Fondamenti di Architetture e Programmazione
III Compitino

Esercizi di preparazione

Cognome **Pippi** Nome **Pippo**
Matricola **696969**

- Usate il retro dei fogli per scrivere lo svolgimento degli esercizi di programmazione.
- Nei seguenti quesiti, quando vi è richiesto di scrivere un programma, potete limitarvi al *corpo* del metodo main, assumendo se necessario che in e out siano due variabili di classe ConsoleInputManager e ConsoleOutputManager (rispettivamente), già dichiarate e inizializzate.

1. Scrivete un programma che legga una sequenza di esattamente 10 stringhe, e stampi solo quelle con lunghezza pari.

   Ecco un esempio di esecuzione (le parti in grassetto sono state inserite dall'utente):

   `Stringa 1:` **mamma**
   `Stringa 2:` **pipo**
   `Stringa 3:` **xyp231**
   `...`

   ```
   pipo
   xyp231
   ...
   ```

(Svolgimento sul retro)

```java
String[] s = new String[10];
int i;
for (i=0;i<10;i++)
 s[i]=in.readLine("Stringa " + (i+1) + ": ");
for (i=0;i<10;i++)
 if (s[i].length()%2==0)
  out.println(s[i]);
```

2. Considerate la classe Corso i cui oggetti corrispondono a *corsi universitari*; questa classe ha un costruttore pubblico

   - public Corso(String nome, String docente, int cfu)

   che costruisce un corso universitario denominato nome, il cui docente titolare ha cognome docente e con numero di crediti cfu. La classe possiede i metodi

   - public String getNome()
   - public String getDocente()
   - public int getCfu()

   che forniscono, rispettivamente, il nome del corso, il cognome del docente titolare ed il numero di crediti.

   Scrivete un programma che operi come segue:

   (a) chieda all'utente di inserire un numero intero, diciamo N;
   (b) dichiari un array di N corsi e lo riempia con corsi inseriti dall'utente;
   (c) stampi i soli corsi tenuti da "Rossi" con numero di crediti maggiore o uguale a 6 (la classe Corso dispone di un opportuno metodo toString()).

   Ecco un esempio di esecuzione (le parti in grassetto sono state inserite dall'utente):

   Inserisci un intero: **4**

   Nome: **Informatica Generale**
   Docente: **Rossi**
   Cfu: **12**

   Nome: **Matematica Discreta**
   Docente: **Verdi**
   Cfu: **6**

   Nome: **Interazione Uomo-Macchina**
   Docente: **Rossi**
   Cfu: **6**

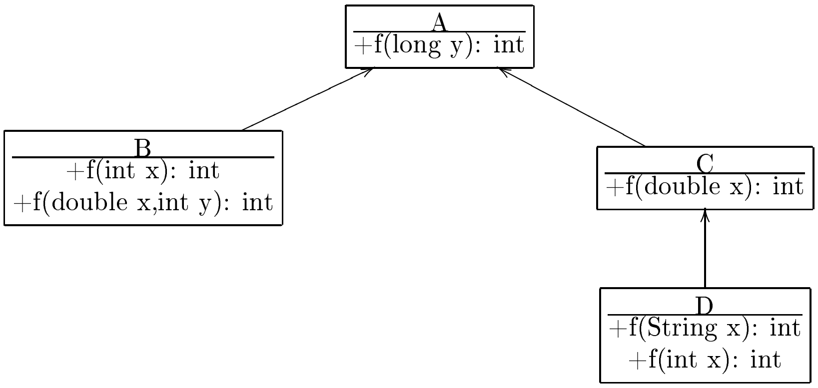
   Nome: **Metodi per il Trattamento dell'Informazione**
   Docente: **Rossi**
   Cfu: **5**

   Corso: Informatica Generale; Tenuto da: Rossi; Cfu: 12
   Corso: Interazione Uomo-Macchina; Tenuto da: Rossi; Cfu: 6

(Svolgimento sul retro)

```java
int n=in.readInt("Inserisci un intero");
Corso[] c = new Corso[n];
for (int i=0; i<n; i++) {
 String nome = in.readLine("Nome: ");
 String docente = in.readLine("Docente: ");
 int cfu = in.readInt("Cfu: ");
 c[i] = new Corso(nome, docente, cfu);
}
for (int i=0; i<n; i++)
 if (c[i].getDocente().equals("Rossi") && c[i].getCfu()>=6)
  out.println(c[i]);
```

3. Considerate la seguente gerarchia dei tipi (rappresentata mediante un diagramma UML):



e ipotizzate che tutte le classi abbiano un costruttore pubblico senza argomenti. Assumete le seguenti definizioni e inizializzazioni:

```
A a1, a2, a3, a4;
B b;
C c1, c2;
D d;

a1 = new A(); a2 = new B(); a3 = new C(); a4 = new D();
b = new B();
c1 = new C(); c2 = new D();
D d = new D();
```

Per ciascuno dei seguenti assegnamenti, dite se l'assegnamento è consentito, oppure se richiede (per poter essere compilato) un operatore di cast nel secondo membro (e in tal caso indicate l'assegnamento corretto), oppure se l'assegnamento non è consentito nemmeno con un cast (nel senso che anche con un cast produrrebbe un errore di compilazione o un'eccezione in esecuzione):

