Programmazione per Bioinformatica Anno Accademico 2013/2014 **Prova di Esame dell'11 Luglio 2014**	*tempo a disposizione: 3 ore* *consegnare solo la 'bella'*

– Stringhe –

Esercizio 1 [7 punti]

Si scriva un metodo

```
public static String raggruppaVocaliConsonanti(String s)
```

che riceve come parametro la stringa `s` e restituisce la stringa ottenuta da `s` spostando all'inizio le vocali (a, e, i, o, u) e alla fine le consonanti. Per esempio, la chiamata

```
raggruppaVocaliConsonanti("PrecipiTevoLissimevolmentE")
```

deve restituire la stringa `"eiieoiieoeEPrcpTvLssmvlmnt"`.
Se necessario, si possono definire dei metodi ausiliari.

– Ricorsione –

Esercizio 2 [7 punti]

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

```
private static int indexMaxEl(int[] vett, int pos)
```

che riceve in input un array di interi `vett` e un intero `pos`. Il metodo restituisce l'indice del massimo elemento tra tutti i valori che si trovano in `vett` dalla posizione `pos` (compresa) fino alla fine. Se `pos` non è un indice valido, il metodo restituisce `-1`.

Per esempio, l'esecuzione del seguente `main`

```
public static void main(String[] args) {
   int[] vett = { 1, 2, 62, 6, 48, 12, 13, 24, 36, 36 };

   System.out.println(indexMaxEl(vett, 0));
   System.out.println(indexMaxEl(vett, 5));
   System.out.println(indexMaxEl(vett, 10));
}
```

deve stampare a video

```
2
8
-1
```

– Classi –

Si consideri le seguente classe, le cui istanze rappresentano dei partiti politici che si presentano alle elezioni:

```
public class Partito {

  // Il nome del partito.
  private final String nome;


  // Voti ricevuti dal partito.
  private int votiRicevuti;


  // Crea un partito con un dato nome.
  // Inizialmente il partito ha zero voti ricevuti.
  public Partito(String nome) {
    this.nome = nome;
    votiRicevuti = 0;
  }

  // Restituisce il nome del partito.
  public String getNome() {
    return nome;
  }

  // Restituisce i voti ricevuti finora dal partito.
  public int getVoti() {
    return votiRicevuti;
  }

  // Incrementa di uno i voti ricevuti dal partito.
  public void aggiungiVoto() {
    votiRicevuti++;
  }

  @Override
  public boolean equals(Object altro) {
    return (altro instanceof Partito) && ((Partito) altro).nome.equals(nome);
  }
}
```

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

Si definisca una classe `Coalizione`, che rappresenta una coalizione di uno o più partiti, con costruttore e metodi pubblici:

- `Coalizione(String nome, Partito... partiti)` che costruisce una coalizione con il nome dato e di cui fanno parte i partiti indicati.
- `String getNome()` che restituisce il nome della coalizione.
- `public Partito[] getPartiti()` che restiruisce un array contenente i partiti in coalizione.
- `public int getVoti()` che restituisce il totale dei voti ricevuti dalla coalizione.
- `public boolean include(Partito partito)` che verifica se il partito dato compare nella coalizione.

Inoltre tale classe deve avere un metodo `equals()` che determina se una coalizione è uguale a un'altra, cioè ha il suo stesso nome.

Esercizio 4 [9 punti]

Si definisca una classe `Elezione` che rappresenta un'elezione politica. La classe deve avere un campo `coalizioni` che raggruppa tutte le coalizioni che si presentano all'elezione. Inoltre deve avere il costruttore e i metodi pubblici elencati di seguito:

- `public Elezione()` che crea una nuova istanza di Elezione, con zero coalizioni.
- `public void registra(Coalizione coalizione)` che registra la coalizione indicata come partecipante all'elezione. Se nell'elezione esiste già una coalizione uguale (con lo stesso nome), il metodo deve generare una `CoalizioneGiaPresenteException`. Se uno dei partiti della coalizione appartiene già ad una coalizione precedentemente creata, deve generare una `PartitoGiaRegistratoException`.
- `public void registra(Partito partito)` che registra il partito indicato all'elezione, dentro una coalizione di cui fa parte solo il partito dato ed ha lo stesso nome del partito. Se una coalizione con lo stesso nome è già presentae, genera una `CoalizioneGiaPresenteException`. Se il partito è già registrato (in qualche coalizione), genera una `PartitoGiaRegistratoException`.
- `public void registraVoto(Partito partito)` che registra un nuovo voto per il partito indicato (e quindi anche per la coalizione di cui esso fa parte). Se il partito non è registrato all'elezione, il metodo genera una `PartitoMaiRegistratoException`.
- `public String toString()` che restituisce una stringa con i risultati dell'elezione. Se stampata a video, la stringa risultante deve mostrare qualcosa del tipo

```
coalizione "Partito dei belli"
    Partito dei belli, voti: 11
    voti di coalizione: 11
coalizione "Partito dei brutti"
    Partito dei brutti, voti: 15
    voti di coalizione: 15
coalizione "Siamo i piu' forti"
    Partito mai tornato, voti: 12
    Partito dei fiori, voti: 10
    voti di coalizione: 22
coalizione "Futuro radioso"
    Partito di tutti, voti: 12
    Partito di titti, voti: 14
    Partito dei nonni, voti: 15
    voti di coalizione: 41
```

Si definiscano inoltre tutte le classi delle *eccezioni* lanciate dai vari metodi.

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.