Esame di Fondamenti di Informatica L-B
Ingegneria Gestionale e dei Processi Gestionali (L-Z)

Appello del 18/4/2008

Esercizio 1 (4 punti)

Definizione ed uso dei *generics* nelle collezioni Java.

Esercizio 2 (6 punti)

Siano date le seguenti funzioni C:

```
int f(int V[], int M) {
  int i, sum=0;
  for(i=0; i<M; i++)
    sum+=V[++i];
  return sum;
}

int g(int V[], int N) {
  int j=0, res=1;
  while(j<N) {
    res*=f(V, j);
    j+=2;
  }
  return res;
}
```

1. Calcolare la complessità in passi base della funzione `f` nei termini del parametro `M` (suggerimento: si supponga `M` pari).
2. Calcolare la complessità in passi base della funzione `g` nei termini del parametro `N` (suggerimento: si supponga `N` pari e si esprima `j`=2·`i`).
3. Calcolare la complessità asintotica della funzione `g` nei termini del parametro `N`.

Esercizio 3 (5 punti)

La stazione ferroviaria del paese di Collate (Parma) intende informatizzare la gestione dei dati relativi ai (pochi) treni che fanno scalo nella stazione. Le informazioni relative ad ogni treno comprendono le stazioni di partenza ed arrivo e l'orario di arrivo alla stazione di Collate (Parma) (ogni treno riparte dopo pochi secondi di sosta in stazione). Poiché la stazione possiede solamente una banchina, si supponga che due treni non possano mai sostare contemporaneamente in stazione.
Si scriva una classe `Treno` per la stazione di Collate (Parma) che:

1. Possieda un opportuno costruttore con parametri.
2. Presenti opportuni metodi che permettano di accedere alle variabili di istanza della classe.
3. Presenti il metodo `toString` che fornisca una descrizione del treno.
4. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Treno` (l'uguaglianza va verificata sulle stazioni di partenza ed arrivo).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un treno passato come parametro (la precedenza va verificata sull'orario di arrivo).

Esercizio 4 (9 punti)

Si scriva una classe `Stazione` che memorizzi le informazioni relative ai treni che fanno scalo in stazione. In particolare, i treni vanno memorizzati in una lista.
La classe `Stazione` deve inoltre:

1. Possedere un opportuno costruttore (la lista di treni va passata come parametro).
2. Presentare un metodo `trovaTreno` che, data la stazione di arrivo e l'orario di partenza, restituisca il primo oggetto `Treno` disponibile verso la località specificata nella data odierna, se un tale treno esiste (si supponga che i treni siano ordinati secondo il punto 5. dell'esercizio 3).
3. Possedere un metodo `aggiungiTreno` che, dato un oggetto `Treno`, provveda ad inserirlo nella lista, mantenendo l'ordinamento sull'orario (suggerimento: si utilizzi il metodo `add(int i, Treno t)` della classe `List`).
4. Presentare il metodo `toString` che restituisca una stringa che fornisca una descrizione dei treni in transito nella stazione.
5. Possedere il metodo `quantiTreni` che, data una stazione di arrivo, restituisca il numero di treni in partenza verso tale stazione.

Esercizio 5 (8 punti)

Si scriva un'applicazione per la stazione di Collate (Parma):

1. Crei un oggetto `Stazione` (inizialmente la stazione non deve contenere alcun treno).
2. Legga da tastiera le informazioni relative ad alcuni treni e le inserisca all'interno della stazione.
3. Letta da tastiera la richiesta relativa ad un viaggiatore (stazione di arrivo ed orario di partenza), provveda a stampare le informazioni relative al primo treno che soddisfi le specifiche.
4. Letti da tastiera i nomi di alcune stazioni, stampi su schermo il nome della stazione che, tra questi, possiede più treni in arrivo.

