RIMBALZO OGGETTI

```
/**
 * Aggiungi qui una descrizione della classe ProvaJava
 * prova figure dentro l'immagine del desktop
 * @author (Paolo P.)
 * @version (1.1.2016)
 */
import java.awt.*;
import java.awt.Robot;
import javax.swing.*;
import java.util.Vector;

class FiguraNelloSpazio{
        public int nPunti;
        public double massa;
        public double raggio;
        public double velocitaBaricentroX;
        public double velocitaBaricentroY;
        public double velocitaBaricentroZ;
        public double velocitaBaricentroModulo;
        public double[][] punti=new double[102][3];
        public Color[] colorePunti=new Color[102];
        public double momentoRotazioneSolidoX;
        public double momentoRotazioneSolidoY;
        public double momentoRotazioneSolidoZ;
        public double momentoRotazioneModulo;
        public static double CalcolaDistanza(double x1,double y1,double z1,double x2,double y2,double z2){
                return Math.sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)+(z1-z2)*(z1-z2));
        }
        public void ScalaPunti(double fattoreDiScala){
                for(int i=1;i<this.nPunti+1;i++){
                        this.punti[i][0]=this.punti[i][0]*fattoreDiScala;
                        this.punti[i][1]=this.punti[i][1]*fattoreDiScala;
                        this.punti[i][2]=this.punti[i][2]*fattoreDiScala;
                }
                this.raggio=this.raggio*fattoreDiScala;
                this.massa=this.massa*fattoreDiScala;
        }
        public void CalcolaRaggio(){
                this.raggio=0;
                for(int i=1;i<this.nPunti+1;i++){
                        if(this.raggio<Math.sqrt((this.punti[0][0]-this.punti[i][0])*(this.punti[0][0]-
this.punti[i][0])+(this.punti[0][1]-this.punti[i][1])*(this.punti[0][1]-this.punti[i][1])+(this.punti[0][2]-
this.punti[i][2])*(this.punti[0][2]-this.punti[i][2]))){
                                this.raggio=Math.sqrt((this.punti[0][0]-this.punti[i][0])*(this.punti[0]
[0]-this.punti[i][0])+(this.punti[0][1]-this.punti[i][1])*(this.punti[0][1]-this.punti[i][1])+(this.punti[0]
[2]-this.punti[i][2])*(this.punti[0][2]-this.punti[i][2]));
                        }
                }
        }
        public void CalcolaBaricentro(){
                this.punti[0][0]=0;
                this.punti[0][1]=0;
                this.punti[0][2]=0;
                for(int i=1;i<this.nPunti+1;i++){
                        this.punti[0][0]=this.punti[0][0]+this.punti[i][0]/(this.nPunti);
                        this.punti[0][1]=this.punti[0][1]+this.punti[i][1]/(this.nPunti);
                        this.punti[0][2]=this.punti[0][2]+this.punti[i][2]/(this.nPunti);
                }
        }
        public void Sposta(double dX,double dY,double dZ){
                for(int i=0;i<this.nPunti+1;i++)
                {
                                this.punti[i][0]=this.punti[i][0]+dX;
                                this.punti[i][1]=this.punti[i][1]+dY;
                                this.punti[i][2]=this.punti[i][2]+dZ;
                }
        }
        public void Ruota(double mX,double mY,double mZ,double teta){
                double[][] a=new double[3][3];
                a[0][0]=mX*mX+(1-mX*mX)*Math.cos(teta);
                a[0][1]=(1-Math.cos(teta))*mX*mY-Math.sin(teta)*mZ;
                a[0][2]=(1-Math.cos(teta))*mX*mZ+Math.sin(teta)*mY;
                a[1][0]=(1-Math.cos(teta))*mX*mY+Math.sin(teta)*mZ;
                a[1][1]=mY*mY+(1-mY*mY)*Math.cos(teta);
                a[1][2]=(1-Math.cos(teta))*mY*mZ-Math.sin(teta)*mX;
                a[2][0]=(1-Math.cos(teta))*mX*mZ-Math.sin(teta)*mY;
                a[2][1]=(1-Math.cos(teta))*mY*mZ+Math.sin(teta)*mX;
                a[2][2]=mZ*mZ+(1-mZ*mZ)*Math.cos(teta);
