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

Appello del 16/6/2010

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

Esercizio 1 (4 punti)

1. Illustrare (con un esempio) il ciclo di esecuzione di un'istruzione da parte della CPU (fetch/execute).
2. Descrivere le principali differenze tra insiemi e liste in Java.

Esercizio 2 (2 punti)

Rappresentare in binario il numero **–0,049** supponendo di utilizzare 8 bit per la mantissa (in modulo e segno) ed 8 bit per l'esponente (in complemento a 2).

Esercizio 3 (5 punti)

Siano dati i seguenti metodi Java:

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

public static int g(int V[],int N) {
  int j, sum=0;
  for(j=0; j<N; j++)
    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 e dispari).
2. Calcolare la complessità in passi base del metodo `g` nei termini del parametro `N` (si supponga `N` dispari e si esprima `j=2 i` + 1).
3. Calcolare la complessità asintotica del metodo `g` nei termini del parametro `N`.

Esercizio 4 (6 punti)

La pizzeria "San Petronio" ha deciso di informatizzare la gestione delle ordinazioni all'interno del proprio locale. In particolare, per ogni piatto presente nel menù occorre memorizzare il nome, la tipologia (es. pizza, primo, bevanda) ed il prezzo di vendita. Si scriva una classe `Piatto` per la pizzeria "San Petronio" 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 la descrizione del piatto.
4. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Piatto` (l'uguaglianza va verificata unicamente sul nome del piatto).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Piatto` passato come parametro (la precedenza va verificata per tipologia crescente e, in caso di parità, si procede per ordine alfabetico sul nome).

Esercizio 5 (8 punti)

Si scriva una classe `Ordinazione` che memorizzi le informazioni relative alle ordinazioni di un tavolo. Oltre al numero del tavolo, i vari piatti ordinati vanno inseriti all'interno di una lista. La classe `Ordinazione` deve inoltre:

1. Presentare un opportuno costruttore (inizialmente, nessun piatto è stato ordinato).
2. Presentare un metodo che restituisca il numero del tavolo relativo all'ordinazione corrente.
3. Possedere il metodo `toString` che fornisca la descrizione dell'ordinazione (inclusa la descrizione di tutti i piatti ordinati).
4. Presentare il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Ordinazione` (l'uguaglianza va verificata sull'uguaglianza del numero di tavolo).
5. Possedere il metodo `aggiungi` che, dato un oggetto di tipo `Piatto`, lo inserisca all'interno della lista dei piatti, mantenendo tale lista ordinata secondo il punto 5. dell'esercizio 4 (suggerimento: si utilizzi il metodo `add(int i, Piatto p)` della classe `List<Piatto>`).
6. Presentare il metodo `totale` che, data una tipologia di piatti, restituisca il prezzo totale di tutti i piatti relativi a tale tipologia. Se non viene specificata alcuna tipologia, occorre restituire il prezzo totale dell'ordinazione.

Esercizio 6 (6 punti)

Si scriva un'applicazione per la pizzeria "San Petronio" che:

1. Crei un insieme di oggetti `Ordinazione`.
2. Crei un oggetto `Ordinazione`, 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. Letto da tastiera il numero di un tavolo, stampi le informazioni sulle ordinazioni di tale tavolo.
5. Letta da tastiera una tipologia di piatto, stampi a video il numero di tavolo con la spesa maggiore in tale categoria.

Soluzione Esercizio 2

$-0{,}049_{10} = -0{,}00001100100_2$ quindi la mantissa è **(1)1100100** e l'esponente -4_{10} = **11111100**.

Soluzione Esercizio 3

Domanda 1:

2 assegnamenti	2	
`while(i++<M)`	`M`/2 + 1	o `(M+1)`/2 + 1
`sum+=V[++i]`	`M`/2	o `(M+1)`/2
otale	`M` + 3	o `M` + 4

Domanda 2:

2 assegnamenti	2
`j<N`	`(N+1)`/2 + 1
`sum+=f(V, ++j)`	`(N+1)`/2
`j++`	`(N+1)`/2
complessità di `f`	$N^2/4 + 5N/2 + 9/4$
Totale	$N^2/4 + 4N + 27/4$

complessità di `f`: $\sum_{\substack{j=1 \\ (j\ \text{dispari})}}^{N}(j+4) = \sum_{i=0}^{(N-1)/2}(2i+5) = \frac{N-1}{2}\frac{N+1}{2} + 5\frac{N+1}{2} = \frac{N^2}{4} + \frac{5N}{2} + \frac{9}{4}$

Domanda 3:

Complessità asintotica: $O(N^2)$

Soluzione Esercizio 4

```
class Piatto implements Comparable<Piatto> {
  private String nome, tipo;
  private float prezzo;

  public Piatto(String nome, String tipo, float prezzo) {
    this.nome=nome;
    this.tipo=tipo;
    this.prezzo=prezzo;
  }

  public String getNome() { return nome; }
  public String getTipo() { return tipo; }
  public float getPrezzo() { return prezzo; }

  public String toString() {
    return nome + "(" + tipo + "): " + prezzo;
  }

  public boolean equals(Object o) { return equals((Piatto) o); }
  public boolean equals(Piatto p) {
    return nome.equals(p.nome);
  }

  public int compareTo(Piatto p) {
    int ret=tipo.compareTo(p.tipo);
    if(ret==0) ret=this.nome.compareTo(p.nome);
    return ret;
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;

class Ordinazione {
  private int tavolo;
  private List<Piatto> l;

  public Ordinazione(int tavolo) {
    this.tavolo=tavolo;
    l=new LinkedList<Piatto>();
  }

  public int getTavolo() { return tavolo; }
  public String toString() { return tavolo+"\n"+l; }

  public boolean equals(Object o) { return equals((Ordinazione) o); }
  public boolean equals(Ordinazione o) {  return tavolo==o.tavolo; }

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

  public float totale(String tipo) {
    float totale=0;
    for(Piatto p:l)
      if((tipo==null)||(p.getTipo().equals(tipo))) totale+=p.getPrezzo();
    return totale;
  }
}
```

Soluzione Esercizio 6

```
import java.util.*;

class Applicazione {
  public static void main(String[] args) {
    Set<Ordinazione> s=new HashSet<Ordinazione>(); // domanda 1
    Scanner scanner=new Scanner(System.in);
    Ordinazione o=new Ordinazione(scanner.nextInt());// domanda 2
    if (!s.add(o)) System.out.println("Tavolo già esistente!");// domanda 3
    int tavolo=scanner.nextInt();
    for(Ordinazione or:s)
      if(or.getTavolo()==tavolo) {
        System.out.println(or); break;
      } // domanda 4
    String tipo=scanner.next();
    float min=0;
    for(Ordinazione or:s) {
      float totale=or.totale(tipo);
      if(totale>min) {
        min=totale; tavolo=or.getTavolo();
      }
    }
    System.out.println(tavolo); // domanda 5
  }
}
```