Per la lettura di dati da tastiera è possibile utilizzare l'oggetto `Lettore.in`, definito all'interno del package `fiji.io`, che possiede i seguenti metodi:

- `boolean leggiBoolean()` Legge un boolean (delimitato da spazi).
- `char leggiChar()` Legge un singolo carattere.
- `double leggiDouble()` Legge un numero razionale (delimitato da spazi).
- `float leggiFloat()` Legge un numero razionale (delimitato da spazi).
- `int leggiInt()` Legge un intero (delimitato da spazi).
- `String leggiLinea()` Legge una linea di testo.
- `String leggiString()` Legge una parola senza spazi al suo interno.

Soluzione Esercizio 2

Domanda 1:

2 assegnamenti	2
`for`	`M`/2 + 1
`sum+=V[++i]`	`M`/2
`i++`	`M`/2
Totale	3 + 3`M`/2

Domanda 2:

2 assegnamenti	2
`while`	`N`/2 + 1
chiamata di `f`	`N`/2
complessità di `f`	3`N`/4 + 3`N`^{2}/8
`j+=2`	`N`/2
Totale	3`N`^{2}/8 + 9`N`/4 + 3

Domanda 3:
Complessità asintotica: $O(\mathtt{N}^2)$

Soluzione Esercizio 3

```
class Treno implements Comparable<Treno> {
  private String partenza, arrivo;
  private int ore, minuti;

  public Treno(String partenza, String arrivo, int ore, int minuti) {
    this.partenza=partenza;
    this.arrivo=arrivo;
    this.ore=ore;
    this.minuti=minuti;
  }

  public String getPartenza() { return partenza; }
  public String getArrivo() { return arrivo; }
  public int getOre() { return ore; }
  public int getMinuti() { return minuti; }

  public String toString() {
    return partenza+"->"+arrivo+", "+ore+":"+minuti;
  }

  public int compareTo(Treno t) {
    return (this.ore*60+this.minuti-t.ore*60+t.minuti);
  }
  public boolean equals(Object o) { return equals((Treno) o); }
  public boolean equals(Treno t) {
    return (partenza.equals(t.partenza)&&arrivo.equals(t.arrivo));
  }
}
```

Soluzione Esercizio 4

```
import java.util.*;
class Stazione {
  private List<Treno> treni;

  public Stazione(List<Treno> l) {
    this.treni=new ArrayList<Treno>();
    for(Treno t:l) this.treni.add(t);
  }

  public Treno trovaTreno(String arrivo, int ore, int minuti) {
    for(Treno t:treni) {
      if(t.getArrivo().equals(arrivo)&&
        (ore*60+minuti)<=(t.getOre()*60+t.getMinuti()))
          return t;
    }
    return null;
  }

  public void aggiungiTreno(Treno treno) {
    int i=0;
    while(i<treni.size()&&treni.get(i).compareTo(treno)<=0) i++;
    treni.add(i, treno);
  }

  public String toString() { return treni.toString(); }

  public int quantiTreni(String arrivo) {
    int n=0;
    for(Treno t:treni) if(t.getArrivo().equals(arrivo)) n++;
    return n;
  }
}
```

Soluzione Esercizio 5

```
import fiji.io.*;
import java.util.*;
class Applicazione {
  public static void main(String[] args) {
    Stazione s=new Stazione(new ArrayList<Treno>()); // domanda 1
    int i, N=Lettore.in.leggiInt(), max=0;

    for(i=0; i<N; i++) {
      Treno t=new Treno(Lettore.in.leggiLinea(),
        Lettore.in.leggiLinea(), Lettore.in.leggiInt(),
        Lettore.in.leggiInt());
      s.aggiungiTreno(t);
    } // domanda 2
    Treno t= s.trovaTreno(Lettore.in.leggiLinea(),
      Lettore.in.leggiInt(), Lettore.in.leggiInt());
    if(t==null) System.out.println("Nessun treno esistente");
    else System.out.println(t);
    String stazione="";
    N=Lettore.in.leggiInt();
    for(i=0; i<N; i++) {
      String arrivo=Lettore.in.leggiLinea();
      int quanti=s.quantiTreni(arrivo);
      if(quanti>max) {
        max=quanti;
        stazione=arrivo;
      }
    }
    System.out.println(stazione);
  }
}
```