                for(int i=1;i<this.nPunti+1;i++)
                {
                                this.punti[i][0]=this.punti[i][0]+a[0][0]*(this.punti[i][0]-
this.punti[0][0])+a[0][1]*(this.punti[i][1]-this.punti[0][1])+a[0][2]*(this.punti[i][2]-this.punti[0][2]);
                                this.punti[i][1]=this.punti[i][1]+a[1][0]*(this.punti[i][0]-
this.punti[0][0])+a[1][1]*(this.punti[i][1]-this.punti[0][1])+a[1][2]*(this.punti[i][2]-this.punti[0][2]);
                                this.punti[i][2]=this.punti[i][2]+a[2][0]*(this.punti[i][0]-
this.punti[0][0])+a[2][1]*(this.punti[i][1]-this.punti[0][1])+a[2][2]*(this.punti[i][2]-this.punti[0][2]);
                        }
                }
                public void AggiornaPosizione(){
                for(int i=0;i<this.nPunti+1;i++)
                {
                        this.punti[i][0]=this.punti[i][0]+this.velocitaBaricentroX*this.velocitaBaricentroModulo+
((this.punti[0][1]-this.punti[i][1])*this.momentoRotazioneSolidoZ-(this.punti[0][2]-this.punti[i]
[2])*this.momentoRotazioneSolidoY)*this.momentoRotazioneModulo;
                        this.punti[i][1]=this.punti[i][1]+this.velocitaBaricentroY*this.velocitaBaricentroModulo+
((this.punti[0][2]-this.punti[i][2])*this.momentoRotazioneSolidoX-(this.punti[0][0]-this.punti[i]
[0])*this.momentoRotazioneSolidoZ)*this.momentoRotazioneModulo;
                        this.punti[i][2]=this.punti[i][2]+this.velocitaBaricentroZ*this.velocitaBaricentroModulo+
((this.punti[0][0]-this.punti[i][0])*this.momentoRotazioneSolidoY-(this.punti[0][1]-this.punti[i]
[1])*this.momentoRotazioneSolidoX)*this.momentoRotazioneModulo;
                        this.velocitaBaricentroModulo=this.velocitaBaricentroModulo*0.9999;
                }
                }
                public void rimbalzoAiBordi(int maxx,int maxy,int maxz){
                        if(((this.punti[0][0]>maxx-this.raggio)&&(this.velocitaBaricentroX>0))||((this.punti[0]
[0]<this.raggio-300)&&(this.velocitaBaricentroX<0)))
                        {
                                this.velocitaBaricentroX=-this.velocitaBaricentroX;
                                this.momentoRotazioneSolidoX=-this.momentoRotazioneSolidoX;
                                this.momentoRotazioneSolidoY=-this.momentoRotazioneSolidoY;
                                this.momentoRotazioneSolidoZ=-this.momentoRotazioneSolidoZ;
                        }
                        if(((this.punti[0][1]>maxy-this.raggio)&&(velocitaBaricentroY>0))||((this.punti[0][1]<this.raggio-
300)&&(this.velocitaBaricentroY<0)))
                        {
                                this.velocitaBaricentroY=-this.velocitaBaricentroY;
                                this.momentoRotazioneSolidoX=-this.momentoRotazioneSolidoX;
                                this.momentoRotazioneSolidoY=-this.momentoRotazioneSolidoY;
                                this.momentoRotazioneSolidoZ=-this.momentoRotazioneSolidoZ;
                        }
                        if(((this.punti[0][2]>maxz-this.raggio)&&(velocitaBaricentroZ>0))||((this.punti[0][2]<this.raggio-
300)&&(this.velocitaBaricentroZ<0)))
                        {
                                this.velocitaBaricentroZ=-this.velocitaBaricentroZ;
                                this.momentoRotazioneSolidoX=-this.momentoRotazioneSolidoX;
                                this.momentoRotazioneSolidoY=-this.momentoRotazioneSolidoY;
                                this.momentoRotazioneSolidoZ=-this.momentoRotazioneSolidoZ;
                        }
                }
}
                class FiguraPianaNelloSpazio extends FiguraNelloSpazio{
                        public Color coloreSpigoli;
                        public int nPoligoni;
                public int nPuntiPoligono;
