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

Appello del 17/10/2014

Compito A

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 insiemi e liste in Java.

Esercizio 2 (2 punti)

Convertire in binario i numeri **86** e **53**, 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 N) {
  int i=N, sum=0;
  while(--i>0)
    sum+=V[i--];
  return sum;
}

public static int g(int V[], int N) {
  int j=0, sum=0;
  do
    sum+=f(V, ++j);
  while (j++<N);
  return sum;
}
```

1. Calcolare la complessità in passi base del metodo `f` nei termini del parametro `N` (si distinguano i casi in cui `N` assume valori pari da quelli in cui assume valori dispari).
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)

Per fronteggiare la possibile pandemia del virus "Motaba" (dal nome dello stato africano in cui tale virus ha avuto origine), l'ospedale Major della capitale dello stato caraibico di St. Marquez ha deciso di informatizzare la gestione dei propri pazienti. A tal scopo, per ogni paziente vengono memorizzati nome e cognome, data di ricovero e una diagnosi preliminare (una semplice stringa). Si scriva una classe `Paziente` per l'ospedale Major 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 paziente.
4. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Paziente` (la verifica va fatta su nome e cognome).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Paziente` passato come parametro (la precedenza va data al paziente ricoverato da più tempo).

Esercizio 5 (7 punti)

Si scriva una classe `Corsia` che memorizzi le informazioni relative ai pazienti ricoverati in una corsia dell'ospedale Major. Per ciascuna corsia occorre memorizzare il piano e il cognome del primario, mentre i pazienti ricoverati vanno memorizzati all'interno di una lista. La classe `Corsia` deve:

1. Presentare un opportuno costruttore con parametri (inizialmente, la lista di pazienti 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 corsia (inclusa la descrizione di tutti i pazienti ricoverati).
4. Possedere il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Corsia` (esiste una sola corsia per piano).
5. Presentare il metodo `aggiungi` che, dato un oggetto `Paziente`, lo inserisca all'interno della lista, mantenendo la lista ordinata secondo il punto 5. dell'esercizio 4.
6. Possedere il metodo `infetti` che restituisca un insieme contenente tutti i pazienti la cui diagnosi corrisponde alla stringa passata come parametro.
7. Presentare il metodo `degente` che, dati un nome e un cognome, indichi se un paziente con tali generalità è ricoverato in corsia.

Esercizio 6 (8 punti)

Si scriva un'applicazione per l'ospedale Major che:

1. Crei un insieme di oggetti `Corsia`.
2. Crei un oggetto `Corsia`, 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 `Paziente`, lette da tastiera le informazioni necessarie.
5. Inserisca il paziente creato al punto 4. tra quelli ricoverati nella corsia di cui al punto 2.
6. Letti da tastiera un nome ed un cognome, stampi a video, se esiste, il piano della corsia in cui tale paziente è ricoverato (si supponga non esistano omonimie).
7. Tra tutti i pazienti ricoverati nelle corsie dell'insieme di cui al punto 1., stampi a video le informazioni del paziente la cui diagnosi corrisponde a "Motaba" ricoverato da più tempo.

Soluzione Esercizio 2

$86_{10} = 01010110_2$
$53_{10} = 00110101_2$

$10001011_2 = -117_{10}$

```
01010110+
00110101
10001011
```

Il numero è diverso dal risultato atteso (139), in quanto quest'ultimo non è rappresentabile con soli 8 bit.

Soluzione Esercizio 3

Domanda 1:

2 assegnamenti	2	o 2
`--i>0N`	N/2 + 1	o (N + 1)/2
`sum+=V[i--]`	N/2	o (N – 1)/2
Totale	N + 3	o N + 2

Domanda 2:

2 assegnamenti	2
`sum+=f(V, ++j)`	(N + 1)/2
`j++<N`	(N + 1)/2
complessità di `f`	$N^2/4 + 3N/2 + 5/4$
Totale	$N^2/4 + 5N/2 + 17/4$

complessità di `f`: $\sum_{\substack{j=1 \\ (j \text{ dispari})}}^{N} (j+2) = \sum_{i=0}^{(N-1)/2} (2i+3) = \frac{N^2}{4} + \frac{3N}{2} + \frac{5}{4}$

Domanda 3:
Complessità asintotica: $O(N^2)$

Soluzione Esercizio 4

