Esame di Fondamenti di Informatica L-B
Ingegneria Gestionale e dei Processi Gestionali (L-Z)

Appello del 16/2/2007

Esercizio 1 (4 punti)

Illustrare il concetto di complessità di un algoritmo.

Esercizio 2 (5 punti)

Siano date le seguenti funzioni C:

```
void f(int V[], int M) {
  V[M]=1;
  while(M%2)
    V[M--]++;
}

 void g(int V[], int N) {
  int i=1;

  while(i<=N)
    f(V,i++);
}
```

1. Calcolare la complessità in passi base della funzione `f` nei termini del parametro `M` (suggerimento: si distinguano i casi in cui `M` assume valori pari da quelli in cui assume valori dispari).
2. Calcolare la complessità in passi base della funzione `g` nei termini del parametro `N` (suggerimento: si distinguano i casi in cui `N` assume valori pari da quelli in cui assume valori dispari).
3. Calcolare la complessità asintotica della funzione `g` nei termini del parametro `N`.

Esercizio 3 (5 punti)

La società assicuratrice "Tutti Sinistri" ha deciso di informatizzare la gestione delle assicurazioni RC auto. Le informazioni su ogni veicolo comprendono la targa, la marca, il modello, i cavalli fiscali (un semplice intero) e l'anno di immatricolazione.
Si scriva una classe `Veicolo` per la società "Tutti Sinistri" che:

1. Possieda un opportuno costruttore con parametri.
2. Presenti opportuni metodi che permettano di accedere alle variabili di istanza della classe.
3. Possieda il metodo `toString` che fornisca una descrizione testuale del veicolo.
4. Presenti un metodo `equals` per stabilire l'uguaglianza con un altro veicolo (l'uguaglianza va verificata unicamente sulla targa).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` che effettui il confronto con un altro veicolo (la precedenza va data al veicolo con anno di immatricolazione più recente, a parità di anno vale l'ordine lessicografico sulla targa).

Esercizio 4 (9 punti)

Si scriva una classe `Cliente` che memorizzi le informazioni relative ai clienti assicurati. In particolare, per ogni cliente, oltre ai dati anagrafici (nome, cognome, anno di nascita) vanno memorizzati i veicoli da lui posseduti (contenuti all'interno di un insieme) ed il valore della classe bonus/malus (un valore intero). La classe `Cliente` deve inoltre:

1. Possedere un opportuno costruttore con parametri (inizialmente il cliente non possiede veicoli ed è in classe 14).
2. Presentare opportuni metodi che permettano di accedere alle variabili di istanza della classe.
3. Possedere un metodo `modificaClasse` che, dato un valore intero, provveda a modificare opportunamente la classe bonus/malus (il valore minimo è comunque 1 ed il massimo 18).
4. Presentare un metodo `aggiungi` che, dato un `Veicolo`, lo inserisca all'interno dell'insieme dei veicoli del cliente, se questo non è già presente.
5. Possedere il metodo `rischio` che calcoli il fattore di rischio del cliente (ovvero classe bonus/malus moltiplicata per il numero di veicoli posseduti).
6. Presentare il metodo `possiede` che, data la targa di un veicolo, verifica se questo è posseduto dal cliente.
7. Possedere il metodo `toString` che restituisca una stringa che includa le informazioni sul cliente (comprendendo anche le informazioni su tutti i veicoli posseduti).

Esercizio 5 (8 punti)

Si scriva un'applicazione per la società "Tutti Sinistri" che:

1. Crei un insieme di oggetti `Cliente`.
2. Lette da tastiera le informazioni relative ad un nuovo cliente, provveda ad inserire un nuovo oggetto nell'insieme, verificando che tale oggetto non sia già contenuto nell'insieme stesso.
3. Lette da tastiera le informazioni relative ad un nuovo veicolo, provveda a creare un nuovo oggetto `Veicolo` e lo assegni al cliente di cui al punto 2.
4. Letta da tastiera la targa di un veicolo, provveda a decrementare di 2 punti la classe del cliente che possiede tale veicolo.
5. Stampi le informazioni relative al cliente avente fattore di rischio più elevato all'interno dell'insieme di cui al punto 1.

