Esame di Programmazione II, 28 settembre 2018

Le note musicali sono distinte in 12 semitoni e vengono scritte diversamente fra italiano e inglese, come si vede nella tabella seguente:

semitono	italiano	inglese
0	do	C
1	do#	C#
2	re	D
3	re#	D#
4	mi	E
5	fa	F
6	fa#	F#
7	sol	G
8	sol#	G#
9	la	A
10	la#	A#
11	si	B

Il simbolo # in italiano si pronuncia *diesis* e in inglese *sharp*.

Esercizio 1 [6 punti] Si completi la seguente classe astratta che implementa una nota, genericamente, senza ancora distinguere fra italiana e inglese:

```
public abstract class Note implements Comparable<Note> {
  protected final int semitone;

  // inizializza la nota per il semitono indicato;
  // se il semitono non e' fra 0 e 11, deve lanciare una IllegalArgumentException
  protected Note(int semitone) { ... }

  @Override public abstract String toString();

  // due note sono uguali se e solo se hanno lo stesso semitono
  @Override public final boolean equals(Object other) { ... }

  // compatibile con equals() e non banale
  @Override public final int hashCode() { ... }

  // le note sono ordinate per semitono crescente
  @Override public final int compareTo(Note other) { ... }
}
```

Esercizio 2 [8 punti] Si scriva la sottoclasse concreta `ItalianNote` di `Note`, il cui metodo `toString()` usa la rappresentazione italiana delle note (`do`, `do#`, ...). Tale sottoclasse dovrà definire 12 costanti pubbliche `DO`, `DO_DIESIS`, `RE`, ..., `SI` di tipo `ItalianNote`, che contengono le note con tali nomi. Tali costanti devono essere gli unici oggetti di classe `ItalianNote` disponibili, nel senso che l'utilizzatore di `ItalianNote` non deve potere costruirne altri. Si definisca similmente la sottoclasse `EnglishNote` di `Note`, il cui metodo `toString()` usa la rappresentazione inglese delle note. Tale classe dovrà definire 12 costanti pubbliche `C`, `C_SHARP`, `D`, ..., `B` di tipo `EnglishNote` e dovrà impedire la costruzione di altre `EnglishNote`.

Esercizio 3 [**5 punti**] Si completi la seguente classe di test JUnit:

```
public class NoteTest {
  // asserisce che il toString() del do italiano e' la stringa "do"
  @Test public void testToString() { ... }

  // asserisce che l'italiano re# e l'inglese D# sono equals()
  @Test public void testEquals() { ... }

  // asserisce che l'italiano re# e l'inglese D# hanno lo stesso hashCode()
  @Test public void testHashCode() { ... }

  // asserisce che l'italiano re# e l'inglese E hanno hashCode() diversi
  @Test public void testHashCodeNonTrivial() { ... }

  // asserisce che l'italiano do viene prima dell'inglese D
  @Test public void testCompareTo() { ... }
}
```

Esercizio 4 [**11 punti**] Si completi la seguente classe, che rappresenta una canzone: contiene il testo della canzone (di una sola riga) e permette di posizionare delle note sul testo:

```
public class Song {
  ...
  public Song(String lyrics) { ... } // lyrics e' il testo della canzone (una riga)

  // posiziona la nota alla posizione indicata. Deve lanciare una IllegalArgumentException
  // se position non e' dentro il testo della canzone. Deve lanciare una
  // NoteAtSamePositionException se c'e' gia' una nota alla posizione indicata
  public void add(Note note, int position) { ... }

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

Esercizio 5 [**2 punti**] Si definisca l'eccezione non controllata `NoteAtSamePositionException`.

Se tutto è corretto, l'esecuzione del seguente codice:

```
Song yellowSubmarine = new Song("In the town where I was born lived a man who sailed the sea");
yellowSubmarine.add(EnglishNote.C_SHARP, 7); // C# su "town"
yellowSubmarine.add(EnglishNote.C_SHARP, 56); // C# su "sea"
yellowSubmarine.add(EnglishNote.F_SHARP, 24); // F# su "born"
yellowSubmarine.add(EnglishNote.G_SHARP, 37); // G# su "man"
// yellowSubmarine.add(EnglishNote.A, 24); // -> NoteAtSamePositionException
System.out.println(yellowSubmarine);
```

dovrà stampare:

```
       C#                F#           G#                  C#
In the town where I was born lived a man who sailed the sea
```

Si possono definire altri campi, metodi, costruttori e classi, ma solo `private`**.**