Dipartimento di Informatica e Sistemistica
Antonio Ruberti

“Sapienza” Università di Roma

Esercitazione 3

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

Vogliamo definire un'interfaccia *Confrontabile* che offra un'operazione che verifica se un oggetto è maggiore di un altro, ed una operazione che verifica se un oggetto è paritetico ad un altro.

Scrivere poi un metodo che dati tre riferimenti a *Confrontabile* restituisca il maggiore tre i tre (o meglio uno qualunque che non abbia tra gli altri due uno maggiore di esso).

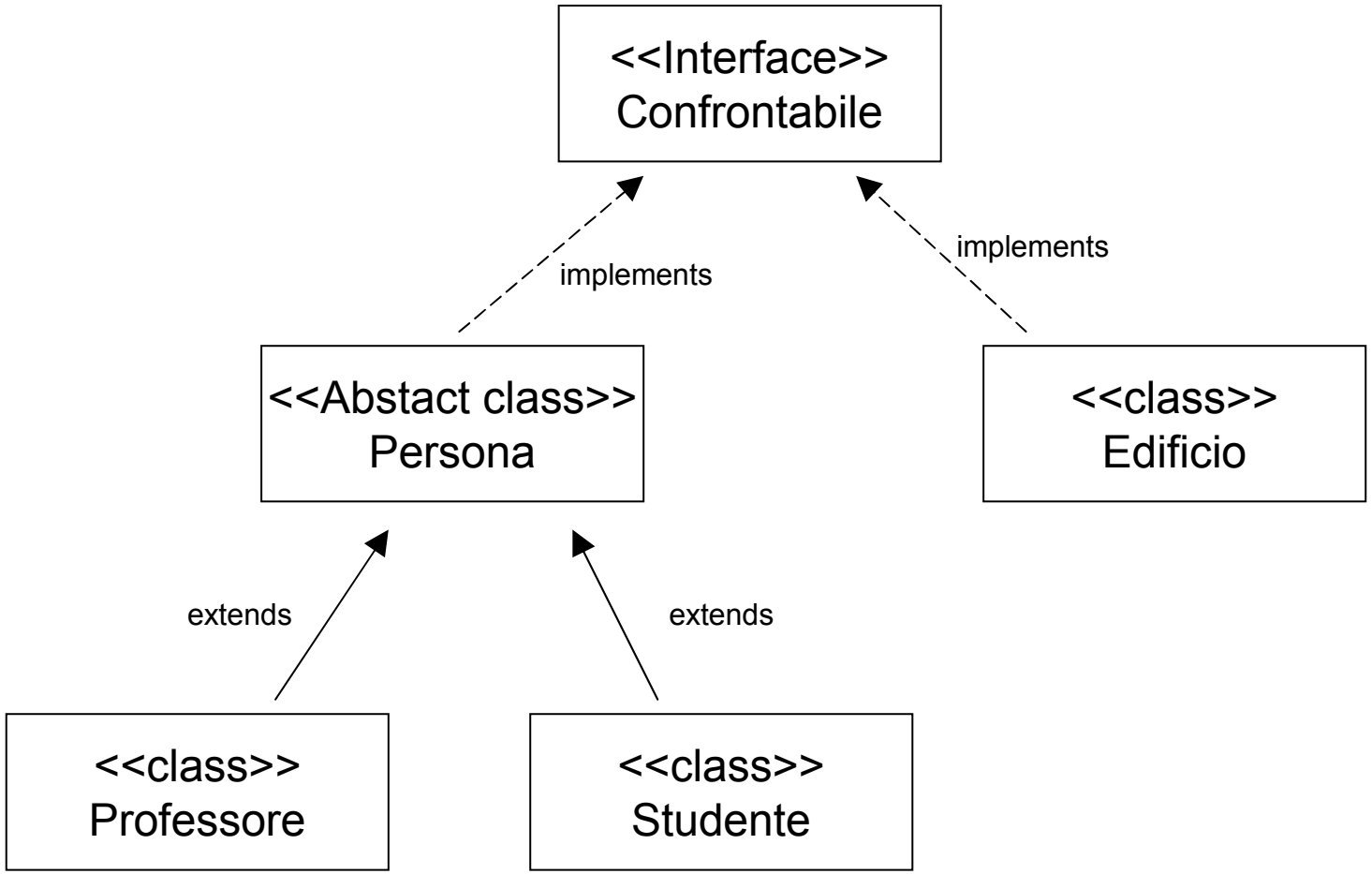
Definire infine due classi che implementano l'interfaccia *Confrontabile:*

