Programmazione
Preparazione al II Compitino

15 gennaio 2015

Cognome **House** Nome **Gregory**
Matricola **123456**

- Nei seguenti quesiti, quando vi è richiesto di scrivere un programma, potete limitarvi al *corpo* del metodo main, assumendo se necessario che in sia una variabile di tipo Scanner, già dichiarate e inizializzate.

1. Scrivete un programma che legga un numero n, seguito da una sequenza di esattamente n stringhe; al termine, deve ristampare la prima metà di ciascuna stringa (per le stringhe di lunghezza dispari, si arrotondi la metà all'intero più piccolo).

   Ecco un esempio di esecuzione (le parti in grassetto sono state inserite dall'utente):

   ```
   Quante stringhe:  4
   Stringa 1:  mamma
   Stringa 2:  pippo
   Stringa 3:  xyp231
   Stringa 4:  urka
   ma
   pi
   xyp
   ur
   ```

(Svolgimento sul retro)

```
System.out.print("Quante stringhe: ");
int n = in.nextInt();
in.nextLine();
String x[] = new String[n];
for (int i=0; i<n; i++) {
 System.out.print("Stringa " + (i+1) + ": ");
 x[i]=in.nextLine();
}
for (int i=0; i<n; i++)
 out.println(x[i].substring(0,x[i].length()/2));
```

2. Scrivete un programma che legga un numero n, seguito da una sequenza di esattamente n stringhe; al termine, deve stampare le stringhe non ripetute.

   Ecco un esempio di esecuzione (le parti in grassetto sono state inserite dall'utente):

```
Quante stringhe:  6
Stringa 1:  genuflesso
Stringa 2:  mamma
Stringa 3:  pippo
Stringa 4:  mamma
Stringa 5:  ramoso
Stringa 6:  genuflesso
pippo
ramoso
```

```
System.out.print("Quante stringhe: ");
int n = in.nextInt();
in.nextLine();
String x[] = new String[n];
for (int i=0; i<n; i++) {
 System.out.print("Stringa " + (i+1) + ": ");
 x[i]=in.nextLine();
}
for (int i=0; i<n; i++) {
 int j;
 for (j=0; j<n; j++)
  if (i!=j && x[i].equals(x[j]))
   break;
 if (j==n)
  out.println(x[i]);
}
```

3. Supponete di avere una classe astratta di nome Contatore le cui istanze rappresentano dei contatori. Questa classe (che ha un costruttore senza argomenti) prevede un metodo valore() che restituisce il valore attuale del contatore, e un metodo aggiorna() che aggiorna il contatore secondo regole che dipendono dalla concreta implementazione. La classe ha due sottoclassi concrete:

   - la classe ContatoreAdditivo, in cui ogni aggiornamento del contatore aggiunge una quantità costante al contatore stesso; la classe ha un costruttore ContatoreAdditivo(**int** valoreIniziale , **int** incremento) che specifica il valore iniziale e l'incremento di cui il contatore viene aumentato ogni volta;
   - la classe ContatoreMoltiplicativo, in cui ogni aggiornamento del contatore moltiplica il contatore stesso per una quantità costante; la classe ha un costruttore ContatoreMoltiplicativo(**int** valoreIniziale , **int** fattore) che specifica il valore iniziale e il fattore per cui il contatore viene moltiplicato ogni volta.

   Implementate nella classe Contatore un metodo toString che restituisca una stringa della forma "valore=XXX dove XXX sia il valore attuale del contatore.

```
@Override
public String toString() {
 return "valore=" + valore();
}
```

4. Scrivete una classe, di nome ContatoreDoppio che funziona come segue:

   - il suo costruttore ContatoreDoppio(Contatore c) costruisce un contatore doppio che ha il contatore specificato come contatore *soggiacente*;
   - il valore del contatore doppio è, in ogni istante, quello del contatore soggiacente;
   - ogni volta che il contatore doppio viene aggiornato, questo scatena due aggiornamenti consecutivi del contatore soggiacente.

```
public class ContatoreDoppio extends Contatore {
 private Contatore soggiacente;

 public ContatoreDoppio(Contatore c) {
  soggiacente=c;
 }

 @Override
 public void aggiorna() {
  soggiacente.aggiorna();
  soggiacente.aggiorna();
 }

 @Override
 public int valore() {
  return soggiacente.valore();
 }
}
```

5. Considerate la classe Mail i cui oggetti corrispondono a dei *messaggi di posta elettronica*; questa classe ha un costruttore pubblico

   - public Mail(String mittente, String[] destinatari, String titolo, String testo)

   che costruisce un messaggio inviato dall'indirizzo mittente a una certa lista di indirizzi destinatari, e con un certo titolo testo. La classe possiede, fra gli altri, i metodi

   - public String getMittente()
   - public boolean eFraIDestinatari(String dest)
   - public int getNumeroDestinatari()

   che, rispettivamente, forniscono il mittente, dicono se un dato indirizzo compare fra i destinatari e forniscono il numero di destinatari.

   Scrivete un programma che operi come segue:

   (a) chieda all'utente di inserire un numero intero, diciamo n;

   (b) dichiari un array di n mail e lo riempia con mail inserite dall'utente;

   (c) stampi i soli messaggi che contengano `rossi@yahoo.com` fra i destinatari e siano inviati a non più di 2 persone (la classe Mail dispone di un opportuno metodo toString()).

