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

Sapienza Università di Roma

17 Maggio 2023

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:

Domanda	1	2	3	4	5	6	Totale
Punteggio Totale	10	4	3	5	12	8	32
Punteggio Ottenuto							

1. [10 punti] Per ogni domanda, indicare con una X la risposta desiderata. Si ricorda che ogni domanda ha al più una risposta corretta. L'assegnazione dei punti alle risposte è la seguente. Verranno attribuiti 2 punti per ogni risposta esatta; -0.4 punti per ogni risposta errata; 0 punti per ogni risposta omessa. Al fine del superamento della soglia è necesssario totalizzare un punteggio di almeno 5 punti.
   (a) Un ciclo for è un costrutto di iterazione ( ) Indeterminato (X) Determinato ( ) Ciclico
   (b) Una costante in Java è indicata dal modificatore ( ) static ( ) abstract (X) final
   (c) I metodi di un'interfaccia sono ( ) protected (X) public ( ) private
   (d) Il contenuto di due stringhe è confrontabile tramite il metodo ( ) compareTo (X) equals ( ) corresponds
   (e) Un metodo static può accessibile tramite (X) la sua classe ( ) il suo oggetto ( ) l'uso di alcun modificatore di visibilità
2. [4 punti] Qual è la differenza tra overloading e overriding? Fornire un esempio minimale, scritto in Java, che descriva quanto richiesto.

   Overloading e Overriding sono concetti legati al polimorfismo il quale, nella programmazione ad oggetti, si riferisce alla capacità di una classe A di assumere i valori di un qualunque tipo descritto dalla classe B, con B sottoclasse di A.

Si parla di overloading quando sono presenti più metodi all'interno di una classe o di una sua superclasse con lo stesso nome, tipo di ritorno ma con tipo e/o numero di parametri diverso. Si parla invece di overriding quando un metodo della classe derivata B sovrascrive il metodo della classe A mantenendo quindi la stessa firma ma cambiando il corpo della funzione.

Un esempio minimale in Java di quanto spiegato è fornito nella cartella Es_1.

3. 3 punti 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 = 12; i <= 7; i++){...}`
   (b) `for(int i = 0; i < 10; i++){...}`
   (c) `for(int i = -10; i >= 0; i++){...}`
   (d) `for(int i = -2; i >= 0; i++){...}`
   (e) `for(int i = -8; i <= 3; i = i + 4){...}`
   (f)
   ```
   for (int k = 0; k < 20; k+=2) {
   if (k + 3 == 1) System.out.println(k + " ");
   }
   ```

4. 5 punti Spiegare le principali differenze tra ArrayList e Set in Java.

   ArrayList implementa l'interfaccia List appartenente a sua volta all'interfaccia Collection di Java. Un ArrayList è una collezione di oggetti memorizzati sequenzialmente secondo una politica FIFO. Esso può contenere valori il cui tipo può essere primitivo o meno, ammettendo duplicati. La dimensione dell'ArrayList è dinamica e l'accesso ai dati è randomico (posizionale).

   I Set, appartenenti anch'essi all'interfaccia Collection, rappresentano un insieme di valori senza duplicati. Tale interfaccia Set è a sua volta implementata da HashSet e TreeSet: la prima utilizza una tabella hash, ofrendo quindi un tempo di esecuzione costante per le operazioni di *add, remove, size, contains.* TreeSet, al contrario, utilizza un albero interno per le operazioni di memorizzazione offrendo quindi la possibilità di avere un ordinamento. HashSet permette la presenza di un valore nullo, mentre TreeSet no.

5. 12 punti 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 utente 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. cliente

6. 8 punti Un supermercato ha accesso ad un database contenente tutte le informazioni sui propri prodotti. Tali informazioni sono rappresentate in formato tabellare, in cui ciascuna riga contiene i seguenti campi separati da uno spazio:

   codiceProdotto1 NomeProdotto1 Quantità1 PrezzoPerUnità1
   codiceProdotto2 NomeProdotto2 Quantità2 PrezzoPerUnità2
   ...

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 prodotto più costoso;
- Il prezzo medio per ciascun prodotto, calcolato come il rapporto tra prezzo per unità e la quantità disponibile. Gestire gli eventuali casi speciali
- I nomi dei prodotti esauriti, ossia con una quantità pari a zero

1) (A) INDETERMINATO
(B) FINAL
(C) PUBLIC
(D) EQUALS
(E) LA SUA CLASSE

3) $a=0 \quad b=\frac{10-0}{1}=10 \quad c=0 \quad d=0 \quad e=\frac{3+8+1}{4}=3$

$f=\frac{20-0}{2}=10$

2) OVERLOADING E OVERRIDING SONO 2 CONCETTI FONDAMENTALI DI POLIMORFISMO.

