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

Appello del 10/7/2015

Esercizio 1 (4 punti)

Definire il concetto di ricorsione.

Esercizio 2 (6 punti)

Siano dati i seguenti metodi Java:

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

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

1. Calcolare la complessità in passi base del metodo `f` nei termini del parametro `N` (si distinguano i casi in cui `N` assume valori pari da quelli in cui assume valori dispari).
2. Calcolare la complessità in passi base del metodo `g` nei termini del parametro `N` (si supponga `N` dispari).
3. Calcolare la complessità asintotica del metodo `g` nei termini del parametro `N`.

Esercizio 3 (5 punti)

In previsione delle prossime elezioni, il rettore dell'Università Statale "La Sorbolona" intende informatizzare le procedure di voto. A tal scopo, per ogni scheda elettorale vanno memorizzati la matricola del votante (le elezioni, infatti, non sono anonime), il cognome del candidato votato e la data di immissione della scheda nell'urna. Si scriva una classe `Scheda` per l'Università Statale "La Sorbolona" che:

1. Possieda un opportuno costruttore con parametri.
2. Presenti opportuni metodi che permettano di accedere alle variabili d'istanza dell'oggetto.
3. Presenti il metodo `toString` che fornisca una descrizione della scheda.
4. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Scheda` (la verifica va fatta unicamente sulla matricola del votante).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Scheda` passato come parametro (in ordine decrescente di data e, a parità, in ordine crescente di matricola del votante).

Esercizio 4 (8 punti)

Si scriva una classe `Seggio` che memorizzi le informazioni riguardanti le schede che sono state compilate in ciascun seggio. Per ciascun seggio occorre memorizzare il numero e l'indirizzo, mentre le schede vanno memorizzate all'interno di una lista. La classe `Seggio` deve:

1. Presentare un opportuno costruttore con parametri (inizialmente, la lista di schede è vuota).
2. Possedere opportuni metodi che permettano di accedere alle variabili d'istanza dell'oggetto.
3. Presentare il metodo `toString` che fornisca la descrizione del seggio (inclusa la descrizione di tutte le schede ivi compilate).
4. Possedere il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Seggio` (la verifica va effettuata su numero di seggio e indirizzo).
5. Presentare il metodo `aggiungi` che, dato un oggetto `Scheda`, lo inserisca all'interno della lista, mantenendo quest'ultima ordinata secondo il punto 5. dell'Esercizio 3.
6. Possedere il metodo `votato` che, dato il nome di un candidato, restituisca il numero di schede che contengono tale nome come candidato votato.
7. Presentare il metodo `giàVotato` che, dato un numero di matricola, indichi se una scheda con tale numero è stata già compilata al seggio.
8. Possedere il metodo `quante` che indichi il numero totale di schede compilate all'interno del seggio.

Esercizio 5 (8 punti)

Si scriva un'applicazione per l'Università Statale "La Sorbolona" che:

1. Crei un insieme di oggetti `Seggio`.
2. Crei un oggetto `Seggio`, lette da tastiera le informazioni necessarie.
3. Inserisca l'oggetto di cui al punto 2. all'interno dell'insieme di cui al punto 1, controllando che tale inserimento sia possibile.
4. Crei un oggetto `Scheda`, lette da tastiera le informazioni necessarie.
5. Controlli che la `Scheda` creata al punto 4. non sia già stata compilata al seggio di cui al punto 2. In caso negativo, la inserisca tra quelle compilate in tale seggio.
6. Dato un insieme di nomi di candidati (che si supponga di avere già creato e riempito), calcoli e stampi, per ciascun elemento di tale insieme, il numero totale di voti ottenuti in tutti i seggi dell'insieme di cui al punto 1.
7. Stampi il numero totale di schede compilate per la votazione.

