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

Appello del 13/10/2017

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 array e collezioni in Java.
2. Definire il concetto di ricorsione.

Esercizio 2 (2 punti)

Rappresentare in binario il numero **12,375** 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 rappresentato, motivando eventuali differenze con il numero originale.

Esercizio 3 (5 punti)

Siano dati i seguenti metodi Java:

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

public static int g(int V[], int N) {
  int j=0, sum=0;
  while(j<N)
    sum+=f(V, 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 `M` assume valori maggiori di `N` da quelli in cui assume valori minori o uguali).
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 (5 punti)

Massimiliano Nappi è un brillante astrofisico che studia gli ammassi stellari nella nostra galassia. Per meglio organizzare le osservazioni effettuate al telescopio, ha deciso di gestire all'interno di un calcolatore le informazioni relative agli astri osservati. In particolare, per ogni stella occorre registrare il nome, la massa (in unità di massa solare, ovvero in rapporto alla massa del sole) e la distanza dal sole in parsec. Si scriva una classe `Stella` per Massimiliano Nappi 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 della stella.
4. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Stella` (la verifica va fatta sul nome e sulla massa).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Stella` passato come parametro (per ordine alfabetico sul nome e, a parità, per massa crescente).

Esercizio 5 (7 punti)

Si scriva una classe `Ammasso` che registri le informazioni riguardanti un singolo ammasso stellare. Per ogni ammasso occorre memorizzare il nome, e l'età apparente (in milioni di anni), mentre le stelle facenti parte dell'ammasso vanno memorizzate all'interno di un insieme. La classe `Ammasso` deve:

1. Presentare un opportuno costruttore con parametri (l'insieme di stelle è inizialmente vuoto).
2. Possedere opportuni metodi che permettano di accedere alle variabili d'istanza dell'oggetto.
3. Presentare il metodo `toString` che fornisca la descrizione dell'ammasso (inclusa la descrizione di tutte le stelle).
4. Possedere il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Ammasso` (la verifica va effettuata esclusivamente sul nome).
5. Presentare il metodo `aggiungi` che, dato un oggetto `Stella`, lo inserisca all'interno dell'insieme, controllando che tale inserimento sia possibile.
6. Possedere il metodo `pesanti` che, dato un valore di masse solari, restituisca il numero di stelle presenti nell'ammasso aventi una massa non inferiore a tale valore.
7. Possedere il metodo `distanza` che restituisca la distanza media delle stelle presenti nell'ammasso.

Esercizio 6 (8 punti)

Si scriva un'applicazione per Massimiliano Nappi che:

1. Crei una lista di oggetti `Ammasso`.
2. Crei un oggetto `Ammasso`, lette da tastiera le informazioni necessarie.
3. Inserisca l'oggetto di cui al punto 2. in coda alla lista di cui al punto 1.
4. Crei un oggetto `Stella`, lette da tastiera le informazioni necessarie.
5. Inserisca l'oggetto di cui al punto 4. all'interno dell'ammasso di cui al punto 2., controllando che tale inserimento sia possibile.
6. Letto da tastiera un valore di masse solari, stampi a video il nome di tutti gli ammassi comprendenti più di cinque stelle con una massa non inferiore a tale valore.
7. Stampi a video la descrizione dell'ammasso a minor distanza media dal sole.

Soluzione Esercizio 2

$12{,}375_{10} = 1100{,}011_2$ quindi la mantissa è **(0)1100011**, l'esponente 4_{10} = **00000100**.
Il numero rappresentato è uguale all'originale.

Soluzione Esercizio 3

Domanda 1:

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

Domanda 2:

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

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

Domanda 3:

Complessità asintotica: $O(N^2)$

Soluzione Esercizio 4

```
class Stella implements Comparable<Stella> {
  private String nome;
  private float massa, distanza;

  public Stella(String nome, float massa, float distanza) {
    this.nome = nome;
    this.massa = massa;
    this.distanza = distanza;
  }

  public String getNome() { return nome; }
  public float getMassa() { return massa; }
  public float getDistanza() { return distanza; }

  public String toString() {
    return nome + " (" + massa + "): " + distanza;
  }

  public boolean equals(Object o) { return equals((Stella) o); }
  public boolean equals(Stella s) {
    return nome.equals(s.nome) && massa==s.massa;
  }

  public int compareTo(Stella s) {
    int ret = this.nome.compareTo(s.nome);
    if(ret!=0) return ret;
    if(this.massa<s.massa) return -1;
    if(this.massa>s.massa) return 1;
    return 0;
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;

class Ammasso {
  private String nome;
  private int eta;
  private Set<Stella> s;

  public Ammasso(String nome, int eta) {
    this.nome = nome;
    this.eta = eta;
    s = new HashSet<Stella>();
  }

  public String getNome() { return nome; }
  public int getEta() { return eta; }
  public String toString() { return nome + "(" + eta + ": " + s;}
  public boolean equals(Object o) { return equals((Ammasso) o); }
  public boolean equals(Ammasso a){ return nome.equals(a.nome);}
  public boolean aggiungi(Stella s) { return this.s.add(s); }

  public int pesanti(float massa) {
    int count=0;
    for(Stella s: this.s) if(s.getMassa()>=massa) count++;
    return count;
  }

  public float distanza() {
    int totale=0;
    for(Stella s: this.s) totale+=s.getDistanza();
    return totale/this.s.size();
  }
}
```

Soluzione Esercizio 6

```
import java.util.*;

class Applicazione {
  public static void main(String[] args) {
    List<Ammasso> l = new ArrayList< Ammasso >();
    Scanner scanner = new Scanner(System.in);
    Ammasso a = new Ammasso(scanner.nextLine(), scanner.nextInt());
    l.add(a);
    Stella s = new Stella(scanner.nextLine(), scanner.nextFloat(),
        scanner.nextFloat());
    if(!a.aggiungi(s)) System.out.println("Inserimento non riuscito!");
    float massa = scanner.nextFloat();
    for(Ammasso x: l)
      if(x.pesanti(massa)>5) System.out.println(x.getNome());
    float minD = 0;
    Ammasso min = null;
    for(Ammasso x: l) {
      float distanza= x.distanza();
      if(min==null||distanza<minD) { min=x; minD=distanza; }
    }
    System.out.println(min.toString());
  }
}
```