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

Appello del 22/2/2011

Esercizio 1 (4 punti)

Gestione ed utilizzo di liste in Java.

Esercizio 2 (6 punti)

Siano dati i seguenti metodi Java:

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

public static int g(int V[],int N) {
  int j, sum=0;
  for(j=0; j<=N-1; )
    sum+=f(V, N, j++);
  return sum;
}
```

1. Calcolare la complessità in passi base del metodo `f` nei termini dei parametri `M` e `N` (si distinguano i casi in cui `M` assume valori minori di `N` da quelli in cui assume valori maggiori o uguali a `N`).
2. Calcolare la complessità in passi base del metodo `g` nei termini del parametro `N`.
3. Calcolare la complessità asintotica del metodo `g` nei termini del parametro `N`.

Esercizio 3 (5 punti)

Lo studio dentistico "Carie ed Eventuali", aperto recentemente, desidera essere tecnologicamente all'avanguardia in tutti gli aspetti del proprio business, compreso il sistema informativo; per questo desidera informatizzare la gestione dei propri pazienti. A tal scopo, ogni visita eseguita all'interno dello studio viene catalogata in un computer, memorizzandone il cognome del paziente (si supponga non esistano omonimie tra i pazienti), l'intervento eseguito (ad es., estrazione, detartrasi, otturazione) e la data della visita stessa. Si scriva una classe `Visita` per lo studio dentistico "Carie ed Eventuali", che:

1. Possieda un opportuno costruttore con parametri.
2. Presenti opportuni metodi che permettano di accedere alle variabili di istanza dell'oggetto.
3. Presenti il metodo `toString` che fornisca la descrizione della visita.
4. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Visita` (l'uguaglianza va verificata sulla data e sul cognome del paziente).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Visita` passato come parametro (la precedenza va data alle visite più recenti e, in caso di parità, si procede per ordinamento alfabetico sul cognome del paziente).

Esercizio 4 (8 punti)

Si scriva una classe `Dottore` che memorizzi le informazioni relative alle visite tenute da ciascun medico dello studio. Oltre memorizzare il cognome del medico, le visite vanno inserite all'interno di una lista. La classe `Dottore` deve inoltre:

1. Presentare un opportuno costruttore (inizialmente un medico non ha sostenuto alcuna visita).
2. Presentare opportuni metodi che permettano di accedere alle variabili di istanza dell'oggetto.
3. Possedere il metodo `toString` che fornisca la descrizione del medico (inclusa la descrizione di tutte le visite sostenute).
4. Possedere il metodo `aggiungi` che, dato un oggetto `Visita`, lo inserisca all'interno della lista, mantenendo tale lista ordinata secondo il punto 5. dell'esercizio 3 (suggerimento: si utilizzi il metodo `add(int i, Visita v)` della classe `List<Visita>`).
5. Presentare il metodo `cerca` che, dato il cognome di un paziente, restituisca un insieme contenente tutte le visite che il medico ha sostenuto con tale paziente.

Esercizio 5 (7 punti)

Si scriva un'applicazione per lo studio dentistico "Carie ed Eventuali"che:

1. Crei un insieme di oggetti `Dottore`.
2. Crei un oggetto `Dottore`, lette da tastiera le informazioni necessarie.
3. Inserisca l'oggetto di cui al punto 2. all'interno dell'insieme di cui al punto 1, verificando che tale insieme non contenga già un oggetto uguale.
4. Crei un oggetto `Visita`, lette da tastiera le informazioni necessarie, e lo aggiunga alle visite tenute dal medico di cui al punto 2.
5. Letto da tastiera il cognome di un paziente, stampi a video la data di tutti gli interventi di tipo "otturazione" sostenuti su tale paziente dal medico di cui al punto 2.
6. Stampi a video la descrizione di tutte le visite sostenute sul paziente di cui al punto 5. da un qualsiasi medico dello studio.

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:

- `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	o 2
`i<M-N`	`M-N+1`	o `1`
`sum+=V[i]`	`M-N`	o `0`
`i++`	`M-N`	o `0`
Totale	`3M - 3N + 3`	o 3

Domanda 2:

2 assegnamenti	2
`j<=N-1`	`N+1`
`sum+=f(V, N, j++)`	`N`
complessità di `f`	$3N^2/2 + 9N/2$
Totale	$3N^2/2 + 13N/2 + 3$

complessità di `f`: $\sum_{j=0}^{N-1}(3N-3j+3) = 3N^2 - \frac{3}{2}N(N-1) + 3N = \frac{3}{2}N^2 + \frac{9}{2}N$

Domanda 3:

Complessità asintotica: $O(N^2)$

Soluzione Esercizio 3

```
class Visita implements Comparable<Visita> {
  private String paziente, intervento;
  private int g, m, a;

  public Visita(String paziente, String intervento, int g, int m, int a) {
    this.paziente=paziente;
    this.intervento=intervento;
    this.g=g; this.m=m; this.a=a;
  }

  public String getPaziente() { return paziente; }
  public String getIntervento() { return intervento; }
  public String getData() { return ""+g+"/"+m+""+a; }

  public String toString() {
    return paziente + ": " + intervento + "(" + getData() + ")";
  }

  public boolean equals(Object o) { return equals((Visita) o); }
  public boolean equals(Visita v) {
    return paziente.equals(v.paziente)&&(g==v.g)&&(m==v.m)&&(a==v.a);
  }

  public int compareTo(Visita v) {
    int ret=v.a-this.a;
    if(ret==0) ret=v.m-this.m;
    if(ret==0) ret=v.g-this.g;
    if(ret==0) ret=this.paziente.compareTo(v.paziente);
    return ret;
  }
}
```

Soluzione Esercizio 4

```
import java.util.*;

class Dottore {
  private String nome;
  private List<Visita> visite;

  public Dottore(String nome) {
    this.nome=nome;
    visite =new ArrayList<Visita>();
  }

  public String getNome() { return nome; }

  public String toString() {
    return nome + ":" + visite.toString();
  }

  public void aggiungi(Visita v) {
    int j=0;
    while((j< visite.size())&&(visite.get(j).compareTo(v)<0)) j++;
    visite.add(j, v);
  }

  public Set<Visita> cerca(String nome) {
    Set<Visita> s=new TreeSet<Visita>();
    for(Visita v:visite)
      if(v.getPaziente().equals(nome)) s.add(v);
    return s;
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;
import fiji.io.*;

class Applicazione {
  public static void main(String[] args) {
    Set<Dottore> s=new HashSet<Dottore>(); // domanda 1
    Dottore d=new Dottore(Lettore.in.leggiLinea()); // domanda 2
    if(!s.add(d)) System.out.println("Dottore già esistente!");// domanda 3
    d.aggiungi(new Visita(Lettore.in.leggiLinea(), Lettore.in.leggiLinea(),
      Lettore.in.leggiInt(), Lettore.in.leggiInt(),
      Lettore.in.leggiInt())); // domanda 4
    String nome= Lettore.in.leggiLinea();
    Set<Visita> visite=d.cerca(nome);
    for(Visita v:visite) // domanda 5
      if(v.getIntervento().equals("otturazione"))
        System.out.println(v.getData());
    for(Dottore doc:s) // domanda 6
      System.out.println(doc.cerca(nome));
  }
}
```