



Ereditarietà

Laboratorio



Esercizio 1

- Scrivere la gerarchia di classi BankAccount, SavingsAccount, CheckingAccount
- Migliorare il metodo addInterest della classe SavingsAccount in maniera tale da calcolare gli interessi sul minimo saldo dall'ultima invocazione del metodo addInterest. *Suggerimento*: Occorre modificare anche il metodo preleva e bisogna usare una variabile di istanza per ricordare il saldo minimo.



Esercizio 2

- Implementare una superclasse `Persona`. Realizzare due classi `Studente` e `Insegnante` che ereditano da `Persona`.
- Una persona ha un nome ed un anno di nascita.
- Uno studente ha una specializzazione, ed un insegnante ha uno stipendio.
- Scrivere i metodi `toString` per tutte le classi.
- Fornire un programma di test per queste classi.



Esercizio 3 –

- Considerare un'azienda con lavoratori retribuiti a ore e lavoratori retribuiti con un fisso settimanale.
- Per i lavoratori a ore, la paga settimanale è calcolata moltiplicando la paga oraria per il numero di ore effettivamente lavorate fino ad un massimo di 40 ore, se le ore lavorate eccedono le 40 ore, allora la parte di ore eccedente 40 viene retribuita una volta e mezzo la paga oraria ordinaria.
- I lavoratori retribuiti con un fisso ricevono la paga di 40 ore indipendentemente dalle ore effettivamente lavorate.
- Progettare le classi necessarie per il calcolo delle paghe settimanali
- Scrivere un programma di test che crei un elenco di 20 lavoratori (determinando casualmente il tipo di retribuzione), permetta di inserire per ogni lavoratore il numero di ore lavorate e dia in output per ogni lavoratore il nome e la sua paga settimanale
- Riutilizzare il codice dove possibile



Esercizio 4

- Considerate l'esercizio in cui bisognava implementare una `Libreria`.
- Migliorate la soluzione facendo in modo che la libreria contenga anche `Video` (titolo; autore; produttore; numero copie disponibili)
- Estendere ulteriormente la soluzione prevedendo anche `AudioLibri` che oltre alle informazioni dei `Libri` possiedono una durata ed un formato audio.



Esercizio 5

- Un episodio di una serie televisiva è caratterizzato da un titolo, una trama e da un riferimento al prossimo episodio della serie. Scrivete una classe **Episode** con opportune variabili d'istanza pubbliche per memorizzare i dati descritti sopra. Dotare la classe **Episode** dei metodi **toString** e **equals**, assumendo che due episodi sono uguali se hanno lo stesso titolo (a meno di maiuscole e minuscole) e se anche i loro episodi successivi sono uguali.
- Una serie televisiva è caratterizzata da un titolo e un episodio iniziale. Scrivere una classe **Serial** con opportune variabili d'istanza private per memorizzare i dati descritti sopra. Dotare la classe **Serial** dei metodi **toString** (che deve stampare le informazioni su tutti gli episodi della serie) e **equals**, assumendo che due serie sono uguali se hanno lo stesso titolo (a meno di maiuscole e minuscole) e se anche i loro episodi iniziali sono uguali (secondo il criterio di confronto definito sugli episodi).
- Scrivere un programma **TestSerial** che crea alcuni oggetti di tipo **Episode** (collegandoli opportunamente) e di tipo **Serial** e verifica che i metodi **toString** e **equals** si comportino come atteso.



Esercizio

- Implementare (riutilizzando il codice in comune) e collaudare le classi:
 - **Personale**
 - ha un nome e una retribuzione
 - **Impiegato**
 - ha un nome, una retribuzione, un incarico
 - **Manager**
 - ha un nome, una retribuzione, un incarico, un'area di competenza
- Per ogni classe implementare il metodo **toString**



Esercizio

- Definire una superclasse astratta **FigureGeometriche** e tre sottoclassi
 - Cerchio, Rettangolo e TriangoloIsoscele
- Per ciascuna classe, definire
 - le variabili di istanza per le dimensioni che caratterizzano il tipo di figura,
 - il costruttore per la creazione di un'istanza di ciascuna figura date le dimensioni,
 - due metodi **getArea()** e **getPerimetro()**, che restituiscono rispettivamente l'area e il perimetro della figura



Esercizio

- Scrivere un programma di Test per le classi create nell'esercizio precedente:
 - Creare un array di figure geometriche, contenente cinque oggetti per ognuna delle classi create, assegnando a ciascuna dimensione un valore casuale;
 - Stampare la somma delle aree e la somma dei perimetri di tutte le figure presenti nell'array



