Dipartimento di Informatica e Sistemistica
Antonio Ruberti

"Sapienza" Università di Roma

Esercitazione 6

Corso di Tecniche di programmazione

Laurea in Ingegneria Informatica

(Canale di Ingegneria delle Reti e dei Sistemi Informatici - Polo di Rieti)

Anno Accademico 2007/2008

Tutor: Ing. Diego Rughetti

Esercizio 1 (1)

Si vuole realizzare una classe Java PhotoGallery che rappresenta una galleria di fotografie online. Ogni *PhotoGallery* ha una url (rappresentata da una stringa) dove essa è situata e contiene foto, ciascuna rappresentata semplicemente come una stringa che denota il link alla foto vera e propria. Le funzionalità degli oggetti della classe sono:

- *crea*: che preso come parametro una stringa *url*, crea un oggetto *PhotoGallery* avente come indirizzo *url* e con zero foto memorizzate;
- *numFoto*: che restituisce il numero di foto presenti nella *PhotoGallery*;
- *aggiungi*: che presa una foto (cioè una stringa) come parametro, l'aggiunge alla *PhotoGallery* se non è già presente; altrimenti non fa nulla;
- *elimina*: che presa una foto (cioè una stringa) come parametro, la elimina dalla *PhotoGallery*, se presente; altrimenti non fa nulla;
- *presente*: che presa una foto (cioè una stringa) come parametro, restituisce true se la foto è presente nella *PhotoGallery*; false altrimenti;
- *retituisciTutteLeFoto*: che restituisce un array contenente tutte le foto (rappresentate come stringhe) presenti nella *PhotoGallery*;
- *svuota*: che elimina tutte le foto dalla *PhotoGallery*.

Esercizio 1 (2)

Domanda 1. Si realizzi la classe PhotoGallery.

Domanda 2. Si realizzi una classe cliente della classe PhotoGallery contenente un metodo statico leggiDaFile che, dati una stringa *f* che rappresenta un nome di un file contenente foto (cioè stringhe) ed una *PhotoGallery g*, aggiunga tutte le foto presenti nel file *f* in *g*.

Soluzione (1)

```
class Nodo {
  public String info;
  public Nodo next;
}

public class PhotoGallery {

  // rappresentazione degli oggetti
  private String url;
  private Nodo listafoto;

  // metodi pubblici
  public PhotoGallery(String u) {
    url = u;
    listafoto = null;
  }
```

Soluzione (2)

```
public int numFoto() {
    int cont = 0;
    Nodo p = listafoto;
    while (p != null) {
      cont++;
      p = p.next;
    }
    return cont;
  }

  public void aggiungi(String foto) {
    if (!presente(foto)) {
      Nodo aux = new Nodo();
      aux.info = foto;
      aux.next = listafoto;
      listafoto = aux;
    }
  }
```

Soluzione (3)

```
public String[] restituisciTutteLeFoto() {
  String[] ris = new String[numFoto()];
  int i = 0;
  Nodo p = listafoto;
  while (p != null) {
    ris[i] = p.info;
    i++;
    p = p.next;
  }
  return ris;
}

public void svuota() {
  listafoto = null;
}
```

Soluzione (4)

```
  public void elimina(String foto) {
    if (listafoto == null)   // l'elemento non e' presente e
      return;                // non devo fare nulla
    else if (listafoto.info.equals(foto))
      listafoto = listafoto.next; // ho eliminato il primo
    else {
      Nodo p = listafoto;
      while (p.next != null && !p.next.equals(foto))
        p = p.next;
      if (p.next!=null)
        p.next = p.next.next;
    }
  }
public boolean presente(String foto) {
    Nodo p = listafoto;
    while (p != null) {
      if (p.info.equals(foto))
        return true;
      p = p.next;
    }
    return false;
  }
```

Soluzione (5)

Implementazione con metodi ricorsivi:

```
public void elimina(String foto) {
        listafoto = eliminaAux(listafoto, foto);
}

// metodo ausiliario
  private static Nodo eliminaAux(Nodo p, String foto) {
    if (p == null)
      return null;
    else if (p.info.equals(foto))
      return p.next;
    else {
      p.next = eliminaAux(p.next,foto);
      return p;
    }
  }
```

Soluzione (6)

```
public boolean presente(String foto) {
    return presenteAux(listafoto, foto);
  }

// metodo ausiliario
    private static boolean presenteAux(Nodo p, String foto) {
    if (p == null)
      return false;
    else if (p.info.equals(foto))
      return true;
    else
      return presenteAux(p.next,foto);
  }
```

Soluzione (7)

```
import java.io.*;

public class ClientePhotoGallery {

  public static void leggiDaFile(String nomefile, PhotoGallery g) throws IOException {
    FileReader f = new FileReader(nomefile);
    BufferedReader br = new BufferedReader(f);
    String s = br.readLine();
    while (s != null) {
      g.aggiungi(s);
      s = br.readLine();
    }
    f.close(); // o equivalentemente br.close();
  }
```

Soluzione (8)

```
// metodo non richiesto dalla traccia
public static void scriviSuFile(String nomefile, PhotoGallery g) throws IOException {
    FileWriter f = new FileWriter(nomefile);
    PrintWriter out = new PrintWriter(f);
    String[] arrayfoto = g.restituisciTutteLeFoto();
    for (int i = 0; i < arrayfoto.length; i++)
      out.println(arrayfoto[i]);
    f.close();  //oppure out.close();
  }
```

Esercizio 2 (1)

Si vuole realizzare una classe Nastro in modo tale che ogni oggetto della classe rappresenti un nastro utilizzato per memorizzare programmi televisivi. Di ogni programma sono di interesse le seguenti informazioni:

il nome (una stringa);
la durata in minuti (un intero).

