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

ESAME del 24 Febbraio 2015

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.

1 Tema d'esame

Lo scopo dell'esercizio è realizzare delle classi che rappresentano dei *monomi* e che implementano le principali operazioni algebriche definite su di essi. Un monomio è un prodotto di fattori numerici e letterali, possibilmente ripetuti, con i fattori letterali espressi come potenze con esponente intero, positivo o nullo (es. a^2, x^{11}, z^0). Per semplicità assumeremo che vi sia un unico fattore numerico (chiamato *coefficiente*), che sia espresso *sempre*, all'inizio del monomio, e che sia un intero relativo; non si faranno invece assunzioni sul numero di fattori letterali. Quindi $2ab^2ac$, $-1a^0z^3$, -3, $0a^2$ sono tutti dei monomi mentre $1a^{-1}b$ non lo è. Assumeremo altresì che i simboli nella parte letterale siano lettere dalla 'a' alla 'z', cioè solo minuscole.
Dicesi *nullo* un qualsiasi monomio con coefficiente 0. Un monomio non nullo è in *forma normale* se ogni simbolo compare al più una volta, con esponente > 0. La forma normale per i monomi nulli è 0. Pertanto le forme normali dei monomi precedenti sono, nell'ordine: $2a^2b^2c$, $-1z^3$, -3, 0. Monomi le cui forme normali hanno la stessa parte letterale 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: `2a1b2a1c1`, `-1a0z3`, `-3`, `0a2`. Le rispettive forme normali sono: `2a2b2c1`, `-1z3`, `-3`, `0`.
Usando le classi specificate in seguito dovrà essere possibile definire dei monomi partendo da una forma generica (con le assunzioni indicate all'inizio), ma la rappresentazione adottata internamente dovrebbe corrispondere a quella normale (tale indicazione va comunque intesa come suggerimento). Per l'output si deve utilizzare la forma normale.

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 `2a1b2` e `-3a1b2` è `-1a1b2`. 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 (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):

- `Monomial`: rappresenta dei generici monomi
- `SimpleMonomial`: sottoclasse di `Monomial`, rappresenta monomi la cui parte letterale è formata da al più un simbolo (introdotta per convenienza, non per necessità)

2 Specifica delle classi

Le classi (**pubbliche**!) dovranno esporre almeno i metodi e costruttori **pubblici** specificati, più 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 Monomial

Rappresenta dei generici monomi. Deve disporre dei seguenti costruttori e metodi pubblici:

- `Monomial(String monomial)`
  Costruttore che accetta in input una stringa che descrive un generico monomio, ad esempio `"2a1b2a0c1"`. Il costruttore *dovrebbe* tradurre il monomio in una rappresentazione interna riconducibile alla forma normale. Come *unico* controllo, verifica che gli esponenti abbiano valori positivi o nulli, in caso contrario solleva l'eccezione `IllegalArgumentException`. Per il resto si assume che la stringa sia nel formato descritto in precedenza.
  Suggerimento: per una prima implementazione semplice (in "bozza") potete ipotizzare di ricevere solo stringhe normalizzate (in cui una lettera appare una sola volta) e solo in seguito estendere al caso generale. Nella sezione 3 si trovano utili indicazioni su come effettuare il parsing della stringa che descrive un monomio, e su come rappresentarlo internamente.
  Importante Il comportamento di *tutti* i metodi specificati nel seguito fa riferimento implicitamente alla forma normale del monomio. Inoltre i metodi che implementano le operazioni NON devono in alcun modo modificare gli operandi.
- `int coefficient()`
  Restituisce il cefficiente numerico del monomio.
- `String literal ()`
  Restituisce la parte letterale del monomio; la stringa vuota (`""`) nel caso essa non sia presente.
- `int getExp (char x)`
  Restituisce l'esponente del simbolo `x` (in particolare `0` se esso non compare nella parte letterale del monomio). Solleva una eccezione `IndexOutOfBoundsException` se `x` non è compreso tra `'a'` e `'z'`.
- `String toString()`
  Fornisce una descrizione testuale dell'istanza, nel formato descritto in precedenza. 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 similar (Monomial m)`
  Restituisce `true` se e soltanto se `m` ha la stessa parte letterale dell'istanza.
- `Monomial opposite()`
  Restituisce l'opposto dell'istanza.
- `Monomial sum (Monomial m)`
  Restituisce la somma di `m` e dell'istanza, se sono monomi simili; altrimenti restituisce `null`.
- `Monomial product (Monomial m)`
  Restituisce il prodotto di `m` per l'istanza.
- `Monomial quotient (Monomial m)`
  Restituisce il risultato della divisione dell'istanza per `m`, se sono divisibili; altrimenti restituisce `null`.