Esercizio

- La classe **Contenitore** (del pacchetto `it.unisa.lp2.contenitore`) è caratterizzata da **Nome**, **Codice**, **Prezzo**, **Capienza**. Corredarla di un insieme di metodi di accesso e metodi modificatori adeguati. Sovrascrivere i metodi **toString**, **equals** e **clone**.
- Le classi che implementano i concetti di contenitori cubici e contenitori cilindrici. Rispetto ai contenitori in generale, i contenitori cubici sono caratterizzati dall'ampiezza del lato e quelli cilindrici dal raggio della base e l'altezza. Sovrascrivere i metodi **toString**, **equals** e **clone**.
- Implementare in Java la classe **ElencoContenitori** come collezione di contenitori. Corredarla dei metodi necessari.
- Scrivere le classi Java per poter calcolare il minimo e il massimo dei contenitori di un elenco rispetto al **Nome** e rispetto al **Codice**. L'implementazione scelta deve favorire il riutilizzo del codice e la versatilità rispetto ad altri criteri di ordinamento. Ad esempio, se in futuro si vuole calcolare il minimo e il massimo rispetto al **Prezzo**, vogliamo poter aggiungere soltanto una nuova classe che implementi la relazione d'ordine per il prezzo senza modificare il resto del codice.



Esercizio GregorianCalendar

- Implementare in Java la classe **Prodotto** che modella le tipologie di prodotti venduti da un supermercato. Ogni prodotto è caratterizzato da codice, descrizione, marca, prezzo, quantità e inOfferta (quest'ultima è una variabile booleana indicante se il prodotto è in offerta). Corredare la classe con i metodi
- **mettiInOfferta(double p, GregorianCalendar g)** che definisce un'offerta di percentuale p con data di scadenza dell'offerta g. Tale metodo setta a true lo stato della variabile inOfferta ed applica la percentuale di sconto p sul prodotto.
- **controllaOfferta()** che setta a false lo stato della variabile inOfferta e ristabilisce il prezzo del prodotto se la data di scadenza dell'offerta è inferiore alla data corrente.
Nota: per ottenere l'ora corrente utilizzate la classe GregorianCalendar.
- Definire inoltre due sottoclassi
 - **Alimentare** caratterizzata da una data di scadenza e da un peso, e che fornisce i seguenti metodi
 - **boolean èScaduto()** che verifica se il prodotto è scaduto.
 - **boolean acquista(int p)** che simula l'acquisto di p grammi di prodotto. Se la variabile p non eccede la quantità disponibile del prodotto allora il prodotto può essere acquistato. Restituisce true in caso di acquisto.
 - **Elettronico** caratterizzata dalle variabili garanziaBase (numero di anni per cui il prodotto è garantito), prezzoAnnualeGaranzia (costo annuale per estendere la garanzia).
 - **double calcolaCostoGaranzia(int a)** che calcola il costo della garanzia per a anni, cioè $a \times \text{prezzoAnnualeGaranzia}$.
 - **boolean acquista(int q)** che simula l'acquisto di q prodotti. Se la variabile q non eccede la quantità presente nel magazzino allora il prodotto può essere acquistato. Restituisce true in caso di acquisto.



Continuazione

- Scrivere la classe **Supermercato** che modella una collezione di prodotti e fornisce i seguenti metodi:
 - **void aggiungiProdotto(Prodotto p)** che inserisce un prodotto in modo ordinato rispetto al codice. *L'implementazione scelta deve favorire il riutilizzo del codice e la versatilità rispetto ad altri criteri di ordinamento. Ad esempio, se in futuro si vuole ordinare rispetto al prezzo, vogliamo poter aggiungere soltanto una nuova classe che implementi la relazione d'ordine per il prezzo senza modificare il resto del codice.*
 - **String daiTipoProdotto(int i)** che restituisce il tipo dell'i-esimo prodotto (alimentare o elettronico).
 - **Prodotto getMinimo()**, che restituisce il prodotto che ha il costo minimo.
 - **Prodotto getMassimo()**, che restituisce il prodotto che ha il costo minimo.
 - **ArrayList<Prodotto> cerca(String marca)**, che cerca e restituisce tutti i prodotti di una certa marca.



Azienda

- Un'azienda ha deciso di informatizzare la contabilità dei propri dipendenti. Si è creato un file in cui, per ogni impiegato che lavora presso l'azienda, sono stati inseriti i seguenti elementi:
 - cognome; nome; codice fiscale; stipendio.
- L'azienda ha la necessità di effettuare le seguenti operazioni:
 - cercare tutti i dipendenti che guadagnano più di una certa somma
 - cercare tutti i dipendenti che hanno un determinato nome o cognome
 - cercare i dipendenti che hanno lo stipendio minimo
 - aumentare lo stipendio di tutti dipendenti di valore percentuale (ed aggiornando il file)
- Creare inoltre una classe main per testare le classi ed i metodi definiti. In particolare occorre
 - leggere da un file contenente i dati dei 6 dipendenti (mostrati di seguito)
 - visualizzare i dipendenti che guadagnano più di 1500 euro
 - visualizzare i dipendenti che si chiamano Gianni (di nome o cognome);
 - visualizzare il dipendente con lo stipendio minimo
 - aumentare lo stipendio del 5 percento.
- Rossi
- Mario
- RSSMRO45T13T404K
- 1450
- Bianchi Marco
- BNCMRC34G15H723T
- 1250