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

Appello del 20/9/2006

Esercizio 1 (4 punti)

Gestione ed utilizzo di collezioni ed iteratori in Java.

Esercizio 2 (6 punti)

Siano date le seguenti funzioni C:

```
int f(int V[], int M, int N) {
  int i;
  int res=0;

  for(i=M; i<N; i++)
    res+=V[i];
  return res;
}

 void g(int U[], int V[], int N) {
  int i=0;

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

1. Calcolare la complessità in passi base della funzione `f` nei termini dei parametri `M` e `N`.
2. Calcolare la complessità in passi base della funzione `g` nei termini del parametro `N`.
3. Calcolare la complessità asintotica della funzione `g` nei termini del parametro `N`.

Esercizio 3 (9 punti)

La direzione dell'Hotel California ha deciso di informatizzare la prenotazione delle camere dell'albergo. Ogni prenotazione deve memorizzare, oltre al nominativo del cliente, le date della prima e dell'ultima notte (che è sicuramente non inferiore alla prima).
Si scriva una classe `Prenotazione` per l'Hotel California che:

1. Possieda un opportuno costruttore con parametri.
2. Presenti il metodo `getNome` che permetta di accedere al nominativo del cliente.
3. Possieda due metodi `getDataArrivo` e `getDataPartenza` che restituiscano una descrizione testuale delle rispettive date nel formato "anno-mese-giorno".
4. Possieda un metodo `sovrapposta` che, data un'altra prenotazione, stabilisca se le due prenotazioni sono sovrapposte, ovvero se esistono giorni in comune.
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` che effettui il confronto con un'altra prenotazione, controllando se una prenotazione precede l'altra o se sono (anche parzialmente) sovrapposte (nel qual caso le prenotazioni sono da considerarsi equivalenti). (Suggerimento: si utilizzino i metodi definiti al punto 3. ed il confronto tra stringhe)

Esercizio 4 (7 punti)

Si scriva una classe `Stanza` che memorizzi i dati di una camera dell'albergo. Oltre al numero della stanza ed al tipo (singola, doppia o matrimoniale), occorre memorizzare le informazioni sulle prenotazioni effettuate per tale camera all'interno di una lista. Ogni `Stanza` deve:

1. Possedere un opportuno costruttore con parametri (inizialmente la stanza non ha prenotazioni).
2. Presenti opportuni metodi che permettano di accedere alle variabili di istanza della classe.
3. Presentare il metodo `aggiungi` che, data una `Prenotazione`, la inserisca nella lista, mantenendo ordinata la lista secondo il criterio espresso al punto 5. dell'esercizio 3. (suggerimento: si utilizzi il metodo `add(int i, Prenotazione m)` della classe `List<Prenotazione>`). Il metodo deve inoltre controllare che la prenotazione da inserire non sia sovrapposta con le altre prenotazioni della stanza.
4. Possedere un metodo `occupata` che, ricevuta in ingresso una data, indichi se tale stanza è occupata o meno in quella data.
5. Presentare un metodo `prenotazioni` che, dato il nome di un cliente, restituisca l'insieme delle prenotazioni di tale cliente per la stanza.
6. Possedere un metodo `equals` per stabilire l'uguaglianza con un'altra stanza (l'uguaglianza va verificata unicamente sul numero).

Esercizio 5 (5 punti)

Si scriva un'applicazione per l'Hotel California che:

1. Crei un insieme di oggetti di tipo `Stanza`.
2. Lette da tastiera le informazioni relative ad una nuova prenotazione (data prima ed ultima notte, nome del cliente) ed il tipo di stanza desiderata, provveda ad effettuare la prenotazione stampando a video il numero della stanza trovata o un messaggio di errore in caso non esistano stanze del tipo richiesto disponibili nel periodo indicato.
3. Letta da tastiera una data, stampi a video il numero di tutte le stanze libere in tale data (ovvero che non hanno prenotazioni).

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.
- `double leggiDouble()` Legge un numero razionale (delimitato da spazi).
- `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:

2 assegnamenti	2
`for` eseguito $N - M$ volte	$N - M + 1$
incremento di `i`	$N - M$
`res+=V[i]`	$N - M$
Totale	$3(N - M) + 3$

Domanda 2:

1 assegnamento	1
`while` eseguito $N - 1$ volte	N
assegnamento	$N - 1$
complessità di `f`	$3/2\ N(N + 1) - 3$
Totale	$3/2\ N^2 + 7/2\ N - 3$

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

Soluzione Esercizio 3

