L10 - Simulazione della verifica (svolta)

Java · traccia completa con soluzione passo-passo · livello esame

Materiale didattico di Moreno Sardella · morenosardella.com

Come funziona. Questa e' una traccia identica per struttura alla verifica di recupero (interfaccia + superclasse + sottoclassi polimorfiche + classe con validazione + scrittura CSV con separatore ';' + lettura file di input). Prima leggila tutta, stima i punti, poi confronta con la soluzione completa in fondo. Punteggio totale: 10.

Esercizi

Esercizio 1 — Interfaccia e superclasse [4.0 pt]

EREDITARIETA' + POLIMORFISMO

Definisci l'interfaccia **Pagabile** con **getImporto():double**, **getTipo():String**, **getId():String**. Crea la superclasse astratta **Impiegato** che la implementa (nome, cognome, codiceFiscale) e tre sottoclassi con **getImporto()** diverso: **ImpiegatoSalariato** (stipendio fisso), **ImpiegatoOrario** (paga * ore), **ImpiegatoAProvvigione** (base + vendite * perc / 100).

```
interface Pagabile {
    double getImporto();
    String getTipo();
    String getId();
}
```

Esercizio 2 — Classe con validazione [2.5 pt]

COSTRUTTORE ROBUSTO

Crea **Fattura** che implementa Pagabile (numero, descrizione, quantita, prezzoPerUnita). Nel costruttore valida che **quantita > 0**: se non lo e', stampa un errore e segna la fattura come non valida. getImporto() = prezzoPerUnita * quantita.

```
Fattura(String num, String desc, int q, double prezzo) {
    if (q &lt;= 0) { /* errore + valida=false */ }
}
```

Esercizio 3 — Gestore e scrittura CSV [2.5 pt]

POLIMORFISMO + FILE

Crea **GestorePagamenti** che riceve un array **Pagabile[]** e con **scriviCSV(String nomeFile)** scrive un CSV con intestazione *TipoEntita;Identificativo;ImportoDovuto* e una riga per elemento (importo con due decimali). Sfrutta il polimorfismo: ogni elemento usa il suo getImporto().

```
out.println(p.getTipo()+";"+p.getId()+";"
            + String.format("%.2f", p.getImporto()));
```

Esercizio 4 — Lettura del file di input [1.0 pt]

SCANNER + SPLIT

Nel main, leggi il file **fatture_input.txt** (righe *numero;descrizione;quantita;prezzo*), costruisci le Fattura, aggiungi solo quelle valide alla lista, poi genera **pagamenti.csv** con il gestore.

```
Scanner in = new Scanner(new File("fatture_input.txt"));
// split(";") + costruzione oggetti
```

Soluzioni

Confronta solo dopo aver provato da solo. Esistono soluzioni equivalenti: conta la logica, non la copia esatta.

Soluzione 1

Interfaccia + superclasse astratta + 3 sottoclassi:

```
interface Pagabile {
    double getImporto(); String getTipo(); String getId();
}
abstract class Impiegato implements Pagabile {
    protected String nome, cognome, codiceFiscale;
    Impiegato(String n,String c,String cf){nome=n;cognome=c;codiceFiscale=cf;}
    public String getId(){ return nome + " " + cognome; }
}
class ImpiegatoSalariato extends Impiegato {
    private double stipendio;
    ImpiegatoSalariato(String n,String c,String cf,double s){super(n,c,cf);stipendio=s;}
    public double getImporto(){ return stipendio; }
    public String getTipo(){ return "ImpiegatoSalariato"; }
}
class ImpiegatoOrario extends Impiegato {
    private double paga; private int ore;
    ImpiegatoOrario(String n,String c,String cf,double p,int o){super(n,c,cf);paga=p;ore=o;}
    public double getImporto(){ return paga * ore; }
    public String getTipo(){ return "ImpiegatoOrario"; }
}
class ImpiegatoAProvvigione extends Impiegato {
    private double base, vendite, perc;
    ImpiegatoAProvvigione(String n,String c,String cf,double b,double v,double p){
        super(n,c,cf); base=b; vendite=v; perc=p;
    }
    public double getImporto(){ return base + vendite * perc / 100; }
    public String getTipo(){ return "ImpiegatoAProvvigione"; }
}
```

Soluzione 2

Fattura con validazione nel costruttore:

```
class Fattura implements Pagabile {
    private String numero, descrizione;
    private int quantita; private double prezzoPerUnita;
    private boolean valida = true;
    Fattura(String num,String desc,int q,double prezzo){
        if (q &lt;= 0){ System.out.println("Errore: quantita non valida "+num); valida=false; return; }
        numero=num; descrizione=desc; quantita=q; prezzoPerUnita=prezzo;
    }
    public boolean isValida(){ return valida; }
    public double getImporto(){ return prezzoPerUnita * quantita; }
    public String getTipo(){ return "Fattura"; }
    public String getId(){ return numero + " - " + descrizione; }
}
```

Soluzione 3

Gestore con scrittura CSV polimorfica:

```
import java.io.*;
class GestorePagamenti {
    private Pagabile[] elementi;
    GestorePagamenti(Pagabile[] e){ elementi = e; }
    public void scriviCSV(String nomeFile) throws IOException {
        PrintWriter out = new PrintWriter(nomeFile);
        out.println("TipoEntita;Identificativo;ImportoDovuto");
        for (Pagabile p : elementi)
            out.println(p.getTipo()+";"+p.getId()+";"+String.format("%.2f", p.getImporto()));
        out.close();
    }
}
```

Soluzione 4

Main con lettura input e generazione CSV:

```
import java.io.*; import java.util.*;
public class Main {
  public static void main(String[] a) throws IOException {
    ArrayList&lt;Pagabile&gt; lista = new ArrayList&lt;&gt;();
    lista.add(new ImpiegatoSalariato("Mario","Rossi","RSSMRA",2500));
    lista.add(new ImpiegatoOrario("Luigi","Verdi","VRDLGU",16,80));
    Scanner in = new Scanner(new File("fatture_input.txt"));
    while (in.hasNextLine()) {
      String[] c = in.nextLine().split(";");
      Fattura f = new Fattura(c[0],c[1],Integer.parseInt(c[2]),Double.parseDouble(c[3]));
      if (f.isValida()) lista.add(f);
    }
    in.close();
    Pagabile[] arr = lista.toArray(new Pagabile[0]);
    new GestorePagamenti(arr).scriviCSV("pagamenti.csv");
  }
}
```