1. Classe astratta persona(per la quale il confronto concerne un valore ritornato da un metodo aliquota())
2. Classe edificio (per la quale il confronto concerne l'altezza)

Il metodo aliquota() è un metodo che ritorna l'aliquota fiscale dell'oggetto persona su cui è invocato. Per le persone che svolgono il lavoro di professore l'aliquota è va da 40 a 50, per le persone che invece svolgono l'attività di studenti l'aliquota va da 10 a 20.

Fornire infine un main di prova

Soluzione


<<Interface>>
Confrontabile
implements
implements
<<Abstact class>>
Persona
<<class>>
Edificio
extends
extends
<<class>>
Professore
<<class>>
Studente

Soluzione (2)

```
interface Confrontabile{
        boolean maggiore(Confrontabile x);
        boolean paritetico(Confrontabile x);
}

class Utilita{
    public static Confrontabile MaggioreTraTre(Confrontabile x1,
                                                Confrontabile x2, Confrontabile x3){
        if((x1.maggiore(x2)||x1.paritetico(x2)) && (x1.maggiore(x3)||x1.paritetico(x3)))
                        return x1;
        else if((x2.maggiore(x1)||x2.paritetico(x1)) && (x2.maggiore(x3)||x2.paritetico(x3)))
                return x2
        else
                return x3;
    }
}
```

Soluzione (3)

```
abstract class Persona implements Confrontabile{
        private int eta;
        public Persona(int e){
                this.eta = e;
        }
        abstract public int aliquota();
        public int Eta(){
                return eta;
        }
        public boolean maggiore(Confrontabile e){
                return aliquota() > ((Persona)e).aliquota();
        }
        public boolean paritetico(Confrontabile e){
                return aliquota() == ((Persona)e).aliquota();
        }
}
```

Soluzione (4)

```
class Edificio implements Confrontabile{
        protected int altezza;
        public Edificio(int i){
                this.altezza = i;
        }
        public boolean maggiore(Confrontabile e){
                return this.altezza > ((Edificio)e).altezza;
        }
        public boolean paritetico(Confrontabile e){
                return altezza == ((Edificio) e).altezza;
        }
}
public class Professore extends Persona{
        private int aliq;
        public Professore(int e, int a) throws Exception{
                super(e);
                if(a>=40 && a<=50)
                        this.aliq = a;
                else
                        throw new Exception("Valore dell'aliquota immesso errato!");
        }
        public int aliquota(){
                return aliq;
        }

}
```

Soluzione (5)

```
public class Studente extends Persona{
        private int aliq;
        public Studente(int e, int a) throws Exception{
                super(e);
                if(a>=10 && a<=20)
                        this.aliq = a;
                else
                        throw new Exception("Valore dell'aliquota immesso errato!");
        }
        public int aliquota(){
                return aliq;
        }

}
```

Soluzione (6)

```
class Prova{
        public static void main(String [] arg){
                try{

                        Studente s = new Studente(23, 12);
                        Professore p1 = new Professore (40, 43);
                        Professore p2 = new Professore (41, 44);
                        Edificio e1 = new Edificio(1);
                        Edificio e2 = new Edificio(2);
                        Edificio e3 = new Edificio(3);
                }catch(Exception e){
                        e.printStackTrace();
                }
                Persona pp = (Persona) Utilita.MaggioreTraTre(s, p1, p2);
                Edificio ee = (Edificio) Utilita.MaggioreTraTre(e3, e1, e2);
                System.out.println(pp);
                System.out.println(ee);
        }
}
```

Esercizio 2

Si vuole definire un insieme di classi per la gestione di figure geometriche.

