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

Appello del 22/10/2021

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

Esercizio 1 (4 punti)

1. Descrivere le modalità di studio della complessità temporale di un algoritmo.
2. Descrivere le principali differenze tra array e collezioni in Java.

Esercizio 2 (2 punti)

Rappresentare in binario il numero –**0,035** supponendo di utilizzare 8 bit per la mantissa (in modulo e segno) ed 8 bit per l'esponente (in complemento a 2). Si motivino infine eventuali differenze tra il numero rappresentato e 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 M, int N) {
  int sum=0;
  for(j=0;j++<N;)
    sum+=f(V, j, M);
  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 di `N` da quelli in cui assume valori maggiori o uguali).
2. Calcolare la complessità in passi base del metodo `g` nei termini dei parametri `M` e `N` (si supponga `M` maggiore di `N`).
3. Calcolare la complessità asintotica del metodo `g` nei termini dei parametri `M` e `N`.

Esercizio 4 (5 punti)

Mattia Cavalieri ha appena cambiato lavoro ma ha scoperto che il mestiere di rappresentante richiede molta organizzazione. Pertanto, ha deciso di memorizzare le informazioni sulle varie trasferte all'interno di un calcolatore. Per ogni cliente occorre memorizzare la ragione sociale e l'indirizzo, composto da via e numero civico, comune e CAP. Si scriva una classe `Cliente` per Mattia Cavalieri 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 cliente.
4. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Cliente` (la verifica va fatta sulla ragione sociale).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Cliente` passato come parametro (per comune e, a parità, per CAP).

Esercizio 5 (7 punti)

Si scriva una classe `Trasferta` che registri le informazioni riguardanti i clienti visitati durante una trasferta. Per ogni trasferta occorre memorizzare la data, mentre i clienti vanno inseriti all'interno di una lista. La classe `Trasferta` deve:

1. Presentare un opportuno costruttore con parametri (inizialmente, la lista di clienti è vuota).
2. Possedere opportuni metodi che permettano di accedere alle variabili d'istanza dell'oggetto.
3. Presentare il metodo `toString` che fornisca la descrizione della trasferta (inclusa la descrizione di tutti i clienti da visitare).
4. Possedere il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Trasferta` (la verifica va effettuata esclusivamente sulla data).
5. Presentare il metodo `aggiungi` che, dato un oggetto `Cliente`, lo inserisca in coda alla lista.
6. Possedere il metodo `presente` che, dato il nome di un comune, indichi se tale comune è incluso durante la trasferta.
7. Possedere il metodo `quanti` che indichi il numero di clienti da visitare durante la trasferta.

Esercizio 6 (7 punti)

Si scriva un'applicazione per Mattia Cavalieri che:

1. Crei un insieme di oggetti `Trasferta`.
2. Crei un oggetto `Trasferta`, 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 `Cliente`, lette da tastiera le informazioni necessarie.
5. Inserisca l'oggetto di cui al punto 4. all'interno della trasferta di cui al punto 2.
6. Stampi a video la data di tutte le trasferte che prevedono una visita a Cesena.
7. Stampi la data della trasferta che prevede di visitare il maggior numero di clienti.

Soluzione Esercizio 2

$-0{,}035_{10} = -0{,}00001000111\ldots_2$ quindi la mantissa è **(1)1000111**, l'esponente -4_{10} = **11111100**.
Il numero rappresentato è diverso dall'originale poiché quest'ultimo è periodico in base 2 e pertanto necessita di un numero infinito di cifre per la mantissa.

Soluzione Esercizio 3

Domanda 1:

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

Domanda 2:

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

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

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

Soluzione Esercizio 4

```
class Cliente implements Comparable<Cliente> {
  private String nome, indirizzo, comune, CAP;

  public Cliente(String nome, String indirizzo, String comune, String CAP) {
  this.nome = nome;
    this.indirizzo = indirizzo;
    this.comune = comune;
    this.CAP = CAP;
  }

  public String getNome() { return nome; }
  public String getIndirizzo() { return indirizzo; }
  public String getComune() { return comune; }
  public String getCAP() { return CAP; }

  public String toString() {
    return nome + ": " + indirizzo + ", " + comune + " (" + CAP + ")";
  }

  public boolean equals(Object o) { return equals((Cliente) o); }
  public boolean equals(Cliente c) { return nome.equals(c.nome()) }

  public int compareTo(Cliente c) {
    int ret = this.comune.compareTo(c.comune);
    if(ret==0) ret = this.CAP.compareTo(c.CAP);
    return ret;
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;

class Trasferta {
  private int g, m, a;
  private List<Cliente> l;

  public Trasferta(int g, int m, int a) {
    this.g = g; this.m = m; this.a = a;
    l = new LinkedList<Cliente>();
  }

  public String getData() { return g+"/"+m+"/"+a; }
  public String toString() { return getData() + ": " + l; }

  public boolean equals(Object o) { return equals((Trasferta) o); }
  public boolean equals(Trasferta t){ return getData().equals(t.getData()); }

  public void aggiungi(Cliente c) { l.add(c); }
  public boolean presente(String comune) {
    for(Cliente c: l) if(c.getComune().equals(comune)) return true;
    return false;
  }

  public int quanti() { return l.size(); }
}
```

Soluzione Esercizio 6

```
import java.util.*;

class Applicazione {
  public static void main(String[] args) {
    Set<Trasferta> s = new TreeSet<Trasferta>();
    Scanner scanner = new Scanner(System.in);
    Trasferta t=new Trasferta(scanner.nextInt(), scanner.nextInt(),
        scanner.nextInt());

    if(!s.add(t)) System.out.println("Inserimento non avvenuto!");

    Cliente c = new Cliente(scanner.nextLine(), scanner.nextLine(),
        scanner.nextLine(), scanner.nextLine());

    t.aggiungi(c);

    for(Trasferta x: s)
      if(x.presente("Cesena")) System.out.println(x.getData());

    Trasferta max=null;
    int maxCount=0;
    for(Trasferta x: s) {
      int count=x.quanti();
      if(count>maxCount) { max=x; maxCount=count; }
    }
    System.out.println(max.getData());
  }
}
```