                public int[] ordinaPoligoni =new int[101];
                public double[] ordinaDistanze =new double[101];
                        public Color[] colorePoligono=new Color[101];
                        public int[][] poligono=new int[101][4];
                        public double[][] baricentroPoligono = new double[101][3];
                        public void CalcolaBaricentroPoligono(){
                                for(int i=0;i<this.nPoligoni;i++){
                                        this.baricentroPoligono[i][0]=0;
                                        this.baricentroPoligono[i][1]=0;
                                        this.baricentroPoligono[i][2]=0;
                                        for(int k=0;k<this.nPuntiPoligono;k++){
                                                this.baricentroPoligono[i][0]=this.baricentroPoligono[i]
[0]+this.punti[this.poligono[i][k]][0]/this.nPuntiPoligono;
                                                this.baricentroPoligono[i][1]=this.baricentroPoligono[i]
[1]+this.punti[this.poligono[i][k]][1]/this.nPuntiPoligono;
                                                this.baricentroPoligono[i][2]=this.baricentroPoligono[i]
[2]+this.punti[this.poligono[i][k]][2]/this.nPuntiPoligono;
                                        }
                                }
                                // Ordino i le facce del solido a seconda della distanza dei baricentri dei poligoni
dall'osservatore
                                public void OrdinaPoligono(double puntoOsservazioneX,double puntoOsservazioneY,double
puntoOsservazioneZ){
                                        int x;
                                        double y;
```

```
                                for(int i=0;i<this.nPoligoni;i++) {
                                        this.ordinaDistanze[i]=this.baricentroPoligono[i][2];
//CalcolaDistanza(this.baricentroPoligono[i][0]*(400+puntoOsservazioneZ)/(400+this.baricentroPoligono[i][2]),
this.baricentroPoligono[i][1]*(400+puntoOsservazioneZ)/(400+this.baricentroPoligono[i]
[2]),400+this.baricentroPoligono[i][2], puntoOsservazioneX, puntoOsservazioneY,400+puntoOsservazioneZ);
                                        this.ordinaPoligoni[i]=i;
                                }
                                for(int i=0;i<this.nPoligoni-1;i++) for(int j=i+1;j<this.nPoligoni;j++)
if(this.ordinaDistanze[i]<this.ordinaDistanze[j]){
                                        y=this.ordinaDistanze[i];
                                        this.ordinaDistanze[i]=this.ordinaDistanze[j];
                                        this.ordinaDistanze[j]=y;
                                        x=this.ordinaPoligoni[i];
                                        this.ordinaPoligoni[i]=this.ordinaPoligoni[j];
                                        this.ordinaPoligoni[j]=x;
                                }
                        }
                }
                class FiguraSolidaNelloSpazio extends FiguraPianaNelloSpazio{
                        public int tipoSolido;
                        public int nTetraedri;
                        public int[][] tetraedro =new int[5][4];
                }
                class InsiemeDiOggetti{
                        public static int[] ordinaOggetti=new int[51];
                        public static double[] distanzeOggetti =new double[51];
                        public static boolean[][][] verificaCollisioni=new boolean[51][51][2];          //
verificaCollisioni[i][j][0]: Oggetti in collisione (true/false) /verificaCollisioni[i][j][1]: collisione già
servita (true/false)
                        public static Vector<FiguraSolidaNelloSpazio> insieme1=new
Vector<FiguraSolidaNelloSpazio>(0);
                        public static void VerificaCollisioni(){