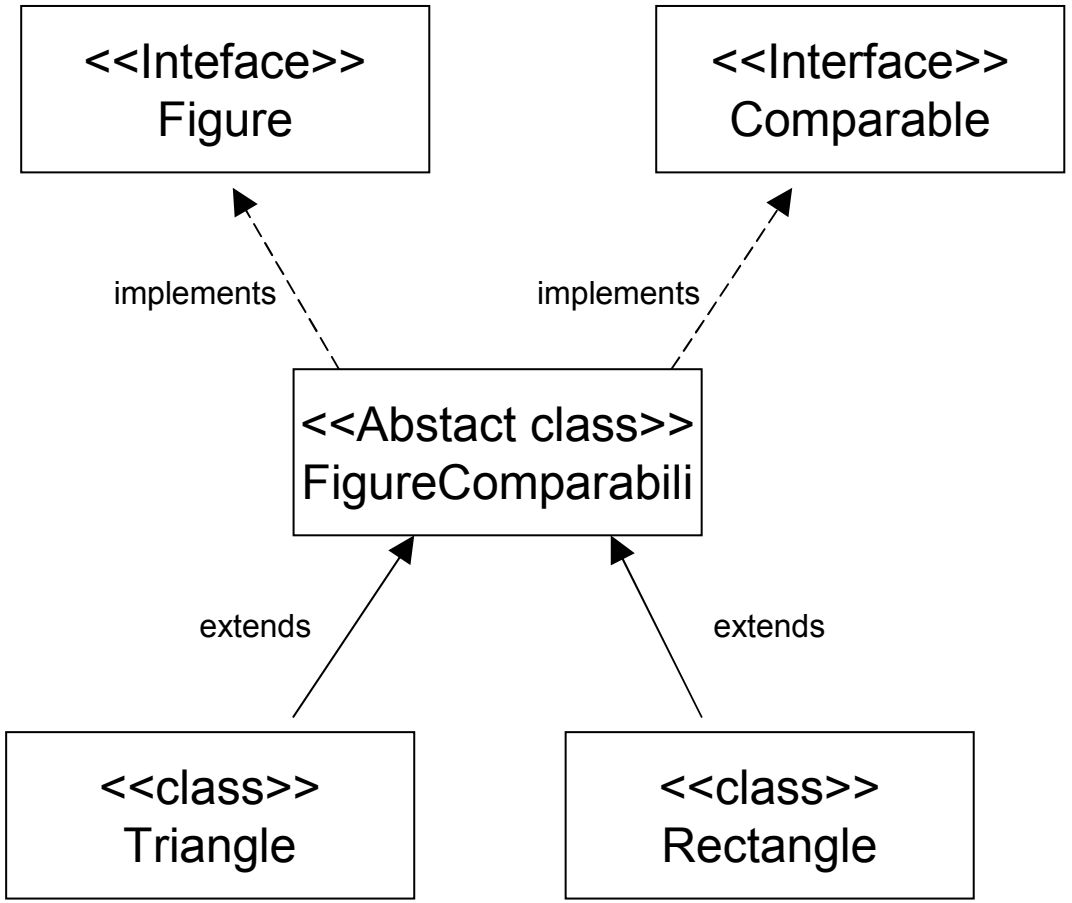
Per tutte le figure geometriche deve essere possibile determinare la tipologia di figura e calcolarne l'area, quindi tutte le figure geometriche sono catatterizzate da un'area ed una tipologia e deve essere possibile venire a conoscenza di tali informazioni.

Per alcune di esse si vuole poi fornire la possibilità di effettuare confronti, ovvero stabilire se sono uguali o differenti. Il confronto deve essere fatto in funzione dell'estensione dell'area. Una figura con area più estesa è più grande di una con area più ridotta. Se le due aree hanno la stessa estensione le figure sono considerate uguali. Il confronto non tiene conto della tipologia.

Modellare infine figure del tipo triangolo e rettangolo.

