Esame di Programmazione II, 14 luglio 2014

Gli identificatori di un linguaggio di programmazione sono spesso scritti in *camel-style*, come ad esempio `camelsAreSweet`, oppure in *snake-style*, come ad esempio `snakes_are_slow_food`. Inoltre vengono spesso usati nella loro versione *progressiva*, come ad esempio `camelsAreSweet_113` oppure `snakes_are_slow_food_113`.

Si consideri la superclasse che definisce un *identificatore*, il cui unico metodo permette di tradurre un identificare in stringa:

```
public interface Identifier {
  @Override public String toString();
}
```

Esercizio 1 [9 punti] Si completi la seguente classe astratta, che implementa un identificatore composto da più parole. I costruttori controllano che le parole componenti non siano vuote e contengano solo caratteri alfabetici minuscoli. Inoltre, controllano che ci sia almeno una parola. Altrimenti lanciano un'eccezione:

```
public abstract class AbstractMultiWordsIdentifier implements Identifier {
  private final String[] words;

  // Fallisce con un'eccezione se non c'e' alcuna parola o se c'e' una parola vuota
  // o se una parola contiene un carattere non alfabetico minuscolo
  protected AbstractMultiWordsIdentifier(String... words)
    throws NoWordsProvidedException, WordIsNotAlphabeticalLowercaseException, EmptyWordException { ... }

  // Fallisce con un'eccezione nelle stesse condizioni viste sopra
  protected AbstractMultiWordsIdentifier(Iterable<String> words)
    throws NoWordsProvidedException, WordIsNotAlphabeticalLowercaseException, EmptyWordException { ... }

  // restituisce le parole fornite al momento della costruzione
  protected final String[] getWords() {
    return words;
  }
}
```

Si noti che si tratta di una classe astratta in cui il metodo `toString()` non è ancora implementato.

Esercizio 2 [3 punti] Si implementino le tre eccezioni usate nel punto precedente, tutte sottoclassi dell'eccezione unchecked `java.lang.IllegalArgumentException`. Le clausole `throws` usate nell'esercizio precedente sono necessarie o possono essere eliminate senza problemi di compilazione?

Esercizio 3 [3 punti] Si completi la seguente classe che implementa un identificatore camel-style, formato da più parole congiunte:

```
public class CamelStyleIdentifier extends AbstractMultiWordsIdentifier {
  protected CamelStyleIdentifier(String... words)
    throws NoWordsProvidedException, WordIsNotAlphabeticalLowercaseException, EmptyWordException { ... }

  protected CamelStyleIdentifier(Iterable<String> words)
    throws NoWordsProvidedException, WordIsNotAlphabeticalLowercaseException, EmptyWordException { ... }

  @Override public String toString() { ... }
}
```

Esercizio 4 [3 punti] Si completi la seguente classe che implementa un identificatore snake-style, formato da più parole congiunte:

```
public class SnakeStyleIdentifier extends AbstractMultiWordsIdentifier {
  protected SnakeStyleIdentifier(String... words)
    throws NoWordsProvidedException, WordIsNotAlphabeticalLowercaseException, EmptyWordException { ... }

  protected SnakeStyleIdentifier(Iterable<String> words)
    throws NoWordsProvidedException, WordIsNotAlphabeticalLowercaseException, EmptyWordException { ... }

  @Override public String toString() { ... }
}
```

Esercizio 5 [4 punti] Si completi la seguente classe, che implementa una versione progressiva di un altro identificatore `base`, in cui cioè viene aggiunto un numero non-negativo `num` in fondo alla rappresentazione stringa dell'identificatore, separandolo con un underscore:

```
public class ProgressiveIdentifier implements Identifier {
  ...
  protected ProgressiveIdentifier(Identifier base, int num) throws NegativeProgressiveNumberException { ... }

  @Override public String toString() { ... }
}
```

L'identificatore `base`, in fondo a cui si aggiunge il numero, può essere di qualsiasi tipo, anche di una classe di identificatore al momento non ancora definita.

Il costruttore deve lanciare una `NegativeProgressiveNumberException` se si tenta di usare un numero progressivo `num` negativo. Si scriva tale eccezione, sottoclasse di `java.lang.IllegalArgumentException`.

Se tutto è corretto, un'esecuzione del seguente `main`:

```
public class Main {
  public static void main(String[] args) {
    Iterable<String> words = readWords();

    System.out.println("In camel-style, le parole che hai inserito diventano");
    System.out.println(new CamelStyleIdentifier(words));
    System.out.println("In snake-style, le parole che hai inserito diventano");
    System.out.println(new SnakeStyleIdentifier(words));
    System.out.println("Le loro versioni progressive 816 sono");
    System.out.println(new ProgressiveIdentifier(new CamelStyleIdentifier(words), 816));
    System.out.println(new ProgressiveIdentifier(new SnakeStyleIdentifier(words), 816));
  }

  private static Iterable<String> readWords() {
    List<String> words = new ArrayList<String>();
    Scanner keyboard = new Scanner(System.in);

    while (true) {
      String word = keyboard.nextLine();
      if (word.equals("END"))
        break;
      words.add(word);
    }

    keyboard.close();

    return words;
  }
}
```

potrebbe essere:

```
camels    <-- input dell'utente
are       <-- input dell'utente
sweet     <-- input dell'utente
and       <-- input dell'utente
clever    <-- input dell'utente
animals   <-- input dell'utente
END       <-- input dell'utente
In camel-style, le parole che hai inserito diventano
CamelsAreSweetAndCleverAnimals
In snake-style, le parole che hai inserito diventano
camels_are_sweet_and_clever_animals
Le loro versioni progressive 816 sono
CamelsAreSweetAndCleverAnimals_816
camels_are_sweet_and_clever_animals_816
```