                                for(int i=0;i<InsiemeDiOggetti.insieme1.size()-1;i++)for(int
j=i+1;j<InsiemeDiOggetti.insieme1.size();j++)
                                {
                                        FiguraSolidaNelloSpazio solido1=insieme1.get(i);
                                        FiguraSolidaNelloSpazio solido2=insieme1.get(j);

        if(solido1.raggio+solido2.raggio<FiguraPianaNelloSpazio.CalcolaDistanza(solido1.punti[0]
[0],solido1.punti[0][1],solido1.punti[0][2],solido2.punti[0][0],solido2.punti[0][1],solido2.punti[0][2]))
                                        {
                                                verificaCollisioni[i][j][0]=false;
                                                verificaCollisioni[i][j][1]=false;
                                                verificaCollisioni[j][i][0]=false;
                                                verificaCollisioni[j][i][1]=false;
                                        }
                                        else
                                        {
                                                verificaCollisioni[i][j][0]=true;
                                                verificaCollisioni[j][i][0]=true;
                                        }
                                }
                        }
                        public static void RimbalzoTraDueSolidi(int i,int j){
                                        FiguraSolidaNelloSpazio solido1=insieme1.get(i);
                                        FiguraSolidaNelloSpazio solido2=insieme1.get(j);
                                /* Calcolo distanza tra i centri */
                                double distanzac1c2=FiguraPianaNelloSpazio.CalcolaDistanza(solido1.punti[0]
[0],solido1.punti[0][1],solido1.punti[0][2],solido2.punti[0][0],solido2.punti[0][1],solido2.punti[0][2]);
                                // Calcolo masse oggetti
                                double m1=solido1.massa;
                                double m2=solido2.massa;
                                double tx1=solido1.velocitaBaricentroX*solido1.velocitaBaricentroModulo;
                                double ty1=solido1.velocitaBaricentroY*solido1.velocitaBaricentroModulo;
                                double tz1=solido1.velocitaBaricentroZ*solido1.velocitaBaricentroModulo;
                                double tx2=solido2.velocitaBaricentroX*solido2.velocitaBaricentroModulo;
                                double ty2=solido2.velocitaBaricentroY*solido2.velocitaBaricentroModulo;
                                double tz2=solido2.velocitaBaricentroZ*solido2.velocitaBaricentroModulo;
                                /* Calcolo quantità di moto oggetti */
                                double vcx=(m1*tx1+m2*tx2)/(m1+m2);
                                double vcy=(m1*ty1+m2*ty2)/(m1+m2);
                                double vcz=(m1*tz1+m2*tz2)/(m1+m2);
                                /* Trasformazione in coordinate di centro massa */
                                double v01modulo=Math.sqrt((tx1-vcx)*(tx1-vcx)+(ty1-vcy)*(ty1-vcy)+(tz1-vcz)*(tz1-
vcz));
                                double v02modulo=Math.sqrt((tx2-vcx)*(tx2-vcx)+(ty2-vcy)*(ty2-vcy)+(tz2-vcz)*(tz2-
vcz));
                                double cosalfa1=((tx1-vcx)*(solido2.punti[0][0]-solido1.punti[0][0])+
(solido1.velocitaBaricentroY-vcy)*(solido2.punti[0][1]-solido1.punti[0][1])+(tz1-vcz)*(solido2.punti[0][2]-
solido1.punti[0][2]))/(distanzac1c2*v01modulo);
                                double cosalfa2=((tx2-vcx)*(solido1.punti[0][0]-solido2.punti[0][0])+
(solido2.velocitaBaricentroY-vcy)*(solido1.punti[0][1]-solido2.punti[0][1])+(tz2-vcz)*(solido1.punti[0][2]-
solido2.punti[0][2]))/(distanzac1c2*v02modulo);
