Corso di Laurea in Informatica
Prova Scritta di Metodologie di Programmazione - Primo Canale

Sapienza Università di Roma

16 Giugno 2021

Durante l'esame non è consentito l'utilizzo di alcunché. Non è consentito inoltre l'utilizzo della matita o di penne il cui colore sia diverso dal nero o dal blu. Il ritiro dalla prova equivale al mancato superamento dell'esame.

Nome e Cognome:

Matricola:

1. Descrivere il meccanismo dell'ereditarietà in modo esaustivo spiegando il suo funzionamento e come i modificatori di visibilità si relazionano a tale meccanismo. Produrre un esempio (con codice minimale) che concretizzi quanto richiesto.

   L'ereditarietà è uno dei meccanismi principali della programmazione ad oggetti. Essa prevede la definizione di una nuova classe, chiamata *classe derivata* o *sottoclasse*, come specializzazione di un'altra, chiamata *classe base* o *superclasse*. In questo modo, viene instaurata una gerarchia attraverso la quale la classe base modella un concetto generico, mentre la classe derivata definisce un concetto più specifico. Tale sottoclasse può aggiungere funzionalità proprie e/o ridefinire il funzionamento di metodi esistenti, offrendo la possibilità di fare uso del polimorfismo. In generale, la classe derivata è in grado di ereditare i campi con modificatore di visibilità di tipo *public*, *default* e *protected*[1], mentre quelli contrassegnati con *private* saranno solamente visibili all'interno della classe base.
   I vantaggi dell'ereditarietà si configurano sia a livello di progettazione che a livello pratico. Infatti, grazie a tale meccanismo è possibile evitare la creazione di classi disgiunte, disincentivando la duplicazione di codice e permettendo il riuso di funzionalità già presenti nel programma. Tutto ciò porta a semplificare la costruzione di nuove classi, facilitando la manutenzione e garantendo la consistenza delle interfacce.

2. Spiegare le principali differenze tra l'utilizzo di un array e della classe *ArrayList* nella gestione di una sequenza di elementi.

   Alcune delle differenze principali tra un array generico e un oggetto di tipo *ArrayList* sono le seguenti:

   - Un array è dichiarato con una grandezza fissata a priori, mentre la classe *ArrayList* permette il ridimensionamento dinamico a fronte di un'aggiunta o rimozione di un elemento;
   - Un array si presta ad una gestione della multidimensionalità (matrici) più semplice, mentre per gli oggetti di tipo *ArrayList* tale gestione risulterebbe più articolata;

[1] La spiegazione sui limiti di visibilità per ciascuno di questi operatori è da intendersi come parte della risposta sebbene non sia stata inclusa.

- La classe *ArrayList* offre tutta una serie di metodi, ottenuti tramite l'implementazione dell'interfaccia *List*, che permettono una gestione più semplice e consistente della collezione rappresentata. Ad esempio, la rimozione di un elemento dalla lista ed il suo conseguente ridimensionamento avvengono per mezzo del metodo *remove*, il quale necessita di un solo parametro di input rappresentante la posizione dell'elemento da rimuovere. Nel caso di un array, la rimozione ed il successivo ridimensionamento sono operazioni a carico del programmatore.

3. Per ogni costrutto iterativo, indicare il numero di volte per il quale viene eseguito il suo corpo. Se non diversamente indicato, si assume che la variabile contatore non venga modificata all'interno del corpo di ciascun costrutto iterativo.

   (a) `for(int i = 2; i <= 9; i++){...}`

   (b) `for(int i = 0; i < 10; i++){...}`

   (c) `for(int i = 10; i > 0; i--){...}`

   (d) `for(int i = -10; i >= 0; i++){...}`

   (e) `for(int i = -2; i >= 0; i++){...}`

   (f) `for(int i = -8; i <= 3; i = i + 4){...}`

   (g)
   ```
   for (int k = 0; k < 20; k+=2) {
   if (k + 3 == 1) System.out.println(k + " ");
   }
   ```

   - a): 8
   - b): 10
   - c): 10
   - d): 0
   - e): 0
   - f): 3
   - g): 10

   Il codice di tale esercizio è disponibile all'interno della cartella *es_3*.

4. Realizzare la classe *Customer* al fine di gestire le informazioni relative ad un cliente di una campagna di promozione commerciale. Tale campagna si articola in questo modo: dopo aver effettuato una serie di acquisti per almeno $100, il cliente riceverà uno sconto di $10 sull'acquisto successivo. Progettare quindi i seguenti metodi:

   ```
   public void makePurchase(double amount) // registra un acquisto
   public boolean discountReached() // true se e solo se l'utente ha raggiunto lo sconto
   ```

   Al fine di verificare la correttezza delle propria implementazione, si realizzi un programma di collaudo (test). Esso dovrà rappresentare la seguente situazione: un cliente dovrà ottenere uno sconto, il quale lo utilizzerà per fare degli acquisti. L'importo totale derivante da tali acquisti dovrà essere maggiore di $90 ma minore di $100 in modo tale da non poter ottenere uno sconto. Successivamente, l'utente dovrà effettuare un altro acquisto, il quale lo porterà a godere dello sconto sull'acquisto successivo.

   La soluzione di tale esercizio è disponibile all'interno della cartella *es_4*.

5. Il responsabile di una filiale bancaria ha accesso ad un database contenente tutte le informazioni sui conti correnti dei propri clienti. Tali informazioni sono rappresentate in formato tabellare, in cui ciascuna riga contiene i seguenti campi separati da uno spazio:

numeroDiConto1 Nome1 Cognome1 saldo1
numeroDiConto2 Nome2 Cognome2 saldo2
...

Scrivere un programma che legga un file di testo con questa struttura, segnalando un opportuno errore in caso di file inesistente. Successivamente, visualizzare:

- Le informazioni relative al conto con il saldo maggiore;
- Il saldo medio, calcolato solamente su tutti i conti con saldo positivo;
- I conti in rosso, ossia tutti i conti con un saldo negativo.

La soluzione di tale esercizio è disponibile all'interno della cartella *es_5*.