Laboratorio di Programmazione
Edizione 1 - Turni A, B, C

ESAME del 4 Aprile 2016

Avvertenze

- Nello svolgimento dell'elaborato è possibile usare qualunque classe delle librerie standard di Java.
- Non è invece ammesso l'uso delle classi del package `prog` allegato al libro di testo del Prof. Pighizzini e impiegato nella prima parte del corso.
- Si consiglia CALDAMENTE l'utilizzo dello script "checker.sh" per compilare ed effettuare una prima valutazione del proprio elaborato. Si consiglia anche di leggere il sorgente dei `Test_*.java` per capire cosa devono offrire le classi da sviluppare.
- Ricordarsi, quando si programma: *Repetita NON iuvant* o DRY (*Don't Repeat Yourself*).

ESERCIZIO FILTRO

INIZIARE PRIMA CON QUESTO, se non si è in grado di portare a termine questo esercizio NON PROSEGUIRE.
La correttezza di questo esercizio è condizione NECESSARIA perché i docenti correggano il resto del compito.
Realizzare una classe `Digits`, dotata del solo `main`, che letta da linea di comando una sequenza di numeri interi positivi visualizzi per ciascuno numero la somma delle cifre di cui è composto. Non è necessario effettuare alcun controllo sull'input.
Ecco un **esempio** d'uso:

```
$ java Digits 33 15 19 120
33 : somma 6
15 : somma 6
19 : somma 10
27 : somma 9
120 : somma 3
```

Tema d'esame

Lo scopo dell'esercizio è realizzare un modello minimale di acquario per pesci, sia d'acqua dolce che salata. Le diverse classi java permettono di descrivere entità e concetti quali acquario, pesce d'acqua dolce, d'acqua salata, volume dell'acquario, densità di popolazione, etc.
Le **classi** da realizzare sono le seguenti:

1. `Acquario`: rappresenta un generico acquario, classe **astratta**
2. `AcquarioDiMare`: sottoclasse concreta di Acquario, contiene solo i pesci di mare
3. `AcquarioDolce`: sottoclasse concreta di Acquario, contiene solo i pesci d'acqua dolce
4. `Pesce`: rappresenta un generico pesce, classe **astratta**
5. `PesceDacquaDolce`: sottoclasse concreta di Pesce
6. `PesceDiMare`: sottoclasse concreta di Pesce
7. `Main`: main di test (in aggiunta ai test automatici)

Specifica delle classi

Le classi (**pubbliche**!) dovranno esporre almeno i metodi e costruttori **pubblici** indicati, più eventuali altri metodi e costruttori se ritenuti opportuni. Gli attributi (campi) delle classi devono essere dichiarati **privati**. Per leggere e modificarne i valori, creare opportunamente, e solo dove indicato, i metodi di accesso (`set` e `get`). Se si usano classi che utilizzano tipi generici, si suggerisce di utilizzarne le versioni opportunamente istanziate (es. `ArrayList<String>` invece di `ArrayList`). Ogni classe deve avere il metodo `toString` che descriva lo stato delle istanze.

1 Acquario

Classe **astratta**, contiene istanze di pesce, ha un volume, è un parallelepipedo. Deve implementare almeno i seguenti metodi:

- `public Acquario(int l,int a,int p)` costruttore che accetta larghezza, altezza e profondità (in cm)
- `public boolean aggiungi(Pesce p)` aggiunge un pesce all'acquario
- `public Pesce togli(int quale)` toglie un pesce dall'acquario
- `public double densita()` calcola e restituisce la densità (numero pesci per unità di volume) dell'acquario
- `public int getVolume()` calcola e restituisce il volume dell'acquario, in cm cubici
- `public int quanti()` calcola e restituisce il numero di pesci presenti
- `public int quantiMuoiono()` calcola e restituisce quanti pesci morirebbero (si veda la classe Pesce) data l'attuale densità
- `public String toString()` ritorna una rappresentazione dello stato dell'oggetto

2 AcquarioDolce

Sottoclasse concreta di Acquario, accetta solo pesci di acqua dolce e implementa un limite (fissato a 100) al numero di pesci che si possono mettere. Deve implementare almeno i seguenti metodi:

- fare opportuni *override* dei metodi della classe da cui questa deriva

3 AcquarioDiMare

Sottoclasse concreta di Acquario, accetta solo pesci di mare e implementa un limite (fissato a 20000 cm cubici) al volume. Deve implementare almeno i seguenti metodi:

- fare opportuni *override* dei metodi della classe da cui questa deriva. Il costruttore deve lanciare un'eccezione se il volume corrispondente ai parametri passati supera il limite.