```
class Prenotazione implements Comparable {
  private int ga, ma, aa, gp, mp, ap;
  private String nome;

  public Prenotazione(int gga, int mma, int aaa, int ggp, int mmp, int aap,
      String nome) {
    ga=gga; ma=mma; aa=aaa;
    gp=ggp; mp=mmp; ap=aap;
    this.nome=nome;
  }

  public String getDataArrivo() { return ""+aa+"-"+ma+"-"+ga; }
  public String getDataPartenza() { return ""+ap+"-"+mp+"-"+gp; }
  public String getNome() { return nome; }

  public boolean sovrapposta(Prenotazione p) {
    return(this.compareTo(p)==0); }

  public int compareTo(Prenotazione p) {
    int ret=p.getDataArrivo().compareTo(this.getDataArrivo());
    if(ret<0) {
      if(p.getDataPartenza().compareTo(this.getDataArrivo())<0) return 1;
      return 0;
    }
    else if(ret>0) {
      if(this.getDataPartenza().compareTo(p.getDataArrivo())<0) return -1;
      return 0;
    }
    else return 0;
  }

  public int compareTo(Object o) { return compareTo((Prenotazione) o); }
}
```

Soluzione Esercizio 4

```
import java.util.*;
class Stanza {
  private int numero;
  private String tipo;
  private List<Prenotazione> l;

  public Stanza(int numero, String tipo) {
    this.numero=numero;
    this.tipo=tipo;
    this.l=new ArrayList<Prenotazione>();
  }

  public boolean aggiungi(Prenotazione p) {
    int i=0;
    while(i<l.size() && l.get(i).compareTo(p)<0) i++;
    if(i==l.size() || l.get(i).compareTo(p)>0) {
      l.add(i, p);
      return true;
    }
    else return false;
  }

  public int getNumero() { return numero; }
  public String getTipo() { return tipo; }

  public boolean occupata(int g, int m, int a) {
    Prenotazione p=new Prenotazione(g, m, a, g, m, a, "");
    int i=0;
    while(i<l.size() && l.get(i).compareTo(p)<0) i++;
    if(i==l.size() || l.get(i).compareTo(p)>0) return false;
    return true;
  }

  public Set<Prenotazione> prenotazioni(String nome) {
    Set<Prenotazione> s=new HashSet<Prenotazione>();
    for(Prenotazione p:l) if(p.getNome().equals(nome)) s.add(p);
    return s;
  }

  public boolean equals(Stanza s) { return this.numero==s.numero; }
  public boolean equals(Object o) { return equals((Stanza) o); }
}
```

Soluzione Esercizio 5

```
import java.util.*;
import fiji.io.*;
class Applicazione {
  public static void main(String[] args) {
    Set<Stanza> s=new HashSet<Stanza>(); // domanda 1
    boolean trovata=false;

    Prenotazione p=new Prenotazione(Lettore.in.leggiInt(),
      Lettore.in.leggiInt(), Lettore.in.leggiInt(), Lettore.in. leggiInt(),
      Lettore.in.leggiInt(), Lettore.in.leggiInt(), Lettore.in.leggiLinea());
    String tipo=Lettore.in.leggiString();
    for(Stanza c: s) if(c.getTipo().equals(tipo) && c.aggiungi(p)) {
      System.out.println(c.getNumero());
      trovata=true;
      break;
    }
    if(!trovata) System.out.println("Prenotazione non possibile"); // domanda 2
    int g=Lettore.in.leggiInt(), m=Lettore.in.leggiInt(),
      a=Lettore.in.leggiInt();
    for(Stanza c: s) if(!c.occupata(g, m, a))
      System.out.println(c.getNumero()); // domanda 3
  }
}
```