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

Appello del 16/10/2015

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

Esercizio 1 (4 punti)

1. Discutere le differenze tra un linguaggio interpretato e uno compilato.
2. Descrivere i tipi primitivi del linguaggio Java.

Esercizio 2 (2 punti)

Convertire in binario i numeri **–39** e **–89**, 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 N) {
  int i=N, sum=0;
  while (++i<=M)
    sum+=V[i];
  return sum;
}

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

Esercizio 4 (5 punti)

Angelo Maletti e Carmelo Sorsi sono due intraprendenti dirigenti dell'Agenzia delle Entrate. In vista delle prossime ispezioni, hanno deciso di memorizzare all'interno di un calcolatore le informazioni relative ai contribuenti da controllare. In particolare, occorre registrare il nome, l'indirizzo, il codice fiscale e il reddito complessivo. Si scriva una classe `Contribuente` per Maletti e Sorsi 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 del contribuente.
4. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Contribuente` (la verifica va fatta unicamente sul codice fiscale).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Contribuente` passato come parametro (in ordine decrescente di reddito e, a parità, in ordine alfabetico per nome).

Esercizio 5 (7 punti)

Si scriva una classe `Ispezione` che memorizzi le informazioni sui contribuenti da controllare in una certa verifica. Per ciascun'ispezione occorre memorizzare la data e il nome dell'ispettore, mentre i contribuenti vanno inseriti all'interno di una lista. La classe `Ispezione` deve:

1. Presentare un opportuno costruttore con parametri (inizialmente, non vi sono contribuenti).
2. Possedere opportuni metodi che permettano di accedere alle variabili d'istanza dell'oggetto.
3. Presentare il metodo `toString` che fornisca la descrizione dell'ispezione (inclusa la descrizione di tutti i contribuenti da controllare).
4. Possedere il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Ispezione` (la verifica va effettuata unicamente sulla data).
5. Presentare il metodo `aggiungi` che, dato un oggetto `Contribuente`, lo inserisca all'interno della lista, mantenendo quest'ultima ordinata secondo il punto 5. dell'Esercizio 4.
6. Possedere il metodo `totale` che restituisca la somma dei redditi dei contribuenti da controllare.
7. Presentare il metodo `contribuente` che, dato il codice fiscale di un contribuente, indichi se esso è incluso nell'ispezione.

Esercizio 6 (7 punti)

Si scriva un'applicazione per Maletti e Sorsi che:

1. Crei un insieme di oggetti `Ispezione`.
2. Crei un oggetto `Ispezione`, 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 `Contribuente`, lette da tastiera le informazioni necessarie.
5. Inserisca l'oggetto di cui al punto 4. all'interno dell'ispezione di cui al punto 2.
6. Letto da tastiera il nome di un ispettore, stampi a video la descrizione di tutte le ispezioni da lui effettuate.
7. Stampi a video la data dell'ispezione che prevede la massima somma totale di redditi.

Soluzione Esercizio 2

$-39_{10} = 11011001_2$
$-89_{10} = 10100111_2$

$10000000_2 = -128_{10}$

```
11011001+
10100111
--------
10000000
```

Soluzione Esercizio 3

Domanda 1:

2 assegnamenti	2	o 2
`i++<=M`	1	o $M - N + 1$
`sum+=V[i]`	0	o $M - N$
Totale	3	o $2M - 2N + 3$

Domanda 2:

2 assegnamenti	2
`j>0`	$N + 1$
`sum+=f(V, M, j--)`	N
complessità di `f`	$M^2 + 3N - M$
Totale	$M^2 + 5N - M + 3$

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

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

Soluzione Esercizio 4

```
class Contribuente implements Comparable<Contribuente> {
  private String nome, indirizzo, cf;
  private int reddito;

  public Contribuente(String nome, String ind, String cf, int reddito) {
    this.nome = nome;
    this.indirizzo = ind;
    this.cf = cf;
    this.reddito = reddito;
  }

  public String getNome() { return nome; }
  public String getIndirizzo() { return indirizzo; }
  public String getCF() { return cf; }
  public int getReddito() { return reddito; }

  public String toString() {
    return nome + ", " + indirizzo + " (" + cf + "): " + reddito;
  }

  public boolean equals(Object o) { return equals((Contribuente) o); }
  public boolean equals(Contribuente c) { return this.cf.equals(c.cf); }

  public int compareTo(Contribuente c) {
    int ret = c.reddito - this.reddito;
    if(ret==0) ret = this.nome.compareTo(c.nome);
    return ret;
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;

class Ispezione {
  private List<Contribuente> l;
  private String ispettore;
  private int g, m, a;

  public Ispezione(String ispettore, int g, int m, int a) {
    this.ispettore = ispettore;
    this.g = g; this.m = m; this.a = a;
    l = new ArrayList<Contribuente>();
  }

  public String getIspettore() { return ispettore; }
  public String getData() { return g+"/"+m+"/"+a; }
  public String toString() { return ispettore + "(" + getData() + "):" + l;}
  public boolean equals(Object o) { return equals((Ispezione) o); }
  public boolean equals(Ispezione i){ return getData().equals(i.getData());}

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

  public int totale() {
    int totale = 0;
    for(Contribuente c: l) totale += c.getReddito();
    return totale;
  }

  public boolean contribuente(String cf) {
    for(Contribuente c: l) if(c.getCF().equals(cf)) return true;
    return false;
  }
}
```

Soluzione Esercizio 6

```
import java.util.*;
class Applicazione {
  public static void main(String[] args) {
    Set<Ispezione> s = new HashSet<Ispezione>();
    Scanner scanner = new Scanner(System.in);
    Ispezione i = new Ispezione(scanner.nextLine(), scanner.nextInt(),
        scanner.nextInt(), scanner.nextInt());
    if(!s.add(i)) System.out.println("Ispezione già esistente!");
    Contribuente c = new Contribuente(scanner.nextLine(),
        scanner.nextLine(), scanner.nextLine(), scanner.nextInt());
    i.aggiungi(c);
    String nome = scanner.nextLine();
    for(Ispezione x: s) if(x.getIspettore().equals(nome))
        System.out.println(x);
    Ispezione max=null;
    int maxR=0;
    for(Ispezione x: s) {
      int totale=x.totale();
      if(totale>maxR) { max=x; maxR=totale; }
    }
    System.out.println(max.getData());
  }
}
```