Esame di Fondamenti di Informatica T-A
Ingegneria Gestionale (A-K)

Appello del 11/2/2010

NOTA: Per il superamento dell'esame è **necessario** ottenere la sufficienza nello svolgimento dell'Esercizio 1.

Esercizio 1 (4 punti)

1. Descrivere le principali differenze tra vettori e collezioni in Java.
2. Descrivere la codifica dei numeri reali all'interno di un elaboratore elettronico.

Esercizio 2 (2 punti)

Rappresentare in binario il numero **–0,326** supponendo di utilizzare 8 bit per la mantissa (in modulo e segno) ed 8 bit per l'esponente (in complemento a 2).

Esercizio 3 (5 punti)

Siano dati i seguenti metodi Java:

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

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

1. Calcolare la complessità in passi base del metodo `f` nei termini dei parametri `M` ed `N` (si distingua il caso in cui `M` assume valori minori o uguali a `N` da quello in cui assume valori maggiori di `N`).
2. Calcolare la complessità in passi base del metodo `g` nei termini dei parametri `M` ed `N` (si supponga `N`≤`M`).
3. Calcolare la complessità asintotica del metodo `g` nei termini dei parametri `M` ed `N`.

Esercizio 4 (6 punti)

La federazione scacchistica dello stato caraibico di St. Marquez (SMFC, St. Marquez Federación de Chess) ha deciso di informatizzare la gestione dei propri tornei di scacchi. In particolare, per ogni giocatore viene registrato il nome, l'anno di nascita ed il punteggio totale associato al giocatore (un intero non negativo). Si scriva una classe `Giocatore` per la federazione SMFC che:

1. Possieda un opportuno costruttore con parametri (il punteggio iniziale di un giocatore è nullo).
2. Presenti opportuni metodi che permettano di accedere alle variabili di istanza dell'oggetto.
3. Possieda il metodo `aggiornaPunti` che, dato un valore intero, provveda ad aggiornare opportunamente il punteggio (che non deve mai essere negativo).
4. Presenti il metodo `toString` che fornisca la descrizione del giocatore.
5. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Giocatore` (l'uguaglianza va verificata unicamente sul nome del giocatore).
6. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Giocatore` passato come parametro (ha precedenza il giocatore con punteggio maggiore e, in caso di parità, si procede per ordine alfabetico).

Esercizio 5 (7 punti)

Si scriva una classe `Partita` che memorizzi le informazioni relative ad una partita tra due giocatori. Oltre ai giocatori coinvolti, occorre memorizzare la data ed il luogo dell'incontro. La classe `Partita` deve inoltre:

1. Presentare un opportuno costruttore.
2. Presentare opportuni metodi che permettano di accedere alle variabili di istanza dell'oggetto.
3. Possedere il metodo `toString` che fornisca la descrizione della partita (dei partecipanti occorre indicare solamente il nome).
4. Presentare il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Partita` (l'uguaglianza va verificata sulla data e sull'identicità dei due giocatori).
5. Possedere il metodo `vincitore` che, dato il nome del giocatore vincitore della partita aggiorni il punteggio dando 1 punto al vincitore e togliendo 1 punto al perdente; la parità viene indicata non fornendo alcun oggetto al metodo: in tal caso non si deve aggiornare il punteggio.

Esercizio 6 (7 punti)

Si scriva un'applicazione per la federazione SMFC che:

1. Crei una lista di oggetti `Giocatore`.
2. Crei un oggetto `Giocatore`, lette da tastiera le informazioni necessarie.
3. Inserisca l'oggetto di cui al punto 2. all'interno della lista di cui al punto 1., mantenendo tale lista ordinata secondo il punto 6. dell'esercizio 4. (suggerimento: si utilizzi il metodo `add(int i, Giocatore g)` della classe `List<Giocatore>`).
4. Crei una `Partita` tra il giocatore di cui al punto 2. e il primo giocatore della lista di cui al punto 3., lette da tastiera le informazioni necessarie.
5. Supponendo che il giocatore di cui al punto 2. vinca la partita appena creata, provveda ad aggiornare opportunamente i punteggi dei giocatori.
6. Stampi le informazioni di tutti i giocatori presenti nella lista di cui al punto 1.