- b=a2: **b=(B)a2**
- a1=d: **Ok**
- a1=c: **Ok**
- a2=b: **Ok**
- a1=c1: **Ok**
- c2=c1: **Ok**
- c1=a2: **Non compila**

4. Facendo riferimento al precedente diagramma UML e alle variabili definite e inizializzate come sopra, considerate le seguenti invocazioni di metodo. Per ciascuna di esse dovete dire se è consentita e in caso affermativo dovete indicare il nome della classe che contiene il metodo che verrà effettivamente eseguito:

   - `d.f("pippo")`: **D: f(String)**
   - `d.f(3,4)`: **Non compila**
   - `a1.f(3,4)`: **Non compila**
   - `a3.f(5)`: **A: f(long)**
   - `a4.f("pippo")`: **Non compila**
   - `b.f(3)`: **B: f(int)**
   - `d.f(3.4)`: **C: f(double)**
   - `a2.f(3,4)`: **Non compila**

5. La classe Integer contiene un metodo statico

```
public static int parseInt( String x, int radix )
```

che data una stringa x la interpreta come un numero intero in base radix e interpreta l'intero corrispondente. Avete una variabile stringa s che contiene la rappresentazione di un numero intero in base 7, e volete calcolarne il doppio mettendo il risultato (un numero) nella variabile intera z; scrivete l'istruzione di assegnamento a z:

```
z=2*Integer.parseInt(s,7);
```

6. Considerate la seguente funzione definita ricorsivamente sugli interi:

$$f(x) = \begin{cases} x+1 & \text{se } x \text{ è multiplo di 10} \\ x - f(x+1) & \text{altrimenti} \end{cases}$$

Scrivete un metodo statico con intestazione

```
public static int f( int x )
```

per il calcolo di f.

```
public static int f(int x) {
 if (x%10==0) return x+1;
 else return x-f(x+1);
}
```

7. Scrivete una classe di nome Prodotto le cui istanze rappresentano i prodotti in vendita in un supermercato. Ogni prodotto è caratterizzato da un nome (p.es. "Pannoloni Morbidoni Fluffy"), un prezzo unitario espresso da un double (p.es. 3.12), e un numero intero che rappresenta quantità disponibile (p.es. 1500).

   La classe deve contenere i seguenti metodi e costruttori:

   - Prodotto(String nome, **double** prezzo, **int** disp): crea un prodotto con dato nome, prezzo unitario e quantità disponibile;
   - Prodotto(String nome, **double** prezzo): come sopra, ma assume che la quantità disponibile sia zero;
   - **void** acquista(**int** qta): aumenta la quantità disponibile del valore qta;
   - **void** vendi(**int** qta) **throws** IllegalStateException: riduce la quantità disponibile del valore qta, e solleva un'eccezione se la quantità disponibile è inferiore (l'eccezione IllegalStateException è un'eccezione già esistente, con gli usuali costruttori);
   - **double** prezzoTotale(**int** qta): restituisce il prezzo totale relativo alla quantità qta;
   - **boolean** scorteFinite(): restituisce true se le scorte di questa merce sono terminate (cioè, se la quantità disponibile è zero);
   - String toString(): restituisce una stringa descrittiva.

(Svolgimento sulla facciata seguente)

```
class Prodotto {
 private String nome;
 private double prezzo;
 int disp;
 public Prodotto(String nome, double prezzo, int disp) {
  this.nome=nome;
  this.prezzo=prezzo;
  this.disp=disp;
 }
 public Prodotto(String nome,double prezzo) {
  this(nome,prezzo,0);
 }
 public void acquista(int qta) {
  disp+=qta;
 }
 public void vendi(int qta) throws IllegalStateException {
  if (disp<qta) throw new IllegalStateException();
  disp-=qta;
 }
 public double prezzoTotale(int qta) {
  return prezzo*qta;
 }
 public boolean scorteFinite() {
  return disp==0;
 }
 public String toString() {
  return nome + ", EUR " + prezzo;
 }
}
```

8. Definite una classe di nome ProdottoScontato che estende Prodotto e i cui oggetti corrispondono a dei prodotti con sconto. Questa classe deve avere:

   - un costruttore pubblico

     ```
     ProdottoScontato(String nome, double prezzo, int disp, double sconto)
     ```

     che specifica il nome del prodotto, il suo prezzo unitario, la quantità disponibile e la percentuale di sconto (p.es., se sconto vale 0.3 significa che il prodotto viene venduto con il 30% di sconto);

   - un metodo

     ```
     public boolean scontoAlto()
     ```

     che restituisce true se e solo se il prodotto ha uno sconto maggiore del 50%;

   - il metodo ereditato prezzoTotale(**int** qta) deve restituire il prezzo al netto dello sconto.

```
class ProdottoScontato extends Prodotto {
 private double sconto;
 public ProdottoScontato(String nome, double prezzo, int disp, double sconto) {
  super(nome, prezzo, disp);
  this.sconto=sconto;
 }
 public double prezzoTotale(int qta) {
  return super.prezzoTotale(qta)*(1-sconto);
 }
 public boolean scontoAlto() {
  return sconto>.5;
 }
}
```