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

Appello del 18/1/2012

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

Esercizio 1 (4 punti)

1. Descrivere la codifica dei numeri interi all'interno di un elaboratore elettronico.
2. Descrivere le principali differenze tra insiemi e liste in Java.

Esercizio 2 (2 punti)

Rappresentare in binario il numero **5,8** supponendo di utilizzare 8 bit per la mantissa (in modulo e segno) ed 8 bit per l'esponente (in complemento a 2). Si riconverta infine il numero ottenuto in base 10, motivando eventuali differenze tra il numero originale e quello ottenuto.

Esercizio 3 (5 punti)

Siano dati i seguenti metodi Java:

```
public static int f(int V[], int M, int N) {
  int sum=0;
  for(i=M; i>0 && 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<M; j++)
    sum+=f(V, N, 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 `N` assume valori minori di 0 da quelli in cui assume valori maggiori o uguali a 0).
2. Calcolare la complessità in passi base del metodo `g` nei termini dei parametri `M` e `N` (si supponga `M>N`).
3. Calcolare la complessità asintotica del metodo `g` nei termini del parametro `N`.

Esercizio 4 (5 punti)

Il governo dello stato caraibico di St. Samuel desidera informatizzare la gestione della posta nella capitale dello stato. A tal scopo, i dati relativi alle abitazioni della capitale vengono memorizzate registrandone il cognome del proprietario ed il solo numero civico. Si scriva una classe `Abitazione` per le poste di St. Samuel 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 una descrizione dell'abitazione.
4. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Abitazione` (l'uguaglianza va verificata su cognome e numero civico).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Abitazione` passato come parametro (la precedenza va data per numero civico crescente e, in caso di parità, si procede in ordine alfabetico per cognome).

Esercizio 5 (7 punti)

Si scriva una classe `Via` che memorizzi le informazioni relative alle abitazioni presenti in una via. Per ogni via vengono memorizzati il nome della via stessa ed il cognome del postino che si occuperà della distribuzione della posta in tale via. Inoltre, le abitazioni presenti vanno inserite all'interno di una lista. La classe `Via` deve inoltre:

1. Presentare un opportuno costruttore con parametri (inizialmente, una via non contiene alcuna abitazione).
2. Possedere opportuni metodi che permettano di accedere alle variabili di istanza dell'oggetto.
3. Presentare il metodo `toString` che fornisca la descrizione della via (inclusa la descrizione di tutte le abitazioni).
4. Possedere il metodo `aggiungi` che, dato un oggetto `Abitazione`, lo inserisca all'interno della lista, mantenendola ordinata secondo il punto 5. dell'esercizio 4 (suggerimento: si utilizzi il metodo `add(int i, Abitazione a)` della classe `List<Abitazione>`).
5. Presentare il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Via` (l'uguaglianza va verificata solamente sul nome della via).
6. Possedere il metodo `quante` che indichi il numero di abitazioni presenti all'interno della via.
7. Presentare il metodo `totale` che, dato un cognome, restituisca il numero di abitazioni possedute da tale persona all'interno della via.

Esercizio 6 (8 punti)

Si scriva un'applicazione per le poste di St. Samuel che:

1. Crei un insieme di oggetti `Via`.
2. Crei un oggetto `Via`, 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 `Abitazione`, lette da tastiera le informazioni necessarie.
5. Inserisca l'abitazione creata al punto 4. nella via di cui al punto 2.
6. Stampi a video il nome della via che contiene più abitazioni tra quelle dell'insieme di cui al punto 1.
7. Letto da tastiera un cognome, stampi il numero totale di abitazioni possedute da tale persona nelle vie della capitale.

Soluzione Esercizio 2

$5{,}8_{10} = 101{,}\underline{1100}_2$ quindi la mantissa è **(0)1011100**, l'esponente 3_{10} = **00000011**.
Il numero rappresentato è $5{,}75_{10}$ in quanto il numero originale è periodico in base 2 e non è quindi rappresentabile con un numero finito di cifre binarie.

Soluzione Esercizio 3

Domanda 1:

2 assegnamenti	2	o 2
`i>0 && i>N`	M + 1	o M – N + 1
`sum+=V[--i]`	M	o M – N
Totale	2M + 3	o 2M – 2N + 3

Domanda 2:

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

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

Domanda 3:
Complessità asintotica: $O(MN)$

Soluzione Esercizio 4

```
class Abitazione implements Comparable<Abitazione> {
  private String cognome;
  private int numero;

  public Abitazione(String cognome, int numero) {
    this.cognome=cognome;
    this.numero=numero;
  }
  public String getCognome() { return cognome; }
  public int getNumero() { return numero; }
  public String toString() {
    return numero + ": " + cognome;
  }
  public boolean equals(Object o) { return equals((Abitazione) o); }
  public boolean equals(Abitazione a) {
    return numero==a.numero && cognome.equals(a.cognome);
  }
  public int compareTo(Abitazione a) {
    int ret=this.numero-a.numero;
    if(ret==0) ret=this.cognome.compareTo(a.cognome);
    return ret;
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;
class Via {
  private String nome, postino;
  private List<Abitazione> l;
  public Via (String nome, String postino) {
    this.nome=nome;
    this.postino=postino;
    l=new LinkedList<Abitazione>();
  }
  public String getNome() { return nome; }
  public String getPostino() { return postino; }
  public String toString() {
    return nome + " (" + postino + "):" + l.toString();
  }
  public void aggiungi(Abitazione a) {
    int i=0;
    while((i<l.size())&&(l.get(i).compareTo(a)<0)) i++;
    l.add(i, a);
  }
  public boolean equals(Object o) { return equals((Via) o); }
  public boolean equals(Via v) { return nome.equals(v.nome); }
  public int quante() { return l.size(); }
  public int totale(String cognome) {
    int n=0;
    for(Abitazione a:l) if(a.getCognome().equals(cognome)) n++;
    return n;
  }
}
```

Soluzione Esercizio 6

```
import java.util.*;
class Applicazione {
  public static void main(String[] args) {
    Set<Via> s=new TreeSet<Via>();
    Scanner scanner=new Scanner(System.in);
    Via v=new Via(scanner.next(), scanner.next());
    if(!s.add(v)) System.out.println("Via già presente!");
    Abitazione a=new Abitazione(scanner.next(), scanner.nextInt());
    v.aggiungi(a);
    Via max=null;
    for(Via x: s) {
      if(max==null || x.quante()>max.quante())
        max=x;
    }
    System.out.println(max);
    String cognome=scanner.next();
    int tot=0;
    for(Via x: s) tot+=x.totale(cognome);
    System.out.println(tot);
  }
}
```