Soluzione Esercizio 2

$-0{,}326_{10} = -0{,}01010011_2$ quindi la mantissa è **(1)1010011** e l'esponente -1_{10} = **11111111**.

Soluzione Esercizio 3

Domanda 1:

2 assegnamenti	2	
`while(i<=N)`	N – M + 2	o 1
`sum+=V[i++]`	N – M + 1	o 0
Totale	2 N – 2 M + 5	o 3

Domanda 2:

2 assegnamenti	2
`j<N`	N + 1
`sum+=f(V, j, M)`	N
`j++`	N
complessità di `f`	$2\,MN - N^2 + 6\,N$
Totale	$2\,MN - N^2 + 9\,N + 3$

complessità di `f`: $\sum_{j=0}^{N-1}(2M - 2j + 5) = 2MN - \frac{2}{2}N(N-1) + 5N$

Domanda 3:

Complessità asintotica: O(MN)

Soluzione Esercizio 4

```
class Giocatore implements Comparable<Giocatore> {
  private String nome;
  private int anno, punti;

  public Giocatore(String nome, int anno) {
    this.nome=nome;
    this.anno=anno;
    this.punti=0;
  }

  public String getNome() { return nome; }
  public int getAnno() { return anno; }
  public int getPunti() { return punti; }
  public void aggiornaPunti(int punti) {
    this.punti+=punti;
    if(this.punti<0) this.punti=0;
  }

  public String toString() {
    return nome + "(" + anno + "): " + punti;
  }

  public boolean equals(Object o) { return equals((Giocatore) o); }
  public boolean equals(Giocatore g) {
    return nome.equals(g.nome);
  }

  public int compareTo(Giocatore g) {
    int ret=g.punti-this.punti;
    if(ret==0) ret=this.nome.compareTo(g.nome);
    return ret;
  }
}
```

Soluzione Esercizio 5

```
class Partita {
  private Giocatore g1, g2;
  private String luogo;
  private int g, m, a;

  public Partita(Giocatore g1, Giocatore g2, String luogo, int g, int m,
      int a) {
    this.g=g; this.m=m; this.a=a;
    this.luogo=luogo;
    this.g1=g1; this.g2=g2;
  }

  public String getLuogo () { return luogo; }
  public String getData() { return ""+g+"/"+m+"/"+a; }

  public String toString() {
    return luogo+"("+getData()+"): "+g1.getNome()+"-"+g2.getNome();
  }

  public boolean equals(Object o) { return equals((Partita) o); }
  public boolean equals(Partita p) {
    return (getData().equals(p.getData())&&g1==p.g1&&g2==p.g2);
  }

  public void vincitore(String nome) {
    if(nome==null) return;
    if(g1.getNome().equals(nome)) {
      g1.aggiornaPunti(1);
      g2.aggiornaPunti(-1);
    }
    else if(g2.getNome().equals(nome)) {
      g2.aggiornaPunti(1);
      g1.aggiornaPunti(-1);
    }
  }
}
```

Soluzione Esercizio 6

```
import java.util.*;
class Applicazione {
  public static void main(String[] args) {
    List<Giocatore> l=new ArrayList<Giocatore>(); // domanda 1
    Scanner scanner=new Scanner(System.in);
    Giocatore g=new Giocatore(scanner.next(), scanner.nextInt());
    int i=0;
    while((i<l.size())&&(l.get(i).compareTo(g)<0)) i++;
    l.add(i, g); // domanda 3
    Partita p=new Partita(g, l.get(0), scanner.next(), scanner.nextInt(),
      scanner.nextInt(), scanner.nextInt());// domanda 4
    p.vincitore(g.getNome()); // domanda 5
    System.out.println(l); // domanda 6
  }
}
```