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

Appello del 14/9/2018

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

Esercizio 1 (4 punti)

1. Illustrare il ciclo di esecuzione di un'istruzione da parte della CPU (fetch/decode/execute).
2. Illustrare i concetti di over-riding e di late binding di un metodo.

Esercizio 2 (2 punti)

Convertire in binario i numeri **–46** e **–82**, 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-M];
  return sum;
}

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

Esercizio 4 (5 punti)

Roberto Cipullo è molto impaziente: il giorno delle nozze sta finalmente per arrivare, ma occorre organizzare il trasferimento al luogo delle nozze degli invitati giunti dal paese natale e ospitati in albergo. Per questo, Roberto ha deciso di informatizzare la gestione degli invitati memorizzandone il nome, il numero di stanza e il numero di persone (ogni invitato, infatti, può venire accompagnato da parenti, figli, ecc.). Si scriva una classe `Invitato` per Roberto Cipullo 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 dell'invitato.
4. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Invitato` (la verifica va fatta unicamente su nome e numero di stanza).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Invitato` passato come parametro (in ordine alfabetico per nome e, a parità, in ordine crescente di numero di invitati).

Esercizio 5 (7 punti)

Si scriva una classe `Navetta` che registri le informazioni riguardanti i bus navetta utilizzati per il trasporto degli invitati. Per ogni navetta occorre memorizzare la targa e il numero massimo di passeggeri mentre gli invitati caricati vanno memorizzati all'interno di un insieme. La classe `Navetta` deve:

1. Presentare un opportuno costruttore con parametri (inizialmente, l'insieme degli invitati è vuoto).
2. Possedere opportuni metodi che permettano di accedere alle variabili d'istanza dell'oggetto.
3. Presentare il metodo `toString` che fornisca la descrizione della navetta (inclusa la descrizione di tutti gli invitati ivi caricati).
4. Possedere il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Navetta` (la verifica va effettuata unicamente sulla targa).
5. Presentare il metodo `aggiungi` che, dato un oggetto `Invitato`, lo inserisca all'interno dell'insieme, controllando che tale inserimento sia possibile.
6. Possedere il metodo `invitato` che, data una stringa, indichi se un invitato con tale nome è presente all'interno della navetta.
7. Possedere il metodo `residui` che indichi il numero di posti ancora disponibili sulla navetta.

Esercizio 6 (8 punti)

Si scriva un'applicazione per Roberto Cipullo che:

1. Crei una lista di oggetti `Navetta`.
2. Crei un oggetto `Navetta`, 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 `Invitato`, lette da tastiera le informazioni necessarie.
5. Inserisca l'oggetto di cui al punto 4. all'interno della prima navetta, tra quelle contenute nella lista di cui al punto 1., in grado di contenerlo, controllando che tale navetta esista.
6. Letto da tastiera un nome di un invitato, stampi a video la targa della navetta che ospita tale invitato (si supponga ne esista solo una).
7. Stampi a video la descrizione della navetta avente il maggior numero di posti disponibili.

Soluzione Esercizio 2

$-46_{10} = 11010010_2$
$-82_{10} = 10101110_2$

$$\begin{array}{r} 11010010+ \\ \underline{10101110} \\ 10000000 \end{array}$$

$10000000_2 = -128_{10}$
Il numero rappresentato è uguale a quello atteso.

Soluzione Esercizio 3

Domanda 1:

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

Domanda 2:

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

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

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

Soluzione Esercizio 4

```
class Invitato implements Comparable<Invitato> {
  private String nome;
  private int stanza, invitati;

  public Invitato(String nome, int stanza, int invitati) {
    this.nome = nome;
    this.stanza = stanza;
    this.invitati = invitati;
  }

  public String getNome() { return nome; }
  public int getStanza() { return stanza; }
  public int getInvitati() { return invitati; }

  public String toString() {
    return nome + ", " + stanza + " (" + invitati + ")";
  }

  public boolean equals(Object o) { return equals((Invitato) o); }
  public boolean equals(Invitato i) {
    return this.nome.equals(i.nome) && this.stanza==i.stanza;
  }

  public int compareTo(Invitato i) {
    int ret = this.nome.compareTo(i.nome);
    if(ret==0) ret = this.invitati - i.invitati;
    return ret;
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;

class Navetta {
  private String targa;
  private int capienza;
  private Set<Invitato> s;

  public Navetta(String targa, int capienza) {
    this.targa= targa;
    this.capienza = capienza;
    s = new HashSet<Invitato>();
  }

  public String getTarga() { return targa; }
  public String toString() { return targa+ "(" + residui() + ": " + s;}
  public boolean equals(Object o) { return equals((Navetta) o); }
  public boolean equals(Navetta n){ return this.targa.equals(n.targa); }
  public boolean aggiungi(Invitato i) {
    if(i.getInvitati()<=residui() && s.add(i)) {
      capienza -= i.getInvitati();
      return true;
    }
    return false;
  }

  public boolean invitato(String nome) {
    for(Invitato i: s) if(i.getNome().equals(nome)) return true;
    return false;
  }

  public int residui() { return capienza; }
}
```

Soluzione Esercizio 6

```
import java.util.*;

class Applicazione {
  public static void main(String[] args) {
    List<Navetta> l = new ArrayList<Navetta>();
    Scanner scanner = new Scanner(System.in);
    Navetta n = new Navetta(scanner.next(), scanner.nextInt());
    l.add(n);
    Invitato i = new Invitato(scanner.nextLine(), scanner.nextInt(),
        scanner.nextInt());
    int j=0;
    while(j<l.size() && !l.get(j).aggiungi(i)) j++;
    if(j==l.size()) System.out.println("Inserimento non avvenuto!");
    String nome=scanner.nextLine();
    for(Navetta x: l) if(x.invitato(nome)) System.out.println(x);
    Navetta max = null;
    for(Navetta x: l) if(max==null || x.residui()>max.residui()) max=x;
    System.out.println(max.toString());
  }
}
```