Esame di Programmazione II, 29 settembre 2023
(si consegnino i file `.java`)

Si crei un progetto Eclipse e il package `it.univr.instructions`*. Si copino al suo interno le classi del compito. Non si modifichino le dichiarazioni dei metodi e delle classi. Si possono definire altri campi, metodi, costruttori e classi, ma devono essere* `private`*. La soluzione che verrà consegnata dovrà compilare, altrimenti non verrà corretta.*

Un *macchina a stack* è un calcolatore che esegue una sequenza di istruzioni (*programma*), il cui effetto è di modificare uno stack di interi. Tipicamente, l'esecuzione di ciascuna istruzione aggiunge e/o toglie uno o più elementi dalla cima dello stack. Le istruzioni sono le seguenti:

- `push(c)`: aggiunge la costante intera c in cima allo stack;
- `pop`: rimuove dallo stack l'intero che si trova in cima; se lo stack fosse vuoto, blocca l'esecuzione con un errore;
- `add/sub/mul/div`: rimuove dallo stack i due interi i_1 ed i_2 che si trovano in cima (cioè la cima i_2 dello stack e l'elemento subito sotto i_1) e aggiunge al loro posto un intero, risultato della addizione/sottrazione/moltiplicazione/divisione (rispettivamente) di i_1 per i_2; se lo stack avesse meno di due elementi, blocca l'esecuzione con un errore; nel caso della `div`, se i_2 fosse zero blocca l'esecuzione con un errore (non si può dividere per zero);
- `repeat(c, ins)`: esegue c volte l'istruzione `ins`; si avrà sempre $c \geq 0$. Questa istruzione può generare un errore nel caso in cui `ins` generasse un errore.

Per esempio, l'esecuzione del programma

```
push(5) push(13) push(17) add
```

parte dallo stack vuoto, aggiunge 5 in cima allo stack, poi aggiunge 13 in cima allo stack, poi aggiunge 17 in cima allo stack e infine toglie 13 e 17 dallo stack e aggiunge 30 al loro posto. Alla fine lo stack conterrà 5 e, in cima, 30. Il risultato dell'esecuzione è proprio il numero (30 nel nostro esempio) che si trova infine in cima allo stack. Un altro esempio è

```
push(13) add
```

In questo caso l'esecuzione del programma fallisce perché l'istruzione `add` viene eseguita in uno stack che ha solo un elemento, 13, mentre la `add` richiede almeno due elementi nello stack per potere essere eseguita correttamente.

Esercizio 1 (2 punti). Si implementi l'eccezione controllata `IllegalProgramException`, con un unico costruttore, che riceve il messaggio (stringa) che descrive perché una esecuzione è fallita.

Esercizio 2 (7 punti). L'iterfaccia `Instruction` descrive le istruzioni: hanno un metodo che esegue l'istruzione a partire da un certo stack e un altro metodo per ottenere una rappresentazione stringa dell'istruzione, comodo per poterla stampare:

```
public interface Instruction {
  void execute(List<Integer> stack) throws IllegalProgramException;
  String toString();
}
```

Si definiscano le classi `PUSH`, `ADD`, `SUB`, `MUL` e `DIV`, che implementano `Instruction`. **Suggerimento:** si guardi la classe `POP`, già fatta e completa, e ci si ispiri ad essa.

Esercizio 3 (7 punti). Si implementi la classe `REPEAT`, che implementa `Instruction`, il cui unico costruttore riceve c e `ins`. Se c fosse negativo, deve lanciare una `IllegalArgumentException`.

Esercizio 4 (8 punti). Una macchina a stack è descritta dall'interfaccia `Machine`, il cui unico metodo restituisce il risultato dell'esecuzione di un programma:

```
public interface Machine {
  int getResult();
}
```

Si completi l'implementazione `SimpleMachine` di `Machine`, nell'unico `TODO` indicato nel codice.

Esercizio 5 (6 punti). Si definisca una sottoclasse `PrintingMachine` di `SimpleMachine` che si differenzia solo per un dettaglio: per ciascuna istruzione, che viene eseguita, stampa l'istruzione, seguita dallo stack risultante dopo la sua esecuzione.

Se tutto è corretto, l'esecuzione del `Main` (già completo, da non modificare) stamperà:

```
     * * * Eseguo [push(5), push(13), push(17), add] * * *

Simple machine:
result = 30

Printing machine:
push(5): [5]
push(13): [5, 13]
push(17): [5, 13, 17]
add: [5, 30]
result = 30

 * * * Eseguo [repeat(5, push(13)), repeat(4, add)] * * *

Simple machine:
result = 65

Printing machine:
repeat(5, push(13)): [13, 13, 13, 13, 13]
repeat(4, add): [65]
result = 65

 * * * Eseguo [repeat(5, push(13)), repeat(5, add)] * * *

Simple machine:
Errore: Operandi insufficienti per un'operazione binaria

Printing machine:
repeat(5, push(13)): [13, 13, 13, 13, 13]
repeat(5, add): Errore: Operandi insufficienti per un'operazione binaria
```