Esame di Fondamenti di Informatica T-1/T-A
Ingegneria Gestionale (A-K)

Appello del 18/2/2013

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

Esercizio 1 (4 punti)

1. Descrivere l'architettura egli elaboratori secondo Von Neumann.
2. Definire il concetto di ricorsione.

Esercizio 2 (2 punti)

Rappresentare in binario il numero **51,375** supponendo di utilizzare 8 bit per la mantissa (in modulo e segno) ed 8 bit per l'esponente (in complemento a 2). Si motivino infine eventuali differenze tra il numero originale e quello rappresentato.

Esercizio 3 (5 punti)

Siano dati i seguenti metodi Java:

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

public static int g(int V[], int N) {
  int sum=0;
  for(int j=0; ++j<N; )
    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 4 (6 punti)

La multinazionale "D.U.M. Business" sta per avviare una massiccia campagna di vendita di frigoriferi al popolo lappone ed ha necessità di tracciare l'attività dei propri venditori. Ogni venditore è caratterizzato dal nome (lo si ipotizzi senza spazi), dal numero di giorni di servizio (almeno 1) e dal numero di frigoriferi venduti. Si scriva una classe `Venditore` per la "D.U.M. Business" che:

1. Possieda un opportuno costruttore con parametri.
2. Presenti opportuni metodi che permettano di accedere alle variabili di istanza dell'oggetto.
3. Possieda il metodo `aggiungiVenduti` che incrementi il numero di frigoriferi venduti di un valore passato come parametro.
4. Presenti il metodo `toString` che fornisca una descrizione del venditore.
5. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Venditore` (la verifica va fatta sul nome ed i giorni di servizio).
6. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Venditore` passato come parametro (in ordine crescente rispetto a numero di frigoriferi venduti).

Esercizio 5 (6 punti)

Si scriva una classe `Divisione` che memorizzi le informazioni relative ad una serie di venditori della "D.U.M. Business". Per ogni divisione si memorizzi inoltre la sede operativa (es. "Nord-Trøndelag"). La classe `Divisione` deve:

1. Presentare un opportuno costruttore con parametri (inizialmente, una divisione non ha venditori).
2. Possedere opportuni metodi che permettano di accedere alle variabili di istanza dell'oggetto.
3. Presentare il metodo `toString` che fornisca la descrizione della divisione (inclusa la descrizione di tutti i suoi venditori).
4. Possedere il metodo `aggiungi` che, dato un oggetto `Venditore`, lo aggiunga a quelli gestiti dalla divisione (si noti che è possibile avere duplicati).
5. Possedere il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Divisione` (la verifica va fatta sulla sede operativa).
6. Possedere il metodo `totale` che restituisca il totale di frigoriferi venduti dai venditori della divisione.
7. Possedere il metodo `migliore` che restituisca il venditore con il maggior numero di frigoriferi venduti.

Esercizio 6 (7 punti)

Si scriva un'applicazione per l'agenzia "D.U.M. Business" che:

1. Crei un insieme di oggetti `Divisione`.
2. Crei un oggetto `Divisione`, 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 l'inserimento sia possibile.
4. Crei un oggetto `Venditore`, lette da tastiera le informazioni necessarie.
5. Inserisca il venditore creato al punto 4. tra quelli associati alla divisione di cui al punto 2.
6. Stampi a video i dati del miglior venditore di tutte le divisioni.
7. Stampi a video il numero totale di frigoriferi venduti da tutte le divisioni.

Soluzione Esercizio 2

$51{,}375_{10} = 110011{,}011_2$ quindi la mantissa è **(0)1100110**, l'esponente 6_{10} = **00000110**.
Il numero rappresentato è 51, diverso dal numero originale in quanto quest'ultimo ha un numero di cifre significative maggiore di 7.

Soluzione Esercizio 3

Domanda 1:

2 assegnamenti	2	o 2
`++i<N`	N/2 + 1	o N/2 + 1/2
`sum+=V[i++]`	N/2	o N/2 – 1/2
Totale	N +3	o N +2

Domanda 2:

2 assegnamenti	2
`++j<N`	N
`sum+=f(V, j)`	N – 1
complessità di `f`	$N^2/2 + 2N - 5/2$
Totale	$N^2/2 + 4N - 3/2$

complessità di `f`: $\sum_{\substack{j=1 \\ (j \text{ pari})}}^{N-1} (j+3) + \sum_{\substack{j=1 \\ (j \text{ dispari})}}^{N-1} (j+2) = \sum_{j=1}^{N-1}(j+2) + \frac{(N-1)}{2} = \frac{N^2}{2} + 2N - \frac{5}{2}$

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

Soluzione Esercizio 4

```
class Venditore implements Comparable<Venditore> {
  private String nome;
  private int giorni;
  private int venduti;

  public Venditore(String nome, int giorni, int venduti) {
    this.nome = nome;
    this.giorni = giorni;
    this.venduti = venduti;
  }

  public String getNome() { return nome; }
  public int getGiorni() { return giorni; }
  public int getVenduti() { return venduti; }

  public void aggiungiVenduti(int quanti) { venduti += quanti; }

  public String toString() {
    return nome + " per " + giorni + ":" + venduti;
  }

  public boolean equals(Object o) { return equals((Venditore) o); }
  public boolean equals(Venditore v) {
    return nome.equals(v.nome) && giorni == v.giorni;
  }

  public int compareTo(Venditore v) {
    return venduti - v.venduti;
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;

class Divisione {
  private String sede;
  private List<Venditore> l;

  public Divisione(String sede) {
    this.sede = sede;
    l=new ArrayList<Venditore>();
  }

  public String getSede() { return sede; }

  public String toString() {
    return "Divisione di " + sede + ": " + l.toString();
  }

  public void aggiungi(Venditore v) { l.add(v); }

  public boolean equals(Object o) { return equals((Divisione) o); }
  public boolean equals(Divisione d) { return sede.equals(d.sede); }

  public int totale() {
    int tot = 0;
    for (Venditore v : l)
      tot+= v.getVenduti();
    return tot;
  }

  public Venditore migliore() {
    Venditore best = null;
    for (Venditore v : l)
      if (best == null || v.compareTo(best) > 0) best = v;
    return best;
  }
}
```

Soluzione Esercizio 6

```
import java.util.*;

class Applicazione {
  public static void main(String[] args) {
    Set<Divisione> s=new HashSet<Divisione>();
    Scanner scanner=new Scanner(System.in);
    Divisione d=new Divisione(scanner.nextLine());
    if (!s.add(d))
      System.out.println("Divisione già esistente!");
    Venditore v=new Venditore(scanner.nextLine(), scanner.nextInt(),
      scanner.nextInt());
    d.aggiungi(v);
    Venditore best = null;
    for (Divisione div: s) {
      Venditore x = div.migliore();
      if (best == null || x.compareTo(best) > 0)
        best = x;
    }
    System.out.println("Miglior venditore: " + best);
    int tot = 0;
    for (Divisione div: s)
      tot += div.totale();
    System.out.println("Totale venduti: " + tot);
  }
}
```