                                /* Calcolo nuove velocità dopo il rimbalzo */
                                solido1.velocitaBaricentroModulo=Math.sqrt(tx1*tx1+ty1*ty1+tz1*tz1+0.01);
                                solido2.velocitaBaricentroModulo=Math.sqrt(tx2*tx2+ty2*ty2+tz2*tz2+0.01);
                                solido1.velocitaBaricentroX=(tx1-2*cosalfa1*(solido2.punti[0][0]-solido1.punti[0]
[0])*v01modulo/distanzac1c2)/solido1.velocitaBaricentroModulo;
                                solido1.velocitaBaricentroY=(ty1-2*cosalfa1*(solido2.punti[0][1]-solido1.punti[0]
[1])*v01modulo/distanzac1c2)/solido1.velocitaBaricentroModulo;
                                solido1.velocitaBaricentroZ=(tz1-2*cosalfa1*(solido2.punti[0][2]-solido1.punti[0]
[2])*v01modulo/distanzac1c2)/solido1.velocitaBaricentroModulo;
                                solido2.velocitaBaricentroX=(tx2-2*cosalfa2*(solido1.punti[0][0]-solido2.punti[0]
[0])*v02modulo/distanzac1c2)/solido2.velocitaBaricentroModulo;
                                solido2.velocitaBaricentroY=(ty2-2*cosalfa2*(solido1.punti[0][1]-solido2.punti[0]
[1])*v02modulo/distanzac1c2)/solido2.velocitaBaricentroModulo;
                                solido2.velocitaBaricentroZ=(tz2-2*cosalfa2*(solido1.punti[0][2]-solido2.punti[0]
[2])*v02modulo/distanzac1c2)/solido2.velocitaBaricentroModulo;
                                // Riduci velocità dopo il rimbalzo
                                solido1.velocitaBaricentroModulo=1.5*Math.sqrt(solido1.velocitaBaricentroModulo);
//Math.sqrt(Math.sqrt(solido1.velocitaBaricentroModulo*solido1.velocitaBaricentroModulo));
                                solido2.velocitaBaricentroModulo=1.5*Math.sqrt(solido2.velocitaBaricentroModulo);
//Math.sqrt(Math.sqrt(solido2.velocitaBaricentroModulo*solido2.velocitaBaricentroModulo));
                                // Cambia direzione rotazione
                                solido1.momentoRotazioneSolidoX=-solido1.momentoRotazioneSolidoX;
                                solido1.momentoRotazioneSolidoY=-solido1.momentoRotazioneSolidoY;
                                solido1.momentoRotazioneSolidoZ=-solido1.momentoRotazioneSolidoZ;
                                solido2.momentoRotazioneSolidoX=-solido2.momentoRotazioneSolidoX;
                                solido2.momentoRotazioneSolidoY=-solido2.momentoRotazioneSolidoY;
                                solido2.momentoRotazioneSolidoZ=-solido2.momentoRotazioneSolidoZ;
                                solido1.momentoRotazioneModulo=solido1.momentoRotazioneModulo*0.99;
                                solido2.momentoRotazioneModulo=solido2.momentoRotazioneModulo*0.99;
                                // Salva le impostazioni
                                InsiemeDiOggetti.insieme1.set(i,solido1);
                                InsiemeDiOggetti.insieme1.set(j,solido2);
                        }
                        public static void AggiungiSolido(int indice1,int tipoSolido1){
                                final double p1=30.0/Math.sqrt(3);
                                double x1,x2,x3,r1;
                                int coloreR=(int)(Math.random()*220);
        int coloreG=(int)(Math.random()*220);
        int coloreB=(int)(Math.random()*220);
                                FiguraSolidaNelloSpazio solido1=new FiguraSolidaNelloSpazio();
                                solido1.tipoSolido=tipoSolido1;
                                solido1.massa=(Math.random()*100+30.01)/60.001;
                                solido1.velocitaBaricentroX=0.5-Math.random();
        solido1.velocitaBaricentroY=0.5-Math.random();
        solido1.velocitaBaricentroZ=0.5-Math.random();


solido1.velocitaBaricentroX=solido1.velocitaBaricentroX/Math.sqrt(solido1.velocitaBaricentroX*solido1.velocitaB
