Corso di Laboratorio di Programmazione I

A.A. 2015/2016

Progetto

Regole per lo svolgimento:

- Il progetto deve essere svolto singolarmente o in gruppi di al massimo 2 persone.
- Il progetto deve essere consegnato **almeno 5 giorni prima** della prova orale della parte di Programmazione I e Laboratorio di Programmazione I relativa all'appello in cui si desidera registrare l'esame.
- NON è permesso utilizzare **nessuna libreria** del linguaggio Java.
- Il progetto deve contenere i metodi richiesti **rispettandone esattamente il nome, il tipo e l'ordine dei parametri formali, ed il tipo di ritorno**.
- Si è naturalmente liberi di sviluppare metodi aggiuntivi di supporto ai metodi richiesti, laddove lo si ritenga utile.
- La consegna (del file Stringhe.java) va effettuata all'indirizzo **luca.moscardelli@unich.it** mettendo come oggetto **[jmm] consegna progetto laboratorio 2016**
  e nel testo dell'email il/i nome/i dello/degli studente/i che consegna/no.

Testo del progetto:

Sviluppare in **Java** una libreria per la gestione delle stringhe tramite **array di char**.

ATTENZIONE: NON si deve far uso in nessun modo della classe String e dei suoi metodi, come anche di nessuna altra libreria del linguaggio Java**. NON** si deve far uso dell'operatore **+** usato come concatenazione di stringhe, né delle virgolette (").

La libreria deve essere composta dai seguenti metodi, tutti inclusi nella stessa classe **Stringhe** salvata nel file **Stringhe.java**.
Per tutti i metodi, se non diversamente specificato, si può assumere (senza effettuare controlli) che gli array di **char** passati come parametri e rappresentanti le stringhe **NON valgano null**.

1. `static boolean isEmpty (char[] string)`
   prende come parametro un array di **char** e restituisce **true** se esso ha lunghezza 0, **false** altrimenti.
2. `static int length (char[] string)`
   prende come parametro un array di **char** e restituisce la sua lunghezza.
3. `static char charAt (char[] string, int index)`
   prende come parametri un array **string** di **char** che rappresenta una stringa e un intero **index**, e ritorna il carattere presente nella stringa **string** alla posizione **index**. Se **index** non rientra nell'intervallo 0,…,string.lenght-1 viene restituito **0**.

4. `static char[] concat (char[] string1, char[] string2)`
   prende come parametri due array **string1** e **string2** di **char** che rappresentano due stringhe, e restituisce una nuova stringa ottenuta concatenando **string2** a **string1**.
5. `static boolean contains (char[] string, char[] substring)`
   prende come parametri due array **string** e **substring** di **char** che rappresentano due stringhe, e restituisce **true** se la stringa **string** contiene la stringa **substring** al suo interno, **false** altrimenti.
6. `static boolean equals (char[] string1, char[] string2)`
   prende come parametri due array **string1** e **string2** di **char** che rappresentano due stringhe, e restituisce **true** se **string1** e **string2** sono uguali, **false** altrimenti.
7. `static char[] substring (char[] string, int beginIndex, int endIndex)`
   prende come parametri un array **string** di **char** che rappresenta una stringa e due interi **beginIndex** e **endIndex**, e restituisce la sottostringa di **string** che inizia alla posizione **beginIndex** e termina alla posizione **endIndex**.
   Se **beginIndex** e/o **endIndex** non rientrano nell'intervallo 0,…,string.lenght-1, o se **endIndex**<**beginIndex**, viene restituito **null**.
8. `static boolean startsWith (char[] string, char[] prefix)`
   prende come parametri due array **string** e **prefix** di **char** che rappresentano due stringhe, e restituisce **true** se **string** inizia con **prefix**, **false** altrimenti.
9. `static boolean endsWith (char[] string, char[] suffix)`
   prende come parametri due array **string** e **suffix** di **char** che rappresentano due stringhe, e restituisce **true** se **string** termina con **suffix**, **false** altrimenti.
10. `static char[] trim (char[] string)`
    prende come parametro un array **string** di **char** che rappresenta una stringa e restituisce una nuova stringa che è una copia di **string** in cui sono però omessi tutti gli spazi iniziali e finali (laddove presenti).
11. `static char[] replace (char[] string, char oldChar, char newChar)`
    prende come parametri un array **string** di **char** che rappresenta una stringa e due char **oldChar** e **newChar**, e restituisce una nuova stringa che è una copia di **string** in cui sono però sostituite tutte le occorrenze di **oldChar** con **newChar**.
12. `static int indexOf (char[] string, char ch)`
    prende come parametri un array **string** di **char** che rappresenta una stringa e un char **ch**, e restituisce l'indice della prima occorrenza del carattere **ch** in **string**. Se non esiste nessuna occorrenza, viene restituito **-1**.
13. `static int indexOf (char[] string, char ch, int fromIndex)`
    prende come parametri un array **string** di **char** che rappresenta una stringa, un char **ch** e un intero **fromIndex**, e restituisce l'indice della prima occorrenza del carattere **ch** in **string**, iniziando la ricerca a partire da **fromIndex.** Se non esiste nessuna occorrenza, viene restituito **-1**.
14. `static int indexOf (char[] string, char[] substring)`
    prende come parametri due array **string** e **substring** di **char** che rappresentano due stringhe, e restituisce l'indice della prima occorrenza della sottostringa **substring** in **string**. Se non esiste nessuna occorrenza, viene restituito **-1**.
15. `static int indexOf (char[] string, char[] substring, int fromIndex)`
    prende come parametri due array **string** e **substring** di **char** che rappresentano due stringhe