Soluzione Esercizio 2

Domanda 1:

2 assegnamenti	2	o 2
`sum+=V[++i]`	$N/2 + 1$	o $(N + 1)/2$
`i++<N`	$N/2 + 1$	o $(N + 1)/2$
Totale	$N + 4$	o $N + 3$

Domanda 2:

2 assegnamenti	2
`j>0`	$N + 1$
`sum+=f(V, --j)`	N
complessità di `f`	$N^2/2 + 4N - 1/2$
Totale	$N^2/2 + 6N + 5/2$

complessità di `f`: $\sum_{\substack{j=1 \\ j \text{ pari}}}^{N} (j+4) + \sum_{\substack{j=1 \\ j \text{ dispari}}}^{N} (j+3) = \sum_{j=1}^{N}(j+3) + \sum_{\substack{j=1 \\ j \text{ pari}}}^{N} 1 = \frac{N^2}{2} + 4N - \frac{1}{2}$

Domanda 3:
Complessità asintotica: $O(N^2)$

Soluzione Esercizio 3

```
class Scheda implements Comparable<Scheda> {
  private String matricola, candidato;
  private int g, m, a;

  public Scheda (String matricola, String candidato, int g, int m, int a) {
    this.matricola = matricola;
    this.candidato = candidato;
    this.g = g; this.m = m; this.a = a;
  }

  public String getMatricola() { return matricola; }
  public String getCandidato() { return candidato; }
  public String getData() { return g+"/"+m+"/"+a; }

  public String toString() {
    return matricola + ", " + getData()+ " (" + candidato+ ")";
  }

  public boolean equals(Object o) { return equals((Scheda) o); }
  public boolean equals(Scheda s) {
    return this.matricola.equals(s.matricola);
  }

  public int compareTo(Scheda s) {
    int ret = s.a - this.a;
    if(ret==0) ret = s.m - this.m;
    if(ret==0) ret = s.g - this.g;
    if(ret==0) ret = this.matricola.compareTo(s.matricola);
    return ret;
  }
}
```

Soluzione Esercizio 4

```
import java.util.*;
class Seggio {
  private List<Scheda> l;
  private int numero;
  private String indirizzo;

  public Seggio (int numero, String indirizzo) {
    this.numero = numero;
    this.indirizzo = indirizzo;
    l = new ArrayList<Scheda>();
  }
  public int getNumero() { return numero; }
  public String getIndirizzo() { return indirizzo; }
  public String toString() { return numero + ", " + indirizzo + "):" + l; }
  public boolean equals(Object o) { return equals((Seggio) o); }
  public boolean equals(Seggio s) {
    return this.numero == s.numero && this.indirizzo.equals(s.indirizzo);
  }
  public int quante() { return l.size(); }
  public void aggiungi(Scheda s) {
    int i = 0;
    while((i<l.size())&&(l.get(i).compareTo(s)<0)) i++;
    l.add(i, s);
  }
  public int votato(String candidato) {
    int n=0;
    for(Scheda s: l) if(s.getCandidato().equals(candidato)) n++;
    return n;
  }
  public boolean giàVotato(String matricola) {
    for(Scheda s: l) if(s.getMatricola().equals(matricola)) return true;
    return false;
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;
import fiji.io.*;
class Applicazione {
  public static void main(String[] args) {
    Set<Seggio> s = new TreeSet<Seggio>();
    Seggio e = new Seggio(Lettore.in.leggiInt(), Lettore.in.leggiLinea());
    if(!s.add(e)) System.out.println("Seggio già esistente!");
    Scheda c = new Scheda(Lettore.in.leggiLinea(), Lettore.in.leggiLinea(),
      Lettore.in.leggiInt(), Lettore.in.leggiInt(), Lettore.in.leggiInt());
    if(!e.giàVotato(c.getMatricola())) e.aggiungi(c);
    Set<String> candidati = new HashSet<String>();
    for(String cand: candidati) {
      int n=0;
      for(Seggio x: s) n+=x.votato(cand);
      System.out.println(cand + ": " + n);
    }
    int totale = 0;
    for(Seggio x: s) totale+=x.quante();
    System.out.println("Totale schede: " + totale);
  }
}
```