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

Appello del 16/7/2010

Esercizio 1 (4 punti)

Realizzazione di programmi modulari in C.

Esercizio 2 (6 punti)

Siano dati i seguenti metodi Java:

```
public static int f(int V[],int M) {
  int sum=0;
  for(int i=0; i<M/2; )
    sum+=V[i++];
  return sum;
}

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

1. Calcolare la complessità in passi base del metodo `f` nei termini del parametro `M` (si distinguano i casi in cui `M` assume valori pari e dispari).
2. Calcolare la complessità in passi base del metodo `g` nei termini del parametro `N` (si supponga `N` pari).
3. Calcolare la complessità asintotica del metodo `g` nei termini del parametro `N`.

Esercizio 3 (5 punti)

I progettisti del nuovo gioco di carte collezionabili "Il raduno della magia" hanno deciso di informatizzare la gestione delle carte personaggi del gioco. In particolare, per ogni carta occorre memorizzare il nome del personaggio, la categoria (es., mago/guerriero/fata) ed il punteggio di attacco e di difesa. Si scriva una classe `Carta` per il gioco "Il raduno della magia" che:

1. Possieda un opportuno costruttore con parametri.
2. Presenti opportuni metodi che permettano di accedere alle variabili di istanza dell'oggetto.
3. Presenti il metodo `toString` che fornisca la descrizione del personaggio.
4. Possieda il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Carta` (l'uguaglianza va verificata sul nome e sulla categoria del personaggio).
5. Implementi l'interfaccia `Comparable`, definendo il metodo `compareTo` per stabilire la precedenza con un oggetto `Carta` passato come parametro (la precedenza va verificata per punteggio di attacco decrescente e, in caso di parità, si procede per punteggio di difesa decrescente).

Esercizio 4 (8 punti)

Si scriva una classe `Bustina` che memorizzi le informazioni relative alle carte presenti in ogni bustina. Oltre al codice univoco della bustina, le varie carte vanno inserite all'interno di un insieme. La classe `Bustina` deve inoltre:

1. Presentare un opportuno costruttore (inizialmente, la bustina non contiene alcuna carta).
2. Presentare un metodo che restituisca il codice della bustina.
3. Possedere il metodo `toString` che fornisca la descrizione della bustina (inclusa la descrizione di tutti i personaggi ivi contenuti).
4. Presentare il metodo `equals` per stabilire l'uguaglianza con un altro oggetto `Bustina` (l'uguaglianza va verificata sul codice della bustina).
5. Possedere il metodo `aggiungi` che, dato un oggetto di tipo `Carta`, lo inserisca all'interno dell'insieme, controllando che questo non contenga già un oggetto uguale.
6. Presentare il metodo `cerca` che, dato un nome, indichi se un personaggio con tale nome è contenuto o meno all'interno della bustina.
7. Presentare il metodo `paladino` che, data una categoria di personaggi, restituisca il personaggio di tale categoria con il maggior punteggio difensivo tra quelli presenti all'interno della bustina.

Esercizio 5 (7 punti)

Si scriva un'applicazione per il gioco "Il raduno della magia" che:

1. Crei una lista di oggetti `Bustina`.
2. Crei un oggetto `Bustina`, 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. Letto da tastiera il nome di un personaggio, stampi il codice di tutte le bustine tra quelle all'interno della lista di cui al punto 1. che contengono almeno una carta avente tale nome.
5. Stampi a video la descrizione di tutte le bustine contenute nella lista di cui al punto 1., incluse le descrizioni di tutti i personaggi contenuti.
6. Letta da tastiera una categoria di personaggi, stampi il nome del personaggio di tale categoria con il maggior punteggio difensivo per tutte le bustine nella lista 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:

- `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	
`i<M/2`	`M`/2 + 1	o (`M+1`)/2
`sum+=V[i++]`	`M`/2	o (`M-1`)/2
otale	`M` + 3	o `M` + 2

Domanda 2:

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

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

Domanda 3:

Complessità asintotica: $O(N^2)$

Soluzione Esercizio 3

```
class Carta implements Comparable<Carta> {
  private String nome, categoria;
  private int attacco, difesa;

  public Carta(String nome, String categoria, int attacco, int difesa) {
    this.nome=nome;
    this.categoria=categoria;
    this.attacco=attacco;
    this.difesa=difesa;
  }

  public String getNome() { return nome; }
  public String getCategoria() { return categoria; }
  public int getAttacco() { return attacco; }
  public int getDifesa() { return difesa; }

  public String toString() {
    return nome + "(" + categoria + "): " + attacco + "/" + difesa;
  }

  public boolean equals(Object o) { return equals((Carta) o); }
  public boolean equals(Carta c) {
    return nome.equals(c.nome)&&categoria.equals(c.categoria);
  }

  public int compareTo(Carta c) {
    int ret=c.attacco-this.attacco;
    if(ret==0) ret=c.difesa-this.difesa;
    return ret;
  }
}
```

Soluzione Esercizio 4

```
import java.util.*;

class Bustina {
  private int codice;
  private Set<Carta> s;

  public Bustina(int codice) {
    this.codice=codice;
    s=new HashSet<Carta>();
  }

  public int getCodice() { return codice; }
  public String toString() { return codice+"\n"+s; }

  public boolean equals(Object o) { return equals((Bustina) o); }
  public boolean equals(Bustina b) {  return codice==b.codice; }

  public boolean aggiungi(Carta c) { return s.add(c); }

  public boolean cerca(String nome) {
    for(Carta c:s) if(c.getNome().equals(nome)) return true;
    return false;
  }


  public Carta paladino(String categoria) {
    Carta paladino=null;
    for(Carta c:s)
      if((c.getCategoria().equals(categoria))
        &&((paladino==null)||(c.getDifesa()>paladino.getDifesa())))
          paladino=c;
    return paladino;
  }
}
```

Soluzione Esercizio 5

```
import java.util.*;
import fiji.io.*;

class Applicazione {
  public static void main(String[] args) {
    List<Bustina> l=new LinkedList<Bustina>(); // domanda 1
    Bustina b=new Bustina(Lettore.in.leggiInt());// domanda 2
    l.add(b); // domanda 3
    String nome=Lettore.in.leggiLinea();
    for(Bustina o:l)
      if(o.cerca(nome)) System.out.println(o.getCodice()); // domanda 4
    System.out.println(l.toString()); // domanda 5
    String categoria= Lettore.in.leggiLinea();
    Carta paladino=null;
    for(Bustina o:l) {
      Carta c=o.paladino(categoria);
      if((paladino==null)||(c.getDifesa()>paladino.getDifesa()))
        paladino=c;
    }
    if(paladino!=null) System.out.println(paladino); // domanda 6
  }
}
```