aricentroX+solido1.velocitaBaricentroY*solido1.velocitaBaricentroY+solido1.velocitaBaricentroZ*solido1.velocita
BaricentroZ);


solido1.velocitaBaricentroY=solido1.velocitaBaricentroY/Math.sqrt(solido1.velocitaBaricentroX*solido1.velocitaB
aricentroX+solido1.velocitaBaricentroY*solido1.velocitaBaricentroY+solido1.velocitaBaricentroZ*solido1.velocita
BaricentroZ);


solido1.velocitaBaricentroZ=solido1.velocitaBaricentroZ/Math.sqrt(solido1.velocitaBaricentroX*solido1.velocitaB
aricentroX+solido1.velocitaBaricentroY*solido1.velocitaBaricentroY+solido1.velocitaBaricentroZ*solido1.velocita
BaricentroZ);
                        solido1.velocitaBaricentroModulo=(Math.random()*100+30.01)/6.001;
                        solido1.coloreSpigoli=new Color((int)(Math.random()*256),(int)(Math.random()*256),(int)
(Math.random()*256));
                        solido1.momentoRotazioneSolidoX=0.5-Math.random();
                        solido1.momentoRotazioneSolidoX=0.5-Math.random();
                        solido1.momentoRotazioneSolidoX=0.5-Math.random();


solido1.momentoRotazioneSolidoX=solido1.momentoRotazioneSolidoX/Math.sqrt(solido1.momentoRotazioneSolidoX*solid
o1.momentoRotazioneSolidoX+solido1.momentoRotazioneSolidoY*solido1.momentoRotazioneSolidoY+solido1.momentoRotaz
ioneSolidoZ*solido1.momentoRotazioneSolidoZ);

solido1.momentoRotazioneSolidoY=solido1.momentoRotazioneSolidoY/Math.sqrt(solido1.momentoRotazioneSolidoX*solid
o1.momentoRotazioneSolidoX+solido1.momentoRotazioneSolidoY*solido1.momentoRotazioneSolidoY+solido1.momentoRotaz
ioneSolidoZ*solido1.momentoRotazioneSolidoZ);

solido1.momentoRotazioneSolidoZ=solido1.momentoRotazioneSolidoZ/Math.sqrt(solido1.momentoRotazioneSolidoX*solid
```

```
o1.momentoRotazioneSolidoX+solido1.momentoRotazioneSolidoY*solido1.momentoRotazioneSolidoY+solido1.momentoRotazioneSolidoZ*solido1.momentoRotazioneSolidoZ);
            solido1.momentoRotazioneModulo=(Math.random()*100+30.01)/1300.001;
          switch (tipoSolido1) {
                    case 0:                                                 // Inizializza sfere
                            solido1.nPunti=0;
                            solido1.nPuntiPoligono=0;
                            solido1.nPoligoni=0;
                            solido1.punti[0][0]=0;
                            solido1.punti[0][1]=0;
                            solido1.punti[0][2]=0;
                            solido1.raggio=30;
                            solido1.ScalaPunti(Math.random()+0.2);
                    break;
                    case 1:{                                        // Inizializza tetraedri
                            solido1.nPunti=4;
                            solido1.nPoligoni=4;
                        solido1.nPuntiPoligono=3;
                        solido1.punti=new double[][] {{0,0,0},{30,0,0},{0,0,30},{0,30,0},{-15,-15,-15}};
                        solido1.CalcolaRaggio();
                        solido1.ScalaPunti(Math.random()*1.2+0.4);
                        solido1.CalcolaBaricentro();
                        solido1.poligono = new int[][] {{1,2,3},{1,4,3},{1,2,4},{2,4,3}}; // Inizializza le facce del tetraedro
                        solido1.CalcolaBaricentroPoligono();
                        // Definizione dei colori delle facce e degli spigoli
                            for(int i=0;i<4;i++){
                                    solido1.colorePoligono[i]=new Color(coloreR+(int)(Math.random()*36),coloreG+(int)(Math.random()*36),coloreB+(int)(Math.random()*36));
                            }
                    }
                            break;
                    case 2:{                                        // Inizializza  cubi
                            solido1.nPunti=8;
