza o di classe.

Esprimere gli invarianti della classe e definire i costruttori più opportuni che garantiscano la verifica degli invarianti.

Definire metodi di classe **`private`** ausiliari per validare pre-condizioni e assicurare che gli invarianti vengano rispettati.

Scrivere un semplice programma di test che fa uso del costrutto **`assert`** nel metodo `main` della classe separata `Test`.