```
class Paziente implements Comparable<Paziente> {
  private String nome, cognome, diagnosi;
  private int g, m, a;

  public Paziente(String nome, String cognome, String diagnosi, int g,
      int m, int a) {
    this.nome = nome;
    this.cognome = cognome;
    this.diagnosi = diagnosi;
    this.g = g; this.m = m; this.a = a;
  }

  public String getData() { return g + "/" + m + "/" + a; }
  public String getDiagnosi() { return diagnosi; }
  public String getNome() { return nome; }
  public String getCognome() { return cognome; }

  public String toString() {
    return nome+ " " + cognome + "(" + getData()+ "):" + diagnosi;
  }

  public boolean equals(Object o) { return equals((Paziente) o); }
  public boolean equals(Paziente p) {
    return nome.equals(p.nome) && cognome.equals(p.cognome);
  }

  public int compareTo(Paziente p) {
    int ret = this.a - p.a;
    if(ret==0) ret = this.m - p.m;
    if(ret==0) ret = this.g - p.g;
    return ret;
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;

class Corsia {
  private List<Paziente> l;
  private String primario;
  private int piano;

  public Corsia(String primario, int piano) {
    this.primario = primario;
    this.piano = piano;
    l = new LinkedList<Paziente>();
  }

  public String getPrimario() { return primario; }
  public int getPiano() { return piano; }

  public String toString() {
    return primario + "( " + piano + "):" + l.toString();
  }
  public boolean equals(Object o) { return equals((Corsia) o); }
  public boolean equals(Corsia c) { return piano==c.piano; }

  public void aggiungi(Paziente p) {
    int i = 0;
    while((i<l.size())&&(l.get(i).compareTo(p)<0)) i++;
    l.add(i, p);
  }
  public Set<Paziente> infetti(String diagnosi) {
    Set<Paziente> s = new HashSet<Paziente>();
    for(Paziente x:l) if(x.getDiagnosi().equals(diagnosi)) s.add(x);
    return s;
  }
  public boolean degente(String nome, String cognome) {
    Paziente x = new Paziente(nome, cognome, "", 0, 0, 0);
    return l.contains(x);
  }
}
```

Soluzione Esercizio 6

```
import java.util.*;

class Applicazione {
  public static void main(String[] args) {
    Set<Corsia> s=new TreeSet<Corsia>();
    Scanner scanner=new Scanner(System.in);
    Corsia c=new Corsia(scanner.nextLine(), scanner.nextInt());
    if(!s.add(c)) System.out.println("Corsia già esistente!");
    Paziente p=new Paziente(scanner.nextLine(), scanner.nextLine(),
      scanner.nextLine(), scanner.nextInt(), scanner.nextInt(),
      scanner.nextInt());
    c.aggiungi(p);
    String nome = scanner.nextLine(), cognome = scanner.nextLine();
    for(Corsia x: s) if(x.degente(nome,cognome))
      System.out.println(x.getPiano());
    Paziente max = null;
    for(Corsia x: s)
      for(Paziente t: x.infetti("Motaba"))
        if(max==null || t.compareTo(max)<0) max=t;
    System.out.println(max.toString());
  }
}
```

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

Appello del 18/10/2013

Compito B

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 spaziale di un algoritmo.
2. Descrivere le principali differenze tra array e collezioni in Java.

Esercizio 2 (2 punti)

Convertire in binario i numeri **-83** e **–56**, 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 N) {
  int i=N, sum=0;
  while(i-->1)
    sum+=V[--i];
  return sum;
}

public static int g(int V[], int N) {
  int j=0, sum=0;
  do
    sum+=f(V, j++);
  while (++j<N);
  return sum;
}
```

1. Calcolare la complessità in passi base del metodo `f` nei termini del parametro `N` (si distinguano i casi in cui `N` assume valori pari da quelli in cui assume valori dispari).
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)

Per fronteggiare la possibile pandemia del virus "Motaba" (dal nome dello stato africano in cui tale virus ha avuto origine), l'aeroporto internazionale ElFraile della capitale dello stato caraibico di St. Marquez ha deciso di informatizzare la gestione dei passeggeri dei voli in arrivo. A tal scopo, per ogni passeggero vengono memorizzati nome e cognome, data di nascita e nazionalità (una semplice stringa). Si scriva una classe `Passeggero` per l'aeroporto ElFraile 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 passeggero.
4. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Passeggero` (la verifica va fatta su nome e cognome).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Passeggero` passato come parametro (in ordine crescente di età).

Esercizio 5 (7 punti)

Si scriva una classe `Volo` che memorizzi le informazioni relative ai passeggeri di un volo in arrivo all'aeroporto. Per ciascun volo occorre memorizzare il codice (es. "NE143", "NE751", ecc.) e l'aeroporto di provenienza, mentre i passeggeri vanno memorizzati all'interno di una lista. La classe `Volo` deve:

1. Presentare un opportuno costruttore con parametri (inizialmente, la lista dei passeggeri 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 del volo (inclusa la descrizione di tutti i passeggeri).
4. Possedere il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Volo` (il codice del volo è univoco).
5. Presentare il metodo `aggiungi` che, dato un oggetto `Passeggero`, lo inserisca all'interno della lista, mantenendo la lista ordinata secondo il punto 5. dell'esercizio 4.
6. Possedere il metodo `aRischio` che restituisca un insieme contenente tutti i passeggeri la cui nazionalità corrisponde alla stringa passata come parametro.
7. Presentare il metodo `presente` che, dati un nome e un cognome, indichi se un passeggero con tali generalità è presente sul volo.