e un intero **fromIndex**, e restituisce l'indice della prima occorrenza della sottostringa **substring** in **string**, iniziando la ricerca a partire da **fromIndex.** Se non esiste nessuna occorrenza, viene restituito **-1**.

16. `static int lastIndexOf (char[] string, char ch)`

prende come parametri un array **string** di **char** che rappresenta una stringa e un char **ch**, e restituisce l'indice dell'ultima occorrenza del carattere **ch** in **string**. Se non esiste nessuna occorrenza, viene restituito **-1**.

17. `static int lastIndexOf (char[] string, char ch, int fromIndex)`

prende come parametri un array **string** di **char** che rappresenta una stringa, un char **ch** e un intero **fromIndex**, e restituisce l'indice dell'ultima occorrenza del carattere **ch** in **string**, iniziando la ricerca a ritroso a partire da **fromIndex.** Se non esiste nessuna occorrenza, viene restituito **-1**.

18. `static int lastIndexOf (char[] string, char[] substring)`

prende come parametri due array **string** e **substring** di **char** che rappresentano due stringhe, e restituisce l'indice dell'ultima occorrenza della sottostringa **substring** in **string**. Se non esiste nessuna occorrenza, viene restituito **-1**.

19. `static int lastIndexOf (char[] string, char[] substring, int fromIndex)`

prende come parametri due array **string** e **substring** di **char** che rappresentano due stringhe e un intero **fromIndex**, e restituisce l'indice dell'ultima occorrenza della sottostringa **substring** in **string**, iniziando la ricerca a ritroso a partire da **fromIndex.** Se non esiste nessuna occorrenza, viene restituito **-1**.

> ***Definizione*: Ordinamento lessicografico**
> Una stringa s precede una stringa t nell'ordinamento lessicografico se:
> - s è un prefisso di t, oppure
> - se c1 (appartenente a s) e c2 (appartenente a t) sono i primi caratteri di s e t in cui s e t differiscono, allora c1<c2.

20. `static int compare (char[] string1, char[] string2)`

prende come parametri due array **string1** e **string2** di **char** che rappresentano due stringhe, e restituisce un numero negativo se **string1** precede **string2** secondo l'ordinamento lessicografico, **0** se **string1** e **string2** sono uguali, un numero positivo se **string2** precede **string1** nell'ordinamento lessicografico.
In particolare, quando string1 precede string2 nell'ordinamento lessicografico, viene restituito

- **string1.length-string2.length** se **string1** è un prefisso di **string2**;
- **c1-c2** se c1 (appartenente a **string1**) e c2 (appartenente a **string2**) sono i primi caratteri di **string1** e **string2** in cui **string1** e **string2** differiscono.

In modo analogo, quando string2 precede string1 nell'ordinamento lessicografico, viene restituito

- **string1.length-string2.length** se **string2** è un prefisso di **string1**;
- **c1-c2** se c1 (appartenente a **string1**) e c2 (appartenente a **string2**) sono i primi caratteri di **string1** e **string2** in cui **string1** e **string2** differiscono.

Per chi svolge il progetto in coppia, bisogna svolgere anche i seguenti metodi:

21. `static char[] valueOf (boolean b)`
    prende come parametro un booleano **b** e crea e restituisce una nuova stringa che lo rappresenta (come “true” o come “false” a seconda del valore di **b**).
22. `static char[] valueOf (char c)`
    prende come parametro un char **c** e crea e restituisce una nuova stringa che ha **c** come unico carattere.
23. `static char[] valueOf (long n)`
    prende come parametro un long **n** e crea e restituisce una nuova stringa che lo rappresenta.
24. `static char[] valueOf (double x)`
    prende come parametro un double **x** e crea e restituisce una nuova stringa che lo rappresenta.