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

ESAME del 19 Settembre 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" (se non è eseguibile, renderlo tale col comando `chmod`) 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*).
- Un corpo di metodo piú lungo di una decina di righe è un buon indice di errore di ragionamento
- Se avete dubbi sulla interpretazione del testo chiedete!

ESERCIZIO FILTRO

===>>> INIZIARE PRIMA CON QUESTO, se non si è in grado di portare a termine questo esercizio... NON PROSEGUIRE!!! (la correzione del resto dell'elaborato è subordinata alla correttezza di questo primo esercizio)

Realizzare una classe `Perfetto`, dotata del solo `main`, che, letto un numero n naturale da riga di comando, verifica se n è un numero perfetto, cioè se è uguale alla somma di tutti i suoi divisori propri, che sono tutti i divisori compreso l'1 ed escluso il numero stesso. Ad esempio il numero 6, che ha come divisori propri 1, 2, 3, è un numero perfetto. Il programma deve stampare "SI, il numero n e' perfetto" oppure "NO, il numero n non e' perfetto", a seconda che n sia perfetto o no.
Non è richiesto nessun controllo sull'input, si assuma cioè che l'input sia corretto sintatticamente e semanticamente, e non vuoto.

Ecco due possibili **esempi** di esecuzione:

```
$ java Perfetto 5
NO, il numero 5 NON e' perfetto

$ java Perfetto 6
SI, il numero 6 e' perfetto
```

---

1 Tema d'esame

Lo scopo dell'esercizio è realizzare una classe che rappresenta semplici *monomi* e che implementa alcune operazioni algebriche definite su di essi. Un monomio è un prodotto di un fattore numerico (chiamato *coefficiente*) e di fattori letterali, con i fattori letterali espressi come potenze con esponente intero positivo (es. a^2, b^{11}, c). Ad esempio il monomio $12a^2b^5c$ è il prodotto del *coefficiente* 12 e dei fattori che costituiscono la *parte letterale* $a^2b^5c^1$. Per semplicità assumeremo che il coefficiente sia espresso *sempre* all'inizio del monomio; assumeremo altresí che i simboli nella parte letterale *non* siano ripetuti ed i relativi esponenti siano > 0 (cioè che il monomio sia in forma *normale*), e che siano usate *solo* lettere minuscole dalla 'a' alla 'e'. Quindi $12a^2b^5c$, $-1.5e^{123}$, -3, 0 sono tutti monomi ammessi mentre $1a^{-1}b$ e $2g^2$ non lo sono. Un simbolo che non compare nella parte letterale (ad esempio d o e in $12a^2b^5c$) ha implicitamente esponente nullo.
Dicesi *nullo* un qualsiasi monomio con coefficiente 0 (e la forma normale per i monomi nulli è semplicemente 0). Monomi che hanno la stessa parte letterale (es. $6b^2e^5$ e $11b^2e^5$) sono detti *simili*.

Ai fini dell'esercizio, la rappresentazione testuale adottata per i monomi prevede che ogni simbolo della parte letterale sia *sempre* seguito dal relativo esponente. Quindi i monomi di cui sopra sono rappresentati, nell'ordine, dalle stringhe: `12a2b5c1`, `-1.5e123`, `-3`, `0`.

Operazioni Monomi simili possono essere sommati (ridotti): il risultato è un monomio simile il cui coefficiente è la somma algebrica dei coefficienti: ad esempio la somma di `12a1b2` e `-3a1b2` è `9a1b2`.
L'opposto di un monomio è un monomio simile il cui coefficiente è l'opposto del coefficiente originale.
Il prodotto di due monomi è un monomio ottenuto moltiplicando i rispettivi coefficienti e le parti letterali: ad esempio il prodotto di `2a1b2` e `-1a2c1` è `-2a3b2c1`.
Un monomio è divisibile per un secondo monomio (non nullo), se esiste un terzo monomio (detto quoziente) che moltiplicato per il secondo dà come risultato il primo. In particolare, il monomio nullo è divisibile per qualsiasi monomio non nullo. Ad esempio `-2a3b2c1` è divisibile per `-1a2c1`, ed il quoziente è `2a1b2`. Viceversa, il secondo non è divisibile per il primo.