   Ecco un esempio di esecuzione (le parti in grassetto sono state inserite dall'utente):

```
Inserisci un intero: 4

Mittente: xxx@valium.it
Quanti destinatari: 2
Destinatario: aaa@vermeschifoso.com
Destinatario: rossi@yahoo.com
Titolo: Appuntamento
Testo: Ci vediamo domani alle otto

Mittente: bbb@valium.it
Quanti destinatari: 3
Destinatario: ccc@magna.magna.it
Destinatario: rossi@yahoo.com
Destinatario: bbb@magna.magna.it
Titolo: Riunione
Testo: La riunione sarà nella sala rossa

Mittente: bbb@valium.it
Quanti destinatari: 1
```

Destinatario: **rossi@yahoo.com**
Titolo: **Riunione 2**
Testo: **La riunione sarà nella sala grigia (non rossa!)**

Mittente: **bbb@valium.it**
Quanti destinatari: **1**
Destinatario: **aaa@vermeschifoso.com**
Titolo: **Baci**
Testo: **Vi voglio molto bene**

‘‘Appuntamento (da xxx@valium.com)
‘‘Riunione 2 (da bbb@valium.com)

```
int n = in.nextInt("Inserisci un intero: ");
in.nextLine();
Mail x[] = new Mail[n];
for (int i=0; i<n; i++) {
 System.out.print("Mittente: ");
 String mitt=in.nextLine();
 System.out.print("Quanti destinatari: ");
 int ndest=in.nextInt();
 in.nextLine();
 String dest[] = new String[ndest];
 for (int j=0; j<ndest; j++) {
  System.out.print("Destinatario: ");
  dest[j]=in.nextLine();
 }
 System.out.print("Titolo: ");
 String tit=in.nextLine();
 System.out.print("Testo: ");
 String txt=in.nextLine();
 x[i]=new Mail(mitt,dest,tit,txt);
}
for (int i=0; i<n; i++)
 if (x[i].eFraIDestinatari("rossi@yahoo.com") &&
     x[i].getNumeroDestinatari()<=2)
  out.println(x[i]);
```

6. Considerate la seguente funzione definita ricorsivamente sugli interi:

$$f(x) = \begin{cases} f(x+1) - f(x-1) & \text{se } x \text{ 'e pari} \\ x & \text{altrimenti} \end{cases}$$

Scrivete un metodo statico con intestazione

```
public static int f( int x )
```

per il calcolo di f.

```
public static int f(int x) {
 if (x%2==0) return f(x+1)-f(x-1);
 else return x;
}
```

7. Scrivete la classe Mail con i seguenti metodi e costruttori:

   - **public** Mail(String mitt, String[] dest, String titolo, String testo): costruisce una mail con dato mittente, destinatari, titolo e testo;
   - **public** String getMittente(): restituisce il mittente;
   - **public boolean** eFraIDestinatari(String dest): restituisce **true** se e solo se dest compare fra i destinatari;
   - **public int** getNumeroDestinatari(): restituisce il numero dei destinatari;
   - **public** String getDestinatario(**int** i) **throws** IllegalArgumentException: restituisce l'i-esimo destinatario (con i compreso fra 0 e $n-1$, dove n è il numero di destinatari); se il valore i è negativo o maggiore di $n-1$, il metodo deve sollevare una IllegalArgumentException.

```
public class Mail {
 private String mitt;
 private String[] dest;
 private String tit;
 private String txt;

 public Mail(String mitt, String[] dest, String tit, String txt) {
  this.mitt=mitt;
  this.dest=dest;
  this.tit=tit;
  this.txt=txt;
 }

 public String getMittente() {
  return mitt;
 }

 public boolean eFraIDestinatari(String d) {
  for (int i=0; i<dest.length; i++)
   if (d.equals(dest[i])) return true;
  return false;
 }

 public int getNumeroDestinatari() {
  return dest.length;
 }

 public String getDestinatario(int i) throws IllegalArgumentException {
  if (i<0 || i>=dest.length)
   throw new IllegalArgumentException();
  return dest[i];
 }
}
```

8. Definite una classe di nome MailCC che estende Mail e i cui oggetti corrispondono a delle e-mail che hanno anche dei destinatari in CC (cioè, che ricevono la e-mail solo per conoscenza). Questa classe deve avere:

   - un costruttore pubblico

     `public MailCC(String mitt,String[] dest,String[] cc,String tit,String txt)`

     che specifica oltre ai dati di una normale e-mail anche l'elenco dei destinatari in CC;
   - un metodo

     `public int numeroCC()`

     che restituisce il numero di destinatari in CC;
   - il metodo ereditato eFraIDestinatari(String dest) deve restituire **true** anche se dest compare fra i destinatari in CC.

```
public class MailCC extends Mail {
 private String[] destCC;

 public MailCC(String mitt,String[] dest,String[] cc,String tit,String txt) {
  super(mitt,dest,tit,txt);
  this.destCC=cc;
 }

 public int numeroCC() {
  return destCC.length;
 }

 @Override
 public boolean eFraIDestinatari(String d) {
  if (super.eFraIDestinatari(d)) return true;
  for (int i=0; i<destCC.length; i++)
   if (d.equals(destCC[i])) return true;
  return false;
 }
}
```