Fornire in fine un esempio di main per il testing delle classi realizzate

Soluzione


<<Inteface>>
Figure
<<Interface>>
Comparable
implements
implements
<<Abstact class>>
FigureComparabili
extends
extends
<<class>>
Triangle
<<class>>
Rectangle

Soluzione (2)

```
interface Figure {

    /* gli oggetti delle classi che realizzano questa interfaccia sono caratterizzati
            da un tipo e un area
    */

    double area();

    String tipo();

}
```

Soluzione (3)

```
abstract class FigureComparabili implements Figure, Comparable {

    protected double dim1;
    protected double dim2;

    FigureComparabili(double a, double b) {
            dim1 = a;
            dim2 = b;
    }

    // lascia astratti i metodi di Figure
    abstract public double area();
    abstract public String tipo();

    // definisce compare() di Comparable (uguale per tutte le figure)
    public int compareTo(Object o) {
            double a1 = this.area();
            double a2 = ((FigureComparabili)o).area(); // il cast serve
            return (int)(a1-a2) ;
    }

}
```

Soluzione (4)

```
class Rectangle extends FigureComparabili {

    Rectangle(double a, double b) {
            super(a, b);
    }

    // definisce area() di Figure
    public double area() {
            return dim1 * dim2;
    }

    // definisce tipo() di Figure
    public String tipo() {
            return "Rectangle ";
    }

}
```

```
class Triangle extends FigureComparabili {
        Triangle(double a, double b) {
                super(a, b);
        }
    // definisce area() di Figure
    public double area() {
            return dim1 * dim2 / 2;
    }

    // definisce tipo() di Figure
    public String tipo() {
            return "Triangle ";
    }
}
```

Soluzione (5)

```
class Interfacce2 {

    public static void main(String args[]) {

            FigureComparabili f1,f2;

            f1 = new Rectangle(9, 5);
            System.out.println(f1.tipo() + f1.area());
            f2= new Triangle(10, 8);
            System.out.println(f2.tipo() + f2.area());
            System.out.println("Confronto: " + f1.compareTo(f2));

    }
}
```

Esercizio 3

Scrivere una classe Esperimento che mantenga informazioni riguardo ai dati sperimentali misurati durante un esperimento. I dati vengono forniti sotto forma di valori interi positivi memorizzati su file (uno per ciascuna linea).
La classe Esperimento deve avere le seguenti funzionalità:

- calcolo e restituzione del numero dei valori
- calcolo e restituzione della somma dei valori
- calcolo e restituzione del massimo dei valori
- calcolo e restituzione del minimo dei valori
- verifica se un dato valore è presente nel file
- scrittura su nuovo file (il cui nome è passato come argomento) dei valori

Note:
si realizzino le funzionalità richeste implementando metodi statici.
In particolare, ciascuna funzionalità deve essere realizzata da due metodi:
(i) un metodo public che prende in input la stringa del nome del file, lo apre e chiama il metodo ricorsivo;
(ii) un metodo ricorsivo (ovviamente privato) che accede al file (mediante un riferimento BufferedReader) e implementa la funzionalità richiesta;

Esercizio 1 (2)

Struttura del file dati:

34
45
15
25
17
25
1

Main per verificare il comportamento della classe

```
import java.io.*;

public class Main {

public static void main (String[] args) throws IOException {
                                        int numValori = Esperimento.conta("dati.txt");
                                        System.out.println("Il numero dei valori e' "+numValori);

                                        int sommaValori = Esperimento.somma("dati.txt");

                                        System.out.println("La somma dei valori e' "+sommaValori);

                                        int maxValore = Esperimento.massimo("dati.txt");
                                        System.out.println("Il valore massimo e' "+maxValore);

                                        int minValore = Esperimento.minimo("dati.txt");
                                        System.out.println("Il valore minimo e' "+minValore);

                                        boolean isPresent = Esperimento.presente("dati.txt", -1);
                                        System.out.print("Il valore -1 ");
                                        if (!isPresent) System.out.print("non ");
                                        System.out.println("e' presente");

        Esperimento.copia("dati.txt", "copy.txt");

                                        }

}
```

Soluzione

```
mport java.io.*;

public class Esperimento{

        //calcolo e restituzione del numero di valori
        static public int conta(String nomeFile) throws IOException {
                FileReader f = new FileReader(nomeFile);
                BufferedReader br = new BufferedReader(f);
                int num = contaAux(br);
                f.close();
                return num;

                }



        static private int contaAux(BufferedReader br) throws IOException {
                String s = br.readLine();
                if (s == null) return 0;
                else return 1 + contaAux(br);
        }
```