Classi Le classi da realizzare sono le seguenti (dettagli nelle sezioni successive):

- `Monomio`: rappresenta dei monomi

2 Specifica delle classi

Le classi (**pubbliche**!) dovranno esporre almeno i metodi e costruttori **pubblici** specificati, piú eventuali altri metodi e costruttori *privati*, se ritenuti opportuni.
Gli attributi (campi) delle classi devono essere *privati*.
Se si usano tipi generici, si suggerisce di utilizzarne le versioni opportunamente istanziate (es. `ArrayList<String>` invece di `ArrayList`).

2.1 class Monomio

Rappresenta dei monomi (in *forma normale*) con lettere dalla 'a' alla 'e'. Deve disporre dei seguenti costruttori e metodi pubblici. Nell'implementazione della classe potrebbe essere utile il metodo statico `equals` della classe `Arrays`, che si trova nel package `util`.

- `Monomio(double coeff, int[] esponenti)`
  Costruttore che accetta come argomenti un double (il coefficiente del monomio) e un array di 5 interi (gli esponenti per a, b, ..., e rispettivamente). Il costruttore deve verificare che gli esponenti siano 5 e abbiano tutti valore maggiore o uguale a zero. In caso contrario solleva l'eccezione `IllegalArgumentException`.
- `Monomio(double coeff)` Costruttore che costruisce un monomio costante; accetta come argomento solo un double, il coefficiente).
- `Monomio(double coeff, char sym, int exp)` Costruttore che costruisce un monomio che ha un solo simbolo; accetta come argomento un double (il coefficiente del monomio), un char (il simbolo) e un int (l'esponente del simbolo). Solleva l'eccezione `IllegalArgumentException` se l'esponente non ha valore maggiore o uguale a zero o se il simbolo non è compreso tra `'a'` e `'e'`.
- `double coefficiente()`
  Restituisce il coefficiente del monomio.
- `int getEsponente (char x)`
  Restituisce l'esponente del simbolo `x`. Solleva una eccezione `IndexOutOfBoundsException` se `x` non è compreso tra `'a'` e `'e'`.
- `void setEsponente (char x, int exp)`
  Imposta a `exp` l'esponente del simbolo `x`. Solleva un'eccezione `IndexOutOfBoundsException` se `x` non è compreso tra `'a'` e `'e'`. Solleva un'eccezione `IllegalArgumentException` se `exp` è < 0
- `String toString()`
  Fornisce una descrizione testuale dell'istanza, nel formato descritto in precedenza (cioè ad esempio per $12a^2b^5c$ restituisce `12a2b5c1`). In particolare restituisce `"0"` se il monomio è nullo. Si ricordi che i fattori letterali con esponente nullo vanno omessi.
- `boolean equals(Object o)`
  Restituisce `true` se e soltanto se `o` è un monomio equivalente all'istanza.
- `boolean simile (Monomio m)`
  Restituisce `true` se e soltanto se `m` ha la stessa parte letterale dell'istanza.
- `Monomio opposto()`
  Restituisce l'opposto dell'istanza.
- `Monomio somma (Monomio m)`
  Restituisce la somma di `m` e dell'istanza, se sono monomi simili; altrimenti restituisce `null`.

- `Monomio prodotto (Monomio m)`
  Restituisce il prodotto di `m` per l'istanza.
- `Monomio quoziente (Monomio m)`
  Restituisce il risultato della divisione dell'istanza per `m`, se sono divisibili; altrimenti restituisce `null`.

3 Raccomandazioni

La codifica delle classi dovrebbe essere svolta in modo incrementale, ed accompagnata da frequenti sessioni di compilazione e da test (parziali). Logicamente l'esercizio è diviso in tre blocchi sequenziali: implementazione del costruttore e dei metodi di base della classe `Monomio`; implementazione dei metodi che realizzano le operazioni; implementazione della sottoclasse.
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 void nomeDelMetdo () {
   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

Per la consegna, eseguite l'upload dei SINGOLI file sorgente (NON un file archivio!) dalla pagina web: http://upload.di.unimi.it
Si ricorda che:

- le classi devono essere tutte *public*
- vanno consegnati tutti (e soli) i file *.java* prodotti
- si consiglia di fare upload successivi e frequenti, i docenti vedono solo l'ultima versione di ogni file
- NON va consegnato un file "archivio"! (NO tar, zip, etc.)
- NON vanno consegnati i *.class*
- NON vanno consegnati i file relativi al meccanismo di autovalutazione (*Test_*.java, AbstractTest.java, *.sh*)
- eseguite l'upload dei SINGOLI file sorgente (http://upload.di.unimi.it) nella sessione "Trentini"

*** 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).
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`.
