



Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

Programmazione Concorrente e Distribuita I/O: esercizi

Luigi Lavazza
Dipartimento di Scienze Teoriche e Applicate
luigi.lavazza@uninsubria.it



Esercizio 1: TestIOTerminale

- Leggere un numero intero e visualizzarlo a terminale, utilizzando le classi di input/output di `java.io`.
- Classi da utilizzare: `InputStream`, `InputStreamReader`, `BufferedReader`



Estensioni/varianti

- a) Fare in modo che se l'utente non inserisce dei caratteri interpretabili come un numero intero, il programma segnali l'errore e richieda nuovamente l'inserimento del numero.
- b) Fare in modo che il programma richieda e accetti l'inserimento di numeri interi fino a quando l'utente non inserisce «Basta».



Esercizio 2: CopiaFileArgs

- Interpreta il primo argomento sulla linea di comando come nome di un file di testo origine e il secondo argomento sulla linea di comando come nome di un file di testo destinazione.
- Copia il contenuto del file origine nel file destinazione, leggendo e scrivendo un solo carattere alla volta.
- Classi da utilizzare: **FileReader**, **FileWriter**.



Estensioni/varianti

- 1) Riportare esito e diagnostici.
- 2) Riportare numero di caratteri copiati.



Esercizio 3: ListaDir

- Se sulla linea di comando c'è un argomento lo interpreta come il nome di un file o di una directory; se sulla linea di comando non c'è alcun argomento, si assume come argomento la directory corrente.
- Se l'argomento è il nome di un file, stampa a terminale il suo path assoluto e la sua dimensione in byte;
- Se l'argomento è una directory, stampa a terminale il suo contenuto, cioè la lista dei file e delle directory che essa contiene.
- Classi da utilizzare: **File**.



Esercizio 4: Testo (analisi di testo)

- Scrivere una classe **Testo** che modelli un testo letto da un file e che fornisca i seguenti costruttori e metodi:
- **public Testo(File file)**
 - ▶ Costruisce l'istanza dell'oggetto che modella il testo contenuto nel file specificato come argomento. Si assuma che l'argomento sia il riferimento ad un file di testo esistente.
- **public int numeroParole()**
 - ▶ Restituisce il numero di parole che compaiono nel testo modellato dall'oggetto che esegue il metodo.
- **public int numeroParoleDistinte()**
 - ▶ Restituisce il numero di parole distinte che compaiono nel testo modellato dall'oggetto che esegue il metodo.
- **public int contaOccorrenzeParola(String daCercare)**
 - ▶ Restituisce il numero di occorrenze della parola specificata come argomento nel testo che esegue il metodo.
- **public LinkedList<String> paroleDistinteInOrdineAlfabetico()**
 - ▶ Restituisce la lista delle parole del testo (senza ripetizioni) in ordine alfabetico.



La classe `java.util.StringTokenizer`

- La classe `StringTokenizer` permette di spezzare una stringa in vari pezzi specificando i caratteri da utilizzare come separatori fra i pezzi.
- `public StringTokenizer(String str, String delim)`
 - ▶ Costruisce lo string-tokenizer per `str` che usa come delimitatori tutti i caratteri nella stringa `delim`.
- `boolean hasMoreTokens()`
 - ▶ Restituisce `true` se lo string-tokenizer che esegue il metodo contiene ancora dei token.
- `String nextToken()`
 - ▶ Restituisce il prossimo token contenuto nello string-tokenizer (dà errore se lo string-tokenizer è già stato svuotato).



Esempio `java.util.StringTokenizer`

```
StringTokenizer stk = new StringTokenizer(riga, " ,.:'\"");

// estraiamo le parole dal testo
while (stk.hasMoreTokens()) {
    // prendiamo la prossima parola
    String nuovaParola = stk.nextToken();
    ...
}
```



La classe `java.util.LinkedList<T>`

```
public class LinkedList<E>  
    extends AbstractSequentialList<E>  
    implements List<E>, Deque<E>, Cloneable, Serializable
```

- Implementa la struttura dati dinamica lista. Gli elementi di tipo T sono mantenuti in una struttura lineare.
- `public LinkedList()`
 - ▶ Costruisce una lista vuota.
- `public boolean add(E e)`
 - ▶ Aggiunge l'elemento specificato in fondo alla lista.
- `int size()`
 - ▶ Restituisce il numero di elementi nella lista.
- `boolean contains(Object e)`
 - ▶ Restituisce `true` se e solo se la lista contiene un elemento o tale che `o.equals(e)`.



La classe `java.util.LinkedList<T>`

- `public E get(int index)`
 - ▶ Restituisce l'elemento in posizione `index`. Il metodo solleva l'eccezione `IndexOutOfBoundsException` se `(index < 0 || index >= size())`.
- `public void add(int index, E element)`
 - ▶ Aggiunge l'elemento specificato nella posizione specificata della lista. Il metodo solleva l'eccezione `IndexOutOfBoundsException` se `(index < 0 || index > size())`.



Esempio `java.util.LinkedList<T>`

```
...  
LinkedList<String> lineeDelTesto =  
    new LinkedList<String>();  
  
String riga;  
while ((riga = ...leggo una riga dal file...) != null)  
    lineeDelTesto.add(riga)  
...
```

Esercizio 5: traduttore

- E' disponibile un file "dizionario" in cui ogni riga contiene due parole:
 - ▶ La prima è un nome in italiano
 - ▶ La seconda è un nome in inglese
- Scrivere un programma che:
 - ▶ Legge una parola
 - ▶ La cerca tra le parole italiane nel dizionario
 - ▶ Restituisce la traduzione in inglese
- Il programma termina quando l'utente inserisce la stringa "<Fine>"