2.2 class SimpleMonomial

Sottoclasse di `Monomial`, rappresenta monomi la cui parte letterale è formata al più da un solo simbolo, come `1a3`. Implementa `Comparable<SimpleMonomial>`. *Ridefinisce* (per convenienza) i metodi che come risultato danno un `SimpleMonomial`, ed il cui comportamento è specificato nella superclasse.

- `SimpleMonomial(int coeff, char x, int exp)`
  Costruttore che accetta in input il coefficiente, un simbolo, ed il relativo esponente.
- `int compareTo(SimpleMonomial sm)`
  Confronta l'istanza con `sm` effettuando prima un confronto (lessicale) tra le rispettive parti letterali, quindi considerando i coefficienti. Ad esempio `1a3` deve risultare minore di `-1b4` mentre `1a3` maggiore di `-1a3`.
- `SimpleMonomial opposite()`
  ridefinizione del metodo della superclasse
- `SimpleMonomial sum (Monomial m)`
  ridefinizione del metodo della superclasse
- `SimpleMonomial quotient (Monomial m)`
  ridefinizione del metodo della superclasse

3 Suggerimenti per l'implementazione

Una maniera semplice per separare coefficiente, simboli letterali, e relativi esponenti, in una stringa che rappresenta un monomio, è dato dal metodo `split` della classe `String`. Esso divide una stringa in "token" usando come insieme di separatori quello descritto dalla stringa richiesta come argomento. Ecco un esempio d'uso:

```
String s = "-1b2c2a2a1";
String[] letters = s.split("[^a-z]+");
String[] numbers = s.split("[a-z]");
```

Il contenuto di `letters` e `numbers` è `[, b, c, a, a]` e, rispettivamente, `[-1, 2, 2, 2, 1]`. Proprio quello che volevamo (si noti che il primo elemento in `letters` è la stringa vuota). Se invece invocassimo il metodo sulla stringa `"-1"` otterremmo rispettivamente `[]` (array di lunghezza 0) e `[-1]`.

3.1 Per chi ha dimestichezza con gli array e i char

Il fatto che in Java il tipo `char` sia un intero – quindi utilizzabile in operazioni aritmetiche, come indice in strutture di dati (array, ecc.) e nei cicli – suggerisce di rappresentare la parte letterale di un monomio mediante l'uso di una struttura di dati, associando a ciascuna lettera dell'alfabeto minuscolo ('a' - 'z') il relativo esponente (si rammenti che i simboli che non appaiono nella parte letterale di un monomio è come se avessero esponente nullo). Sebbene la scelta della rappresentazione interna dei monomi sia assolutamente libera, quella suggerita qui permette di implementare i metodi in modo semplice.

3.2 Per chi ha dimestichezza con classi e oggetti o con stringhe

Una strada alternativa è quella di realizzare una classe Factor, contenente un char e un intero, che rappresenti un fattore letterale (lettera + esponente) del monomio e nel Monomial avere un array/Vector/ArrayList di istanze di Factor. Factor potrebbe avere alcuni metodi per la normalizzazione del monomio, ad esempio un "absorb(Factor altro)" che prenda due Factor, ad es. "a4" e "a2", e trasformi uno dei due in "a6".

3.3 Per chi ha dimestichezza con le stringhe

Un'ulteriore possibilità consiste nel creare (in Monomial), un array/Vector/ArrayList di String nella forma (lettera + esponente) che rappresentano i vari fattori letterali del monomio, senza ricorrere a una classe Factor.

4 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 `Monomial`; 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 fake () {
   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.

5 Consegna

Si ricorda che le classi devono essere tutte *public* e che vanno consegnati tutti 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

---

ATTENZIONE!!! NON VERRANNO VALUTATI GLI ELABORATI CON ERRORI DI COMPILAZIONE. UN SINGOLO ERRORE DI COMPILAZIONE INVALIDA **TUTTO** L'ELABORATO.

Per ritirarsi fare l'upload di un file vuoto di nome `ritirato.txt`.