PARLIAMO DI OVERLOADING QUANDO VENGONO CREATI PIÚ COSTRUTTORI CON LO STESSO NOME MA CHE ACCETTANO ARGOMENTI DIVERSI. DI SEGUITO UN ESEMPIO:

```
public sommaNumeri(int numero1, int numero2){
    numero1 = this.numero1;
    numero2 = this.numero2;
    somma = numero1+numero2;
}

public sommaNumeri(double numero1,double numero2){
    numero1 = this.numero1;
    numero2 = this.numero2;
    somma = numero1+numero2;
}
```

QUESTO CODICE PERMETTE DI ACCETTARE SIA DEGLI INT CHE DEI DOUBLE, SENZA CAUSARE UN ERRORE

PARLIAMO DI OVERRIDING QUANDO ABBIAMO UNA CLASSE FIGLIA CHE, EREDITANDO UN METODO DA UNA CLASSE MADRE, LO "SOVRASCRIVE", UTILIZZANDO UN COMANDO PIÚ SPECIFICO PER QUELLA:

```
public class Animale{
  public void faiVerso(){
  System.out.println("Verso di un animale");
  }
}
public class Cane extends Animale{
  @Override
    public void faiVerso(){
      System.out.println("Bau bau!");
    }
}
```

IN QUESTO CODICE, SE SI CREA UN OGGETTO CANE:
`Animale animale = new Cane()` E SI INVOCA IL METODO FAIVERSO, IL VERSO NON SARÀ GENERICO MA QUELLO SPECIFICO DELLA CLASSE

4) LA PRIMA DIFFERENZA SOSTANZIALE È CHE GLI ARRAYLIST DERIVANO DALL'INTERFACCIA "LIST", MENTRE I SET DALL'OMONIMA INTERFACCIA. UN ELEMENTO PUÓ ESSERE INSERITO PIÚ VOLTE IN UN ARRAYLIST, MENTRE I SET MEMORIZZANO OGNI ELEMENTO UNA SOLA VOLTA. UN ELEMENTO IN UN ARRAYLIST PUÓ ESSERE CERCATO MOLTO VELOCEMENTE TRAMITE IL SUO INDICE, MENTRE I SET NON SONO ORDINATI QUINDI BISOGNA ITERARE SU TUTTO QUANTO UTILIZZANDO IL METODO CONTAINS().

9)
```
public class Customer{
    private double balance;
    public Customer(){
        balance = 0;
    }
public boolean discountReached(){
  return (balance >= 100);
}
public void makePurchase(double amount){
   if(discountReached()){
     balance = amount - 10;
   }
   else{
     balance += amount;
   }
}
```

```
public class CustomerTester{
   public static void main(String[] args){
      Customer customer = new Customer();
      customer.makePurchase(100);
      customer.discountReached();
      customer.makePurchase(10$);
      customer.discountReached()
      customer.makePurchase(10);
      customer.discountReached();
   }
}
```

6)
```
public class Product {
  private String code;
  private String name;
  private int number;
  private float priceU;
  public Product (String code, String name, int number, float priceU){
    this.code = code;
    this.name = name;
    this.number=number;
    this.priceU=priceU;
  }
  public String code(){
    return code;
  }
  public String name(){
    return name;
  }
  public int number(){
    return number;
  }
  public float priceU(){
    return priceU;
  }
  public float midPrice (){
    if (priceU==0){
      return Float.NaN;
    }
    else{
      return priceU/number;
    }
  }
  public boolean empty(){
    return (number==0);
  }
  public static Product convert(String line){
    String[] prod = line.split(" ");
    return new Product (prod[0], prod[1], Integer.parseInt(prod[2]), Float.parseFloat(prod[3]));
  }
}
```

NB. product.name() è corrispondente "get"
product.number()

```
public class Store {
  private ArrayList<Product> products;
  public Store (ArrayList <Product> products){
    this.products = products;
  }
  public void printMid(){
    for (Product product: products){
      System.out.println("Il prezzo medio dell'articolo" + product.nome()+" é " + product.midPrice());
    }
  }

  public void printBoolean(){
    for(Product product: products){
      if(product.empty()){
        System.out.println("L'articolo" + product.name + " é esaurito");
      }
    }
  }
  public Product searchMax(){
    Product max = null;
    for(int i=0; i < products.size(); i++){
      if(i==0){
        max = products.get(i);
      }
      if(products.get(i).priceU() > max.priceU()){
        max = products.get(i);
      }
    }
    return max;
}
```

```
public class Main{
  public static void main(String[] args){
    Scanner reader;
    try{
        reader = new Scanner(new File("elements.txt"));
      } catch (FileNotFoundException exc){
        System.out.println("file not found");
        return;
      }
    ArrayList<Product> products = new ArrayList<>();
    String line;
    while (reader.hasNextLine()){
      line = reader.nextLine();
      products.add(Product.convert(line));
    }
    Store store = new Store(products)
    Product max = store.searchMax();
    System.out.println(max.codeL()+" "+max.name()+" "+max.number()+" "+max.priceU());
    store.printMid();
    store.printBoolean();
  }
}
```