Per la lettura di dati da tastiera è possibile utilizzare l'oggetto `Lettore.in`, definito all'interno del package `fiji.io`, che possiede i seguenti metodi:

- `char leggiChar()` Legge un singolo carattere.
- `float leggiFloat()` Legge un numero razionale (delimitato da spazi).
- `int leggiInt()` Legge un intero (delimitato da spazi).
- `String leggiLinea()` Legge una linea di testo.
- `String leggiString()` Legge una parola senza spazi al suo interno.

Soluzione Esercizio 2

Domanda 1:

1 assegnamento	1
`while(M%2)`	1 o 2
assegnamento	1
Totale (`M` pari)	2
Totale (`M` dispari)	4

Domanda 2:

1 assegnamento	1
`while` eseguito `N` volte	`N` + 1
chiamata di `f`	`N`
complessità di `f` (`N` pari)	3 `N`
complessità di `f` (`N` dispari)	3 `N` + 1
Totale (`N` pari)	5 `N` + 2
Totale (`N` dispari)	5 `N` + 3

Domanda 3:

Complessità asintotica: *O*(`N`)

Soluzione Esercizio 3

```
class Veicolo implements Comparable {
  private String targa, marca, modello;
  private int cf, anno;

  public Veicolo(String t, String m, String md, int cf, int a) {
    targa=t; marca=m; modello=m;
    this.cf=cf; anno=a;
  }

  public String getTarga() { return targa; }
  public int getCF() { return cf; }
  public int getAnno() { return anno; }

  public String toString() {
    return targa+" ("+marca+"-"+modello+"): "+cf+", "+anno;
  }

  public boolean equals(Veicolo v) {
    return(targa.equals(v.targa));
  }

  public int compareTo(Veicolo v) {
    int ret=(v.anno-anno);
    if(ret==0) ret=targa.compareTo(v.targa);
    return ret;
  }

  public boolean equals(Object o) { return equals((Veicolo) o); }
  public int compareTo(Object o) { return compareTo((Veicolo) o); }
}
```

Soluzione Esercizio 4

```
import java.util.*;

class Cliente {
  private String nome, cognome;
  private int classe, anno;
  private Set<Veicolo> veicoli;

  public Cliente(String nome, String cognome, int anno) {
    this.nome=nome; this.cognome=cognome;
    this.anno=anno; classe=14;
    veicoli=new HashSet<Veicolo>();
  }

  public String getNome() { return nome+" "+cognome; }
  public void modificaClasse(int p) {
    classe+=p;
    if(classe<1) classe=1;
    if(classe>18) classe=18;
  }
  public boolean aggiungi(Veicolo v) { return veicoli.add(v); }
  public int rischio() { return classe*veicoli.size(); }
  public boolean possiede(String t) {
    Veicolo v=new Veicolo(t, "", "", 0, 0);
    return veicoli.contains(v);
  }

  public String toString() {
    return getNome()+"("+anno+"):\n"+veicoli.toString();
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;
import fiji.io.*;
class Applicazione {
  public static void main(String[] args) {
    Set<Cliente> clienti=new TreeSet<Cliente>(); // domanda 1
    Cliente c, maxC=null;
    String targa;
    int max=0;

    c=new Cliente(Lettore.in.leggiLinea(), Lettore.in.leggiLinea(),
      Lettore.in.leggiInt());
    if(!clienti.add(c)) System.out.println("Cliente già esistente!");
      // domanda 2
    c.aggiungi(new Veicolo(Lettore.in.leggiString(),
      Lettore.in.leggiString(), Lettore.in.leggiString(),
      Lettore.in.leggiInt(), Lettore.in.leggiInt())); // domanda 3
    targa=Lettore.in.leggiString();
    for(Cliente cl:clienti) if(cl.possiede(targa)) cl.modificaClasse(-2);
    for(Cliente cl:clienti)
      if(cl.rischio()>max) {
        maxC=cl;
        max=cl.rischio();
    }
    System.out.println(maxC); // domanda 5
  }
}
```