Programmazione per Bioinformatica Anno Accademico 2013/2014 **Prova di Esame del 19 Settembre 2014**	*tempo a disposizione: 3 ore* *consegnare solo la 'bella'*

– Stringhe –

Esercizio 1 [7 punti]

Si scriva un metodo

```
public static boolean verificaCaratteri(String a, String b)
```

che riceve come parametri due stringhe, `a` e `b`, e ritorna `true` se tutti i caratteri di `a` sono contenuti in `b`, altrimenti ritorna `false`.

Per esempio, l'esecuzione del seguente `main`

```
public static void main(String[] args) {
   System.out.print(verificaCaratteri("pesce",
      "precipitevolissimevolmente") + ", ");
   System.out.print(verificaCaratteri("riccio",
      "precipitevolissimevolmente") + ", ");
   System.out.print(verificaCaratteri("trota",
      "precipitevolissimevolmente") + ", ");
   System.out.println(verificaCaratteri("sgombro",
      "precipitevolissimevolmente"));
}
```

deve stampare a video: `true, true, false, false`.

– Ricorsione –

Esercizio 2 [7 punti]

Si definisca il *metodo ricorsivo* (non è ammesso l'uso di cicli!)

```
private static int contaOccorrenze(int[] vett, int val, int pos)
```

che riceve in input un array di interi `vett` e due interi, `val` e `pos`. Il metodo restituisce il numero di volte che il valore `val` compare in `vett` dalla posizione `pos` (compresa) fino alla fine. Se `pos` non è un indice valido, il metodo restituisce `0`.

Per esempio, l'esecuzione del seguente `main`

```
public static void main(String[] args) {
   int[] vett = { 23, 45, 32, 9, 23, 8, 9, 23, 45, 8 };

   System.out.println(
      contaOccorrenze(vett, 9, 0) + ", " +
      contaOccorrenze(vett, 6, 0) + ", " +
      contaOccorrenze(vett, 45, 3) + ", " +
      contaOccorrenze(vett, 45, -3));
}
```

deve stampare a video: `2, 0, 1, 0`

– Classi –

Esercizio 3 [9 punti] - **commentate il codice della classe** `Nota` **in stile** `javadoc` **-**

Le note musicali sono normalmente divise in 12 *semitoni*, come nella seguente tabella:

semitono	nome italiano	nome 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

Si completi la seguente classe che rappresenta una nota musicale:

```
public abstract class Nota {
   // inserire qui eventuali campi

   protected Nota(int semitono) {
      // inserire qui il corpo del metodo
   }

   public abstract Nota incrementa(int inc);

   public int getSemitono() {
      // inserire qui il corpo del metodo
   }

   @Override
   public abstract String toString();
}
```

dove `getSemitono()` restituisce il semitono della nota, che era stato passato al costruttore. Il metodo `incrementa()`, che deve essere implemetato nelle sottoclassi concrete, restituisce una nota ottenuta aumentando il semitono della nota di `inc` semitoni. Si noti che la scala è circolare per cui, dopo il *si*, si ritorna sul *do*.

Si scrivano poi le sottoclassi concrete `NotaIT` e `NotaUK` di `Nota` che si differenziano per il metodo `toString()`, che restituisce una rappresentazione della nota usando, rispettivamente, la notazione italiana e quella inglese.

Esercizio 4 [9 punti]

Si completi la seguente classe. Una canzone è rappresentata dal `nome` e dalle `note` che la compongono.

```
public class Canzone {
   // inserire qui eventuali campi

   public Canzone(String nome, Nota... note) {
      // inserire qui il corpo del costruttore
   }

   public String getNome() {
      // inserire il corpo del metodo
   }

   public Canzone getCanzoneTrasportata(int inc) {
      // inserire il corpo del metodo
   }

   @Override
   public final String toString() {
      // inserire il corpo del metodo
   }
}
```

Il metodo `getNome()` restituisce il nome delle canzone. Il metodo `toString()` restituisce una stringa con il nome della canzone e tutte le note che la compongono, nel giusto ordine di esecuzione. Il metodo `getCanzoneTrasportata()` restituisce una `Canzone` identica a `this`, ma le cui note sono spostate di `inc` semitoni rispetto a quelle di `this`.

Se tutto è corretto, il seguente programma:

```
public class Main {
   public static void main(String[] args) {
      Canzone boccaDiRosa = new Canzone("Bocca di Rosa", new NotaIT(11),
            new NotaIT(2), new NotaIT(3), new NotaIT(6));
      Canzone letItBe = new Canzone("Let it be", new NotaUK(3),
            new NotaUK(5), new NotaUK(0));

      System.out.println(boccaDiRosa);
      System.out.println(letItBe);
      System.out.println(boccaDiRosa.getCanzoneTrasportata(5));
      System.out.println(letItBe.getCanzoneTrasportata(23));
   }
}
```

stamperà:

```
Bocca di Rosa: si re re# fa#

Let it be: D# F C

Bocca di Rosa: mi sol sol# si

Let it be: D E B
```

Appendice:
Java Platform, Standard Edition 7 API Specification for Class `String`

`char charAt(int index)`: Returns the char value at the specified index.

`int compareTo(String anotherString)`: Compares two strings lexicographically.

`int compareToIgnoreCase(String str)`: Compares two strings lexicographically, ignoring case differences.

`String concat(String str)`: Concatenates the specified string to the end of this string. If the length of the argument string is `0`, then this `String` object is returned. Otherwise, a new `String` object is created, representing a character sequence that is the concatenation of the character sequence represented by this `String` object and the character sequence represented by the argument string.

`boolean equals(Object anObject)`: Compares this string to the specified object.

`boolean equalsIgnoreCase(String anotherString)`: Compares this `String` to another `String`, ignoring case considerations.

`int indexOf(int ch)`: Returns the index within this string of the first occurrence of the specified character, or `-1` if the character does not occur.

`int indexOf(int ch, int fromIndex)`: Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.

`int indexOf(String str)`: Returns the index within this string of the first occurrence of the specified substring.

`int indexOf(String str, int fromIndex)`: Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

`boolean isEmpty()`: Returns true if, and only if, `length()` is `0`.

`int lastIndexOf(int ch)`: Returns the index within this string of the last occurrence of the specified character.

`int lastIndexOf(int ch, int fromIndex)`: Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.

`int lastIndexOf(String str)`: Returns the index within this string of the last occurrence of the specified substring.

`int lastIndexOf(String str, int fromIndex)`: Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.

`int length()`: Returns the length of this string.

`String replace(char oldChar, char newChar)`: Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.

`String replace(CharSequence target, CharSequence replacement)`: Replaces each substring of this string that matches the literal `target` sequence with the specified literal `replacement` sequence.

`String replaceAll(String regex, String replacement)`: Replaces each substring of this string that matches the `regex` with the given `replacement`.

`String replaceFirst(String regex, String replacement)`: Replaces the first substring of this string that matches the `regex` with the given `replacement`.

`boolean startsWith(String prefix)`: Tests if this string starts with the specified `prefix`.

`boolean startsWith(String prefix, int toffset)`: Tests if the substring of this string beginning at the specified index starts with the specified `prefix`.

`String substring(int beginIndex)`: Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

`String substring(int beginIndex, int endIndex)`: Returns a new string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Thus the length of the substring is `endIndex - beginIndex`.

`String toLowerCase()`: Returns the `String` object obtained by converting all of the characters in this `String` to lower case.

`String toUpperCase()`: Returns the `String` object obtained by converting all of the characters in this `String` to upper case.

`String trim()`: Returns a copy of this string, with leading and trailing whitespace omitted.