Ogni nastro ha una certa capienza e ad ogni programma memorizzato sul nastro è associata una posizione (iniziando a contare da 0). Della classe sono di interesse le seguenti funzionalità:

- dato un intero *k* come parametro, creazione di un nastro con capienza di *k* minuti, inizialmente vuoto;
- *capienzaResidua*, che restituisce la capienza residua del nastro in minuti;
- *quantiProgrammi*, che restituisce il numero di programmi memorizzati sul nastro;
- *memorizza*, che dati come parametri il nome e la durata di un programma, memorizza il programma nella prossima posizione sul nastro, se la capienza residua del nastro è sufficiente a contenere il programma; se la capienza residua del nastro non è sufficiente, non viene effettuata la memorizzazione;
- *qualeProgramma*, che data una posizione *n* del nastro, restituisce il nome del programma in posizione *n*, se questo esiste, restituisce null altrimenti.
- *qualeDurata*, che data una posizione *n* del nastro, restituisce la durata del programma in posizione *n*, se questo esiste, restituisce 0 altrimenti.

Esercizio 2 (2)

Parte 1. Si realizzi la classe Nastro utilizzando una lista collegata per rappresentare i programmi memorizzati su un nastro.

Parte 2. Si realizzi una classe cliente della classe Nastro, contenente un metodo statico che, presi come parametri un nastro ed il nome di un file, stampa sul file posizione, nome e durata di tutti programmi memorizzati sul nastro, uno per riga. Il metodo deve inoltre stampare sul file la durata complessiva dei programmi memorizzati sul nastro e la capienza residua del nastro.

Soluzione (1)

```
class NodoLista{
    String nome;
    int durata;
    NodoLista next;
    public NodoLista(String n, int d, NodoLista ne) {
      nome = n; durata = d; next = ne;
    }
}

public class Nastro {

  private int capienza;   // capienza residua del nastro
  private int numero;     // numero di programmi memorizzati sul nastro
  private NodoLista primo,ultimo;  // puntatori alla lista

  public Nastro(int k) {
    capienza = k;
    numero = 0;
    // nodo generatore permanente
    primo = new NodoLista(null,0,null);
    ultimo = primo;
  }
```

Soluzione (2)

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

  public int quantiProgrammi() {
    return numero;
  }

  public void memorizza(String nome, int durata) {
    if (capienza >= durata){
      ultimo.next = new NodoLista(nome,durata,null);
      ultimo = ultimo.next;
      numero++;
      capienza = capienza - durata;
    }
  }
```

Soluzione (3)

```
public String qualeProgramma(int n) {
    NodoLista nl = trovaNodo(primo.next, n);
    if (nl==null)
       return null;
    else
       return nl.nome;
  }

  public int qualeDurata(int n) {
    NodoLista nl = trovaNodo(primo.next, n);
    if (nl == null)
       return 0;
    else
       return nl.durata;
  }
```

Soluzione (4)

```
private static NodoLista trovaNodo(NodoLista lis, int pos) {
    if (lis == null)
       return null;
    else if (pos == 0)
      return lis;
    else
      return trovaNodo(lis.next, pos-1);
  }
```

Soluzione (5)

```
import java.io.*;

public class Main {

  public static void main (String[] args) throws IOException {
    Nastro nas = new Nastro(30);
    nas.memorizza("Primo Programma", 5);
    nas.memorizza("Secondo Programma", 10);
    nas.memorizza("Terzo Progamma", 6);
    nas.memorizza("Quarto Programma", 6);
    Cliente.stampaProgrammi(nas, "listaProgrammi.txt");
  }
}
/* Output stampato sul file "listaProgrammi.txt"
0: Primo Programma 5
1: Secondo Programma 10
2: Terzo Programma 6
3: Quarto Programma 6
Durata complessiva: 27
Capienza residua: 3
*/
```

Soluzione (6)

```
import java.io.*;

public class Cliente {

public static void stampaProgrammi(Nastro nas, String nomefile)throws IOException {
    PrintWriter out = new PrintWriter(new FileWriter(nomefile));
    int durataTotale = 0;
    for (int i = 0; i < nas.quantiProgrammi(); i++) {
      int durata = nas.qualeDurata(i);
      durataTotale = durataTotale + durata;
      out.println(i + ": " + nas.qualeProgramma(i) + " " + durata);
    }
    out.println("Durata complessiva: " + durataTotale);
    out.println("Capienza residua: " + nas.capienzaResidua());
    out.close();
  }
}
```