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

ESAME del 22 Febbraio 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*).

Tema d'esame

Lo scopo dell'esercizio è realizzare un modello minimale di gestione della logistica di trasporto colli. Le diverse classi java permettono di descrivere entità e concetti quali pallet, collo, dimensioni, peso, indirizzo, etc.
Le **classi** da realizzare sono le seguenti:

1. `Collo`: rappresenta un generico collo trasportabile, classe **astratta**
2. `Cilindrico`: sottoclasse concreta di Collo, rappresenta un contenitore cilindrico
3. `Parallelepipedo`: sottoclasse concreta di Collo, rappresenta un contenitore a forma di parallelepipedo
4. `Indirizzo`: rappresenta un indirizzo "valido"
5. `CAP`: rappresenta un CAP "valido"
6. `Pallet`: contenitore di colli

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 Collo

Astratta, **implementa l'interfaccia Comparable**, contiene informazioni su mittente e destinatario (di tipo Indirizzo), "espone" (attraverso un metodo) un peso e deve implementare almeno i seguenti metodi:

- `public Indirizzo getDestinatario()` restituisce il destinatario
- `public void setDestinatario(Indirizzo i)` imposta il destinatario, lancia eccezione se il parametro è nullo
- `public Indirizzo getMittente()` restituisce il mittente
- `public void setMittente(Indirizzo i)` imposta il mittente, lancia eccezione se il parametro è nullo
- `public float getPeso()` restituisce il peso, il peso va calcolato moltiplicando il volume per 0.8
- `public abstract float getVolume()` restituisce (calcolandolo opportunamente) il volume del collo
- `public boolean isSpedibile()` ritorna true se mittente e destinatario **non** sono nulli
- `public String toString()` ritorna una rappresentazione dello stato dell'oggetto

2 Cilindrico

Sottoclasse concreta di Collo, rappresenta un contenitore di forma cilindrica, ha come attributi una altezza e un raggio di base (interi) e deve implementare almeno i seguenti metodi:

- `public Cilindrico(int h, int r)` costruttore che accetta i parametri "altezza" e "raggio di base" (si consiglia di sfruttare i metodi *setXXX*)
- `public float getVolume()` calcola e restituisce il volume del collo
- `public void setH(int cm)` imposta l'altezza
- `public void setR(int cm)` imposta il raggio di base
- `public String toString()` ritorna una rappresentazione dello stato dell'oggetto

3 Parallelepipedo

Sottoclasse concreta di Collo, rappresenta un contenitore a forma di parallelepipedo, ha come attributi una altezza, larghezza e profondità (interi) e deve implementare almeno i seguenti metodi:

- `public Parallelepipedo(int cmx, int cmy, int cmz)` costruttore che accetta i parametri "altezza", "profondità" e "larghezza" (si consiglia di sfruttare i metodi *setXXX*)
- `public float getVolume()` calcola e restituisce il volume del collo
- `public void setX(int cm)` imposta una delle tre dimensioni
- `public void setY(int cm)` imposta una delle tre dimensioni
- `public void setZ(int cm)` imposta una delle tre dimensioni
- `public String toString()` ritorna una rappresentazione dello stato dell'oggetto

4 Indirizzo

Rappresenta un indirizzo "valido" (i singoli attributi devono essere non nulli per considerarlo valido), ha come attributi "nome" (stringa), "via" (stringa), "civico" (int), "cap" (CAP), "città" (stringa). Deve implementare almeno i seguenti metodi:

- `public Indirizzo(String nome, String via, int civico, int cap, String citta)` costruttore che a partire dai parametri passati (se validi) crea un'istanza di Indirizzo, lancia eccezione se qualche parametro non è valido (si consiglia di sfruttare i metodi *setXXX*)
- `public Indirizzo(String nome, String via, int civico, String cap, String citta)` costruttore che a partire dai parametri passati (se validi) crea un'istanza di Indirizzo, lancia eccezione se qualche parametro non è valido (si consiglia di sfruttare i metodi *setXXX*)
- `public Indirizzo(String nome, String via, int civico, CAP cap, String citta)` costruttore che a partire dai parametri passati (se validi) crea un'istanza di Indirizzo, lancia eccezione se qualche parametro non è valido (si consiglia di sfruttare i metodi *setXXX*)
- `public void setCAP(String cap)` imposta il cap (creandolo a partire da una stringa e controllandone la validità)
- `public CAP getCAP()` ritorna il CAP
- `public void setCitta(String citta)` imposta la città, la stringa non deve essere vuota, lancia eccezione se parametro nullo o non valido
- `public void setCivico(int civico)` imposta il civico, deve essere maggiore di 0, lancia eccezione se parametro nullo o non valido
- `public void setNome(String nome)` imposta il nome, la stringa non deve essere vuota, lancia eccezione se parametro nullo o non valido
- `public void setVia(String via)` imposta la via, la stringa non deve essere vuota, lancia eccezione se parametro nullo o non valido
- `public String toString()` ritorna una rappresentazione dello stato dell'oggetto

5 CAP

Rappresenta un codice di avviamento postale valido (numero maggiore di 0, lunghezza 5 caratteri) e deve implementare almeno i seguenti metodi:

- `public CAP(int cap)` costruttore che accetta un intero, ne controlla la validità e crea l'istanza, lancia eccezione se il numero passato non rispetta i criteri di validità (si consiglia di sfruttare i metodi *setXXX*)
- `public CAP(String cap)` costruttore che accetta una stringa, ne controlla la validità e crea l'istanza, lancia eccezione se il numero passato non rispetta i criteri di validità (si consiglia di sfruttare i metodi *setXXX*)
- `public int getCap()` restituisce il numero corrispondente
- `public void setCap(int cap)` imposta il cap a partire da un intero, ne controlla la validità, lancia eccezione se il numero passato non rispetta i criteri di validità
- `public void setCap(String cap)` imposta il cap a partire da una stringa, ne controlla la validità, lancia eccezione se il valorepassato non rispetta i criteri di validità
- `public String toString()` ritorna una rappresentazione dello stato dell'oggetto

6 Pallet

Contenitore di colli, accetta colli fino a raggiungimento della capacità (peso e/o volume) e deve implementare almeno i seguenti metodi:

- `public Pallet(float maxV, float maxP)` costruttore che accetta i due parametri che rappresentano i criteri massimi di accettazione dei colli (maxV e maxP > 0 !!!) (si consiglia di sfruttare i metodi *setXXX*)
- `public float getPeso()` ritorna il peso totale (sommando il peso dei colli contenuti)
- `public float getVolume()` ritorna il volume totale (sommando il volume dei colli contenuti)
- `public boolean metti(Collo c)` accetta (se non raggiunti limiti di peso e/o volume) un collo, se accetta ritorna true, altrimenti false
- `public void ordina()` ordina il contenuto del pallet per volume
- `public int quanti()` ritorna il numero di colli contenuti
- `public void setMaxPeso(float maxP)` imposta (controllando che sia > 0) il peso massimo raggiungibile
- `public void setMaxVolume(float maxV)` imposta (controllando che sia > 0) il volume massimo raggiungibile
- `public Collo togli(int quale)` rimuove un collo (per indice) dal pallet, restituendone l'istanza, ritorna `null` se l'indice non è valido
- `public String toString()` ritorna una rappresentazione dello stato dell'oggetto

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.
