Programmazione per Bioinformatica Anno Accademico 2014/2015 **Appello del 16 Luglio 2015**	*tempo a disposizione: 2 ore* *consegnare solo la "bella"*

– Stringhe –

Esercizio 1 [6 punti]

Si scriva un programma completo che chieda in input all'utente una stringa di testo e poi stampi a video la sua conversione in termini di basi azotate del DNA. Per effettuare la conversione, il programma dovrà prima trasformare il testo tutto in maiuscolo e poi dovrà eliminare ogni carattere non corrispondente ad una delle 4 basi azotate del DNA (A, C, G, T).

Per esempio, se l'utente inserisce la stringa "cgd*TTqa1 ecgGh", il programma dovrà stampare a video "CGTTACGG".

Suggerimento: per confezionare l'algoritmo è possibile utilizzare i metodi per le stringhe di java. La loro documentazione è riportata alla fine del testo.

– Ricorsione –

Esercizio 2 [8 punti]

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

```
public static int getMaxIndex(int[] v)
```

che riceve in input un array di interi `v`. Il metodo restituisce l'indice del massimo elemento tra tutti i valori che si trovano in `v`. Nel caso ci siano più elementi di valore massimo, il metodo deve ritornare indifferentemente l'indice di uno di essi. Per l'implementazione è consentita la creazione di metodi di supporto (qualora si ritenessero necessari).

Per esempio, l'esecuzione del seguente `main`:

```
public static void main(String[] args) {

      int[] v1 = {1, -2, 5, 2, 3};
      int[] v2 = {8, -2, 5, 2, 3};
      int[] v3 = {1, -2, 5, 2, 8};

      System.out.println("Indice massimo elemento in v1: " + getMaxIndex(v1));
      System.out.println("Indice massimo elemento in v2: " + getMaxIndex(v2));
      System.out.println("Indice massimo elemento in v3: " + getMaxIndex(v3));
}
```

deve stampare a video:

```
Indice massimo elemento in v1: 2
Indice massimo elemento in v2: 0
Indice massimo elemento in v3: 4
```

– Classi ed interfacce –

Esercizio 3 [6 punti]

Si risponda alle seguenti domande:

1. Cosa è una interfaccia ed in cosa si distingue da una classe astratta?
2. A cosa servono le parole chiave `extends` ed `implements`?
3. Si consideri la seguente interfaccia, relativa ad una struttura dati di tipo insieme di interi:

```
// Interfaccia di un insieme di valori interi.
public interface IntegerSet {

	// Metodo che svuota l'insieme.
	public void clear();

	// Metodo che restituisce il numero di elementi salvati.
	public int size();

	// Metodo che rimuove dall'insieme l'elemento (se presente).
	public void removeElement(int e);

	// Metodo che inserisce nell'insieme l'elemento (a patto che non sia gia' incluso).
	public void insertElement(int e);

	// Metodo che restituisce gli elementi dell'insieme memorizzati in un vettore di inte
	// (restituisce null se l'insieme e' vuoto).
	public int[] getElements();
}
```

   Si riscriva l'interfaccia in modo che sia documentata in stile JavaDoc.

Esercizio 4 [12 punti]

Si crei una classe `Insieme` che implementa l'interfaccia `IntegerSet` presentata nell'esercizio precedente. La classe deve fornire una implementazione completamente funzionante della struttura dati. Lo studente è libero di scegliere la strategia che ritiene più opportuna per la memorizzazione interna degli elementi dell'insieme, ma non è consentito l'uso di strutture dati avanzate come quelle che implementano le interfacce Java `List` e `Set` (l'utilizzo di tali strutture comporterà l'annullamento totale del punteggio dell'esercizio).

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.

`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.