Soluzione (2)

```
//calcolo e restituzione della somma dei valori
static public int somma(String nomeFile) throws IOException {
        FileReader f = new FileReader(nomeFile);
        BufferedReader br = new BufferedReader(f);
        int sum = sommaAux(br);
        f.close();
        return sum;
}

static private int sommaAux(BufferedReader b) throws IOException {
        String s = b.readLine();
        if (s == null) return 0;
        else return Integer.parseInt(s) + sommaAux(b);
}
```

Soluzione (3)

```
//calcolo e restituzione del massimo dei valori
static public int massimo(String nomeFile) throws IOException {
        FileReader f = new FileReader(nomeFile);
        BufferedReader br = new BufferedReader(f);
        int max = massimoAux(br);
        f.close();
        return max;
}

static private int massimoAux(BufferedReader b) throws IOException {
        String s = b.readLine();
        if (s == null) return -1; //sicuramente questo non e' il massimo
        else {
                int aux = Integer.parseInt(s);
                int max = massimoAux(b);
                if (aux > max) return aux;
                else return max;
        }
}
```

Soluzione (4)

```
//calcolo e restituzione del minimo dei valori
static public int minimo(String nomeFile) throws IOException {
        FileReader f = new FileReader(nomeFile);
        BufferedReader br = new BufferedReader(f);
        int min = minimoAux(br);
        f.close();
        return min;
}

static private int minimoAux(BufferedReader b) throws IOException {
        String s = b.readLine();
        if (s == null) return 2000; //ipotizzo che tutti i numeri siano piu' piccoli di 2000
        else {
                int aux = Integer.parseInt(s);
                int min = minimoAux(b);
                if (aux < min) return aux;
                else return min;
        }
}
```

Soluzione (5)

```
// verifica se un dato valore è presente nel file
static public boolean presente(String nomeFile, int v)throws IOException{
        FileReader f = new FileReader(nomeFile);
        BufferedReader br = new BufferedReader(f);
        boolean pres = presenteAux(br, v);
        f.close();
        return pres;
}

static private boolean presenteAux(BufferedReader b, int val)throws IOException{
        String s = b.readLine();
        if (s == null) return false;
        else return (val == Integer.parseInt(s)) || presenteAux(b,val);
}
```

Soluzione (6)

```
// scrittura su nuovo file (il cui nome è passato come argomento) dei valori
static public void copia(String fileInput, String fileOutput)throws IOException{
        FileReader fi = new FileReader(fileInput);
        BufferedReader br = new BufferedReader(fi);
        FileWriter fo = new FileWriter(fileOutput);
        PrintWriter out = new PrintWriter(fo);
        copiaAux(br, out);
        fi.close();
        fo.close();
}

static private void copiaAux(BufferedReader b, PrintWriter o) throws IOException{
        String s = b.readLine();
        if (s == null) return;
        else {
                o.println(s);
                copiaAux(b, o);
        }
}
```

Esercizio 4

Scrivere una classe SequenzaBinaria che rappresenti delle sequenze di caratteri '0' e '1' e implementi le operazioni indicate nel seguente scheletro della classe, in maniera ricorsiva e quindi senza far uso dei relativi metodi della classe String.

```
public class SequenzaBinaria {
        // costruisce l'oggetto SequenzaBinaria da una stringa
        public SequenzaBinaria(String x) { }

        // calcola la lunghezza della sequenza
        public int lunghezza() { }

        // concatena la sequenza binaria t
        public SequenzaBinaria concatena (SequenzaBinaria t) { }

        // restituisce la posizione del primo carattere c nella sequenza oppure -1 se esso non e' presente
        public int indiceDi (char c) { }

        // verifica se la sequenza e' uguale a t
        public boolean uguale (SequenzaBinaria t) { }

        // verifica se p e' un prefisso della sequenza binaria
        public boolean prefisso (SequenzaBinaria p) { }

        // restituisce la lunghezza della sequenza piu' lunga di caratteri c consecutivi
        public int lungSequenzaMassima (char c) { }

}
```

Soluzione

```
Public class SequenzaBinaria {

    private String str="";

    // costruisce l'oggetto SequenzaBinaria da una stringa
    public SequenzaBinaria(String x) {
       str = x;
    }


    // metodi ausiliari
    private char primoCar() {
       return this.str.charAt(0);
    }

    private SequenzaBinaria resto() {
        if (this.str.equals(""))
            throw new RuntimeException("Sequenza binaria nulla");
        else
            return new SequenzaBinaria(this.str.substring(1));
    }

    private boolean vuota() {
        return this.str.equals("");
    }
```

