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

Appello del 14/7/2011

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

Esercizio 1 (4 punti)

1. Descrivere l'architettura di un elaboratore elettronico.
2. Descrivere la codifica dei numeri reali all'interno di un elaboratore elettronico.

Esercizio 2 (2 punti)

Convertire in binario i numeri **47** e **91**, supponendo di utilizzare una rappresentazione a 8 bit in complemento a 2. Si esegua infine la somma dei due numeri, riconvertendo il risultato in base 10, motivando eventuali differenze tra il risultato ottenuto e quello atteso.

Esercizio 3 (5 punti)

Siano dati i seguenti metodi Java:

```
public static int f(int V[], int M) {
  int i=0, sum=0;
  while (i<M/2.0)
    sum+=V[i++];
  return sum;
}

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

1. Calcolare la complessità in passi base del metodo `f` nei termini del parametro `M` (si distinguano i casi in cui `M` 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` pari).
3. Calcolare la complessità asintotica del metodo `g` nei termini del parametro `N`.

Esercizio 4 (6 punti)

Lo studio di amministrazione condominiale "Condomini Amministrati Senza Equivoci" (CASE) vuole informatizzare l'archivio degli interventi effettuati sugli stabili amministrati. A tal scopo, ogni richiesta d'intervento da parte degli abitanti di un condominio viene catalogata in un computer, memorizzandone la data di richiesta, il cognome del richiedente ed una parte testuale che indica il tipo di intervento richiesto; inoltre, per ogni richiesta occorre indicare se l'intervento è stato effettuato o meno. Si scriva una classe `Intervento` per il CASE, che:

1. Possieda un opportuno costruttore con parametri (inizialmente, l'intervento non è stato effettuato).
2. Presenti opportuni metodi che permettano di accedere alle variabili di istanza dell'oggetto.
3. Possieda il metodo `risolto` che indichi che l'intervento relativo alla richiesta è stato effettuato.
4. Presenti il metodo `toString` che fornisca una descrizione dell'intervento.
5. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Intervento` (l'uguaglianza va verificata sulla data e sul cognome del richiedente).
6. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Intervento` passato come parametro (la precedenza va data alla richiesta più recente).

Esercizio 5 (8 punti)

Si scriva una classe `Condominio` che memorizzi le informazioni relative alle richieste di intervento di un singolo condominio. Oltre a memorizzare l'indirizzo del condominio ed il nome dell'amministratore, le richieste di intervento vanno inserite all'interno di una lista. La classe `Condominio` deve inoltre:

1. Presentare un opportuno costruttore con parametri (inizialmente, un condominio non presenta alcuna richiesta).
2. Presentare il metodo `getIndirizzo` che restituisca l'indirizzo del condominio.
3. Possedere il metodo `toString` che fornisca la descrizione del condominio (inclusa la descrizione di tutte le richieste effettuate).
4. Possedere il metodo `aggiungi` che, dato un oggetto `Intervento`, lo inserisca all'interno della lista, mantenendo tale lista ordinata secondo il punto 6. dell'esercizio 4 (suggerimento: si utilizzi il metodo `add(int j, Intervento i)` della classe `List<Intervento>`).
5. Presentare il metodo `necessitaIntervento` che indica se, tra le richieste di intervento del condominio, ne esistono di non ancora risolte.
6. Possedere il metodo `getInterventi` che, dato il cognome di un condomino, restituisca un insieme contenente tutti gli interventi richiesti da tale condomino.

Esercizio 6 (7 punti)

Si scriva un'applicazione per il CASE che:

1. Crei un insieme di oggetti `Condominio`.
2. Crei un oggetto `Condominio`, 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 `Intervento`, lette da tastiera le informazioni necessarie, e lo aggiunga alle richieste del condominio di cui al punto 2.
5. Letto da tastiera il cognome di un condomino, stampi a video la descrizione di tutte le richieste di intervento effettuate da tale persona all'interno del condominio di cui al punto 2.
6. Stampi a video l'indirizzo di tutti i condomini che presentano richieste di intervento non risolte.

Soluzione Esercizio 2

$47_{10} = 00101111_2$
$91_{10} = 01011011_2$

$10001010_2 = -118_{10}$

```
00101111+
01011011
10001010
```

Soluzione Esercizio 3

Domanda 1:

2 assegnamenti	2	o 2
`i<M/2.0`	M/2+1	o M/2+3/2
`sum+=V[i++]`	M/2	o M/2+1/2
Totale	M + 3	o M + 4

Domanda 2:

2 assegnamenti	2
`j<N`	N + 1
`sum+=f(V, ++j)`	N
complessità di `f`	$N^2/2 + 4N$
Totale	$N^2/2 + 6N + 3$

complessità di `f`: $\sum_{\substack{j=1 \\ j \text{ pari}}}^{N} (j+3) + \sum_{\substack{j=1 \\ j \text{ dispari}}}^{N} (j+4) = \sum_{j=1}^{N} j + \frac{3N}{2} + 2N = \frac{N(N+1)}{2} + \frac{7N}{2}$

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

Soluzione Esercizio 4

```
class Intervento implements Comparable<Intervento> {
  private String cognome, richiesta;
  private int a, m, g;
  private boolean risolto;

  public Intervento(String cognome, String richiesta, int g, int m, int a) {
    this.cognome=cognome;
    this.richiesta=richiesta;
    this.g=g; this.m=m; this.a=a;
    this.risolto=false;
  }

  public String getCognome() { return cognome; }
  public String getRichiesta() { return richiesta; }
  public String getData() { return g+"/"+m+"/"+a; }
  public boolean isRisolto() { return risolto; }
  public void risolto() { risolto=true; }

  public String toString() {
    return cognome+ " (" + getData()+"): " + richiesta;
  }

  public boolean equals(Object o) { return equals((Intervento) o); }
  public boolean equals(Intervento i) {
    return cognome.equals(i.cognome) && getData().equals(i.getData());
  }

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

Soluzione Esercizio 5

```
import java.util.*;

class Condominio {
  private String indirizzo, amministratore;
  private List<Intervento> l;

  public Condominio(String indirizzo, String amministratore) {
    this.indirizzo=indirizzo;
    this.amministratore=amministratore;
    l=new LinkedList<Intervento>();
  }

  public String getIndirizzo() { return indirizzo; }

  public String toString() {
    return indirizzo + " (" + amministratore + "):" + l.toString();
  }

  public void aggiungi(Intervento i) {
    int j=0;
    while((j<l.size())&&(l.get(j).compareTo(i)<0)) j++;
    l.add(j, i);
  }

  public boolean necessitaIntervento() {
    for(Intervento i:l)
      if(!i.isRisolto()) return true;
    return false;
  }

  public Set<Intervento> getInterventi(String nome) {
    Set<Intervento> s=new TreeSet<Intervento>();
    for(Intervento i:l)
      if(i.getCognome().equals(nome)) s.add(i);
    return s;
  }
}
```

Soluzione Esercizio 6

```
import java.util.*;

class Applicazione {
  public static void main(String[] args) {
    Set<Condominio> s=new HashSet<Condominio>();
    Scanner scanner=new Scanner(System.in);
    Condominio c=new Condominio(scanner.next(), scanner.next());
    if(!s.add(c)) System.out.println("Condominio già presente!");
    c.aggiungi(new Intervento(scanner.next(), scanner.next(),
      scanner.nextInt(), scanner.nextInt(), scanner.nextInt()));
    String nome=scanner.next();
    System.out.println(c.getInterventi(nome));
    for(Condominio x:s)
      if(x.necessitaIntervento()) System.out.println(x.getIndirizzo());
  }
}
```