**PROGRAMMAZIONE ORIENTATA AGLI OGGETTI (DM 270-088-12CFU)**

**Appello Straordinario del 27-02-2017 – Allievi Informatici**

|  |  |  |  |
| --- | --- | --- | --- |
| **Cognome Nome:** | **Matr:** | **Email (stampatello):** | **Durata**: **3 ore** |

**Esercizio 1** [4 punti] Si deve scrivere un metodo main Java che (i) legge gli elementi di un vettore v di 10 interi, (ii) inverte il contenuto del vettore, (iii) scrive in uscita il vettore modificato. Derivare sistematicamente e riportare una corrispondente versione RASP del programma Java.

**Esercizio 2** [7 punti] Si deve scrivere un programma per l’individuazione, se esiste, di un percorso di uscita da un labirinto, utilizzando la tecnica backtracking. Il labirinto è composto da camere (), numerate da ad , ognuna delle quali può comunicare mediante porte con altrettante camere vicine. Alcune porte possono essere murate. Il contenuto del labirinto è disponibile su un file di tipo testo. Il file contiene sulla sua prima riga l’intero . Ogni altra linea, dalla seconda in poi, contiene numeri interi non negativi, nell’ordine: il numero di una stanza, i quattro numeri (indici) delle stanze direttamente comunicanti con . Un indice indica che la porta è murata. L’indice denota che la porta conduce all’esterno. Il formato di una cinquina di numeri è il seguente:

dove denota un carattere separatore che può essere uno spazio, un ‘-‘ o un ‘$’.

Il programma legge da tastiera il nome del file testo contenente il labirinto, quindi verifica la corretta composizione del file e si arresta immediatamente con una segnalazione diagnostica, qualora il contenuto sia difforme rispetto alle attese. Diversamente, i dati del labirinto vanno caricati su una opportuna struttura dati. Successivamente il programma legge l’indice di una stanza (*cella di partenza*), cerca una via d’uscita, non necessariamente di lunghezza minima, e la visualizza su output. In alternativa, il programma scrive che nessuna via d’uscita è disponibile.

**Esercizio 3** [13 punti] Negli uffici di una grande azienda, gli impiegati sono sistemati in stanze occupate da una o più persone. Ogni stanza dispone di un solo telefono. Su un file di tipo testo è memorizzata una rubrica dei numeri di telefono. In particolare, ogni linea del file contiene cognome, nome e numero di telefono di un impiegato, separati da uno o più spazi.

Si deve scrivere un programma che legga la rubrica e scriva su output l’elenco degli impiegati stanza per stanza, con il relativo numero di telefono. L’elenco va prodotto per numero di telefono crescente. Gli impiegati di una stessa stanza vanno invece elencati in ordine crescente di cognome e a parità di cognome in ordine crescente di nome.

Ai fini della stesura del programma si suggeriscono le seguenti classi e interfacce (riportare un corrispondente diagramma di classi UML).

Classe contenente cognome, nome e riferimento ad una stanza (si veda più avanti), con opportuni metodi accessori e i metodi canonici toString, equals e hashCode.

Classe contenente un numero di telefono e l’elenco degli impiegati allocati nella stanza, più un insieme opportuno di metodi (inclusi i tre metodi canonici).

Interfaccia generica , iterabile, con un insieme minimo di metodi utili ai fini del problema tra cui:

*int size(), void clear(), void add(T elem), void remove(T elem), boolean contains(T elem), T get( T elem ), Comparator<T> getComparator(), void setComparatorAndSort(Comparator<T> c)*

Anche se il tipo T degli elementi non garantisce la disponibilità del confronto naturale degli elementi, un elenco va sempre mantenuto ordinato secondo un comparatore stabilito con un’invocazione del metodo .

Classe astratta che implementa e concretizza quanti più metodi è possibile e certamente i metodi , e .

Classe concreta erede di , che memorizza gli elementi su una lista concatenata semplice a puntatori espliciti (un solo puntatore per nodo). Il costruttore riceve un comparatore da utilizzare per i confronti.

Classe concreta erede di , che memorizza gli elementi su una struttura dati collezione non della libreria di Java. Il costruttore deve garantire la definizione di un opportuno oggetto comparatore.

Classe col metodo main che legge preliminarmente il nome del file testo con la rubrica telefonica, e quindi realizza le operazioni del programma, introducendo gli opportuni comparatori.

**Esempio 4** [6 punti] Con riferimento alla classe studiata a lezione (di cui si chiede di riportare l’intestazione e le principali dichiarazioni interne), aggiungere i seguenti metodi (eventualmente suddivisi in una versione pubblica ed una privata ricorsiva):

Mentre il significato dei primi tre metodi dovrebbe essere auto-esplicativo, il metodo visita in modo depth-first l’albero e memorizza sulla lista ls le informazioni dei soli nodi foglia, nell’ordine da destra verso sinistra.