PROGRAMMAZIONE (9 CFU) – A.A. 2016-2017	ES. 12

1) Realizzate il crivello di Eratostene, un metodo per calcolare i numeri primi noto agli antichi greci. Scegliete un numero *n*: questo metodo calcolerà tutti i numeri primi fino a *n*. Come prima cosa inserite in un **Set** tutti i numeri da *2* a *n*. Poi, cancellate tutti i multipli di *2* (eccetto *2*); vale a dire *4, 6, 8*, … Dopodiché cancellate tutti i multipli di *3* (eccetto *3*), cioè *6, 9, 12*, … Arrivate fino a $n^{1/2}$, quindi visualizzate il **Set**.

2) Scrivete un programma che usi una **Map** in cui sia le chiavi sia i valori sono stringhe: rispettivamente, i nomi degli studenti e i loro voti in un esame. Chiedete all'utente del programma se vuole inserire o rimuovere studenti, modificarne il voto o stampare tutti i voti. La visualizzazione dovrebbe essere ordinata per nome e avere un aspetto simile a questo:

```
Carl: B+
Joe: C
Sarah: A
```

3) Scrivete un programma che legga un testo da un file e lo suddivida in singole parole. Inserite le parole in un **Set** realizzato mediante un **TreeSet**. Dopo aver letto tutti i dati, visualizzate tutte le parole, seguite dalla dimensione dell'insieme risultante. Questo programma determinerà, quindi, quante parole diverse sono presenti in un testo.

4) Leggete da un file tutte le parole presenti e aggiungetele a una mappa le cui chiavi siano le lettere iniziali delle parole e i cui valori siano insiemi contenenti le parole che iniziano con quella stessa lettera. Quindi visualizzate gli insiemi di parole in ordine alfabetico.

5) Scrivete un programma che legga un file di codice sorgente Java e generi un elenco di tutti gli identificatori presenti, visualizzando, accanto a ciascuno di essi, i numeri delle righe in cui compare. Per semplicità considereremo che qualsiasi stringa costituita soltanto da lettere, cifre numeriche e caratteri di sottolineatura sia un identificatore. Dichiarate la variabile `Scanner in` per leggere il file e invocate il metodo `in.useDelimiter("[^A-Za-z0-9_]+")`, in modo che ogni invocazione di `next` restituisca un identificatore.

6) Usate una pila per invertire le parole di una frase. Continuate a leggere parole, aggiungendole alla pila, fin quando non trovate una parola che termina con un punto. A questo punto estraete tutte le parole dalla pila e visualizzatele. Realizzate la pila tramite una **Deque**.

7) Dovete realizzare un elenco di cose da fare (*to do list*). A ciscun compito viene assegnata una priorità, un numero intero da *1* a *9*, e una descrizione. Quando l'utente digita il comando *add* `priorità descrizione` il programma aggiunge una cosa da fare, mentre quando l'utente digita `next` il programma elimina e visualizza la cosa da fare più urgentemente. Il comando *quit* termina il programma. Risolvete il problema usando una **PriorityQueue**.

8) In occasione di manifestazioni particolari, il proprietario di una casa noleggia posti auto nel suo vialetto di casa, che può essere rappresentato da una pila, con il consueto comportamento "*last-in, first-out*". Quando il proprietario di un'automobile se ne va e la sua automobile non è l'ultima, tutte quelle che la bloccano devono essere spostate temporaneamente sulla strada, per poi rientrare nel vialetto. Scrivete un programma che simuli questo comportamento, usando una pila per il vialetto e una per la strada, con numeri interi a rappresentare le targhe delle automobili. Un numero positivo inserisce un'automobile nel vialetto, un numero negativo la fa uscire definitivamente e il numero *0* termina la simulazione. Visualizzate il contenuto del vialetto al termine di ciascuna operazione.