Esercizio 6 (8 punti)

Si scriva un'applicazione per l'aeroporto ElFraile che:

1. Crei un insieme di oggetti `Volo`.
2. Crei un oggetto `Volo`, 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 `Passeggero`, lette da tastiera le informazioni necessarie.
5. Inserisca il passeggero creato al punto 4. tra quelli a bordo dell'aereo di cui al punto 2.
6. Letti da tastiera un nome ed un cognome, stampi a video, se esiste, il codice del volo su cui si trova tale passeggero (si supponga non esistano omonimie).
7. Tra tutti i passeggeri sui voli dell'insieme di cui al punto 1., stampi a video le informazioni del passeggero più anziano la cui nazionalità corrisponde a "Motaba".

Soluzione Esercizio 2

$-83_{10} = 10101101_2$
$-56_{10} = 11001000_2$

$$\begin{array}{r} 10101101+ \\ \underline{11001000} \\ 01110101 \end{array}$$

$01110101_2 = 117_{10}$

Il numero è diverso dal risultato atteso (–139), in quanto quest'ultimo non è rappresentabile con soli 8 bit.

Soluzione Esercizio 3

Domanda 1:

2 assegnamenti	2	o 2
`i-->0`	N/2 + 1	o (N + 1)/2
`sum+=V[--i]`	N/2	o (N – 1)/2
Totale	N + 3	o N + 2

Domanda 2:

2 assegnamenti	2
`sum+=f(V, j++)`	(N + 1)/2
`++j<N`	(N + 1)/2
complessità di `f`	$N^2/4 + 3N/2 + 5/4$
Totale	$N^2/4 + 5N/2 + 17/4$

complessità di `f`: $\sum_{\substack{j=0 \\ (j \text{ pari})}}^{N-1} (j+3) = \sum_{i=0}^{(N-1)/2} (2i+3) = \frac{N^2}{4} + \frac{3N}{2} + \frac{5}{4}$

Domanda 3:

Complessità asintotica: $O(N^2)$

Soluzione Esercizio 4

```
class Passeggero implements Comparable<Passeggero> {
  private String nome, cognome, nazione;
  private int g, m, a;

  public Passeggero(String nome, String cognome, String nazione, int g,
      int m, int a) {
    this.nome = nome;
    this.cognome = cognome;
    this.nazione = nazione;
    this.g = g; this.m = m; this.a = a;
  }

  public String getData() { return g + "/" + m + "/" + a; }
  public String getNazione() { return nazione; }
  public String getNome() { return nome; }
  public String getCognome() { return cognome; }

  public String toString() {
    return nome+ " " + cognome + "(" + getData()+ "):" + nazione;
  }

  public boolean equals(Object o) { return equals((Passeggero) o); }
  public boolean equals(Passeggero p) {
    return nome.equals(p.nome) && cognome.equals(p.cognome);
  }

  public int compareTo(Passeggero p) {
    int ret = p.a - this.a;
    if(ret==0) ret = p.m - this.m;
    if(ret==0) ret = p.g - this.g;
    return ret;
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;

class Volo {
  private List<Passeggero> l;
  private String codice, aeroporto;

  public Volo(String codice, String aeroporto) {
    this.codice = codice;
    this.aeroporto = aeroporto;
    l = new LinkedList<Passeggero>();
  }

  public String getCodice() { return codice; }
  public String getAeroporto() { return aeroporto; }

  public String toString() {
    return codice + "( " + aeroporto + "):" + l.toString();
  }

  public boolean equals(Object o) { return equals((Volo) o); }
  public boolean equals(Volo v) { return codice.equals(v.codice); }

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

  public Set<Passeggero> aRischio(String nazione) {
    Set<Passeggero> s = new HashSet<Passeggero>();
    for(Passeggero x:l) if(x.getNazione().equals(nazione)) s.add(x);
    return s;
  }

  public boolean presente(String nome, String cognome) {
    Passeggero x = new Passeggero(nome, cognome, "", 0, 0, 0);
    return l.contains(x);
  }
}
```

Soluzione Esercizio 6

```
import java.util.*;

class Applicazione {
  public static void main(String[] args) {
    Set<Volo> s=new TreeSet<Volo>();
    Scanner scanner=new Scanner(System.in);
    Volo v=new Volo(scanner.next(), scanner.nextLine());
    if(!s.add(v)) System.out.println("Volo già esistente!");
    Passeggero p=new Passeggero(scanner.nextLine(), scanner.nextLine(),
      scanner.nextLine(), scanner.nextInt(), scanner.nextInt(),
      scanner.nextInt());
    v.aggiungi(p);
    String nome = scanner.nextLine(), cognome = scanner.nextLine();
    for(Volo x: s) if(x.presente(nome,cognome))
      System.out.println(x.getCodice());
    Passeggero max = null;
    for(Volo x: s)
      for(Passeggero t: x.aRischio("Motaba"))
        if(max==null || t.compareTo(max)>0) max=t;
    System.out.println(max.toString());
  }
}
```