4 Pesce

Classe **astratta**, rappresenta un generico pesce, ha un nome (String) e ha come attributo una densità massima di popolazione, sopra la quale il pesce muore per sovraffollamento. Deve implementare almeno i seguenti metodi:

- `public Pesce(String nome, float densitaMax)` costruttore che accetta i parametri "nome", "densità massima"
- `public boolean isVivo(Acquario a)` decide se il pesce è vivo o morto rispetto alla densità dell'acquario che viene passato come parametro
- `public String toString()` ritorna una rappresentazione dello stato dell'oggetto

5 PesceDacquaDolce

Sottoclasse concreta di Pesce, implementa un limite MASSIMO (fissato a 0.001 $pesci/cm^3$) alla densità massima. Deve implementare almeno i seguenti metodi:

- fare opportuni *override* dei metodi della classe da cui questa deriva. Il costruttore deve lanciare un'eccezione se la densità massima supera il limite fissato.

6 PesceDiMare

Sottoclasse concreta di Pesce, implementa un limite MINIMO (fissato a 0.00001 $pesci/cm^3$) alla densità massima. Deve implementare almeno i seguenti metodi:

- fare opportuni *override* dei metodi della classe da cui questa deriva. Il costruttore deve lanciare un'eccezione se la densità massima NON supera il limite fissato.

7 Main

Classe contenente solo il metodo `main` (correttamente dichiarato) che faccia:

- istanzi un acquario dolce e uno salato (entrambi 50cm x 50cm x 50cm)
- istanzi 10 pesci d'acqua dolce con nomi scelti arbitrariamente e densità massime casuali (tra 0.00007 e 0.00009)
- istanzi 10 pesci di mare con nomi scelti arbitrariamente e densità massime casuali (idem come sopra)
- metta i pesci istanziati IN MODO RANDOM nei due acquari, catturando eventuali errori
- stampare a video il risultato dell'inserimento (elenco dei pesci nei due acquari)
- calcolare il numero di pesci vivi e morti nei due acquari

Raccomandazioni

Affinché l'elaborato sia valutato, è richiesto che sia le classi sviluppate che i test risultino *compilabili*. A tal fine i metodi/costruttori che non saranno sviluppati dovranno comuque avere una implementazione fittizia come la seguente:

```
public <tipo> <nomeDelMetodo> (<lista parametri>) {
   throw new UnsupportedOperationException();
}
```

Si suggerisce quindi di dotare da subito le classi di tutti i metodi richiesti, implementandoli in modo fittizio, e poi di sostituire man mano le implementazioni fittizie con implementazioni che rispettino le specifiche.

Consegna

Si ricorda che le classi devono essere tutte *public* e che vanno consegnati tutti (e soli) i file *.java* prodotti. NON vanno consegnati i *.class*. NON vanno consegnati i file relativi al meccanismo di autovalutazione (*Test_*.java, AbstractTest.java, *.sh*). Per la consegna, eseguite l'upload dei SINGOLI file sorgente (NON un file archivio!) dalla pagina web: http://upload.di.unimi.it nella sessione del vostro docente.

*** ATTENZIONE!!! ***

NON VERRANNO VALUTATI GLI ELABORATI CON ERRORI DI COMPILAZIONE O LE CONSEGNE CHE NON RISPETTANO LE SPECIFICHE (ad esempio consegnare un archivio zippato è sbagliato, come anche consegnare ad un docente diverso dal proprio assegnato).
UN SINGOLO ERRORE DI COMPILAZIONE O DI PROCEDURA INVALIDA **TUTTO** L'ELABORATO.

Per ritirarsi fare l'upload di un file vuoto di nome `ritirato.txt`. Se avete caricato dei file nella sessione del docente sbagliato, caricate lì un file vuoto di nome `erratoConsegna.txt` e caricate poi i file nella sessione giusta.