Soluzione (2)

```
// metodi pubblici
    public String toString() {
      return str;
    }

    // calcola la lunghezza della sequenza
    public int lunghezza() {
        if (this.vuota())
         return 0;
      else
         return 1 + this.resto().lunghezza();
    }

    // concatena la sequenza binaria t
    public SequenzaBinaria concatena (SequenzaBinaria t) {
        if (this.vuota())
         return t;
      else {
         SequenzaBinaria aux = this.resto().concatena(t);
         return new SequenzaBinaria(this.primoCar() + aux.str);
      }
    }
```

Soluzione (3)

```
// restituisce la posizione del primo carattere c nella sequenza
// oppure -1 se esso non e' presente
    public int indiceDi (char c) {
        if (this.vuota())
            return -1;
        else if (this.primoCar()==c)
            return 0;
        else {
            int r = this.resto().indiceDi(c);
            if (r==-1)
                return r;
            else
                return r+1;
        }
    }

// verifica se la sequenza e' uguale a t
    public boolean uguale (SequenzaBinaria t) {
        if (this.vuota() && t.vuota())
            return true;
        else
            if (this.vuota() || t.vuota())
                return false;
            else
                return this.primoCar()==t.primoCar() &&
                       this.resto().uguale(t.resto());
     }
```

Soluzione (4)

```
// verifica se p e' un prefisso della sequenza binaria
    public boolean prefisso (SequenzaBinaria p) {
        if (p.vuota())
            return true;
        else {
            return this.primoCar()==p.primoCar() &&
                    this.resto().prefisso(p.resto());
        }
    }

    // restituisce la lunghezza della sequenza piu' lunga di
    // caratteri c consecutivi
    public int lungSequenzaMassima (char c) {
        if (this.vuota())
            return 0;
        else
            return lungSequenzaMassima(c,0,0);
    }
```

Soluzione (5)

```
// metodo ausiliario che si porta dietro la lunghezza della sequenza
    // corrente di caratteri c consecutivi e del massimo corrente
    private int lungSequenzaMassima (char c, int lungCorr, int max){
        if (this.vuota()){
            if (lungCorr>max)
                max=lungCorr;
            return max;
        }
        else{
            if (this.primoCar()==c){
                lungCorr++;
                return this.resto().lungSequenzaMassima(c,lungCorr,max);
            }
            else{
                if (lungCorr>max)
                    max=lungCorr;
                return this.resto().lungSequenzaMassima(c,0,max);
            }
        }
    }
```

Soluzione (6)

```
public class ProvaSequenzaBinaria {

    public static void main(String[] args) {
                SequenzaBinaria s1 = new SequenzaBinaria("0000110111");
                SequenzaBinaria s2 = new SequenzaBinaria("110100001");
                System.out.println(s1+" e' lunga "+s1.lunghezza());
                System.out.println(s2+" e' lunga "+s2.lunghezza());
                SequenzaBinaria s3 = s1.concatena(s2);
                System.out.println("Concatenazione: "+s3);
                System.out.println("Carattere 1: "+s3.indiceDi('1'));
                System.out.println("Carattere 0: "+s3.indiceDi('0'));
                System.out.println("Uguali "+s1+" e "+s2+" ? "+s1.uguale(s2));
                String p1="0001";
                String p2="11";
                System.out.println(p1+" e' prefisso di "+s1+" ? “ + s1.prefisso(new SequenzaBinaria(p1)));
                System.out.println(p2+" e' prefisso di "+s2+" ? “ + s2.prefisso(new SequenzaBinaria(p2)));
                System.out.println("Lunghezza sequenza massima di '1' in "+s1+": "+s1.lungSequenzaMassima('1'));
                System.out.println("Lunghezza sequenza massima di '0' in "+s2+": "+s2.lungSequenzaMassima('0'));
                System.out.println("Lunghezza sequenza massima di '1' in "+s3+": "+s3.lungSequenzaMassima('1'));
    }
}
```