                            solido1.nPoligoni=6;
                            solido1.nPuntiPoligono=4;
                            solido1.punti = new double[][] {{0,0,0},{0,0,0},{30,0,0},{30,30,0},{0,30,0},{0,0,30},{30,0,30},{30,30,30},{0,30,30}};
                            solido1.CalcolaRaggio();
                            solido1.ScalaPunti(Math.random()*1.7+0.4);
                            solido1.CalcolaBaricentro();
                            solido1.poligono=new int[][] {{1,2,3,4},{5,6,7,8},{1,2,6,5},{2,3,7,6},{3,4,8,7},{1,4,8,5}}; // Inizializza le facce del cubo
                            solido1.CalcolaBaricentroPoligono();
                            // Definizione dei colori delle facce e degli spigoli
                                for(int j=0;j<6;j++){
                                    solido1.colorePoligono[j]=new Color(coloreR+(int)(Math.random()*36),coloreG+(int)(Math.random()*36),coloreB+(int)(Math.random()*36));
                                                        }
                                solido1.coloreSpigoli=new Color((int)(Math.random()*256),(int)(Math.random()*256),(int)(Math.random()*256));
                    }
                            break;
                    case 3:                         // Inizializza figure di prova
                    {
                            solido1.nPunti=14;
                            solido1.nPoligoni=24;
                            solido1.nPuntiPoligono=3;
                            solido1.punti = new double[][] {{0,0,0},{-p1,-p1,-p1},{-p1,-p1,p1},{-p1,p1,-p1},{-p1,p1,p1},{p1,-p1,-p1},{p1,-p1,p1},{p1,p1,-p1},{p1,p1,p1},{30,0,0},{0,30,0},{0,0,30},{-30,0,0},{0,-30,0},{0,0,-30}};
                            solido1.CalcolaRaggio();
                            solido1.ScalaPunti(Math.random()*2.2+0.6);
                            solido1.CalcolaBaricentro();
                            solido1.poligono=new int[][] {{1,12,13},{1,14,13},{1,12,14},{2,12,13},{2,12,11},{2,11,13},{3,10,12},{3,10,14},{3,12,14},{4,12,11},{4,12,10},{4,10,11},{5,9,13},{5,9,14},{5,13,14},{6,9,13},{6,9,11},{6,11,13},{7,9,10},{7,10,14},{7,9,14},{8,9,10},{8,9,11},{8,10,11}};// Inizializza poligoni
                            solido1.CalcolaBaricentroPoligono();
                            // Definizione dei colori delle facce e degli spigoli
                                for(int j=0;j<solido1.nPoligoni;j++){
                                    solido1.colorePoligono[j]=new Color(coloreR+(int)(Math.random()*36),coloreG+(int)(Math.random()*36),coloreB+(int)(Math.random()*36));
                                                        }
                    }
                                                        break;
                    case 4:                         // Inizializza figure di prova
                    {
                            coloreR=(int)(Math.random()*200);
                            coloreG=(int)(Math.random()*200);
                            coloreB=(int)(Math.random()*200);
                            solido1.nPunti=100;
                            solido1.nPoligoni=0;
                            solido1.nPuntiPoligono=0;
                            //solido1.punti = new double[][] {{0,0,0},{-p1,-p1,-p1},{-p1,-p1,p1},{-p1,p1,-p1},{-p1,p1,p1},{p1,-p1,-p1},{p1,-p1,p1},{p1,p1,-p1},{p1,p1,p1},{30,0,0},{0,30,0},{0,0,30},{-30,0,0},{0,-30,0},{0,0,-30}};
                            solido1.punti[0][0]=0;
                            solido1.punti[0][1]=0;
                            solido1.punti[0][2]=0;
                            for(int i=1;i<solido1.nPunti+1;i++){
                                    x1=0.5-Math.random();
                                    x2=0.5-Math.random();
                                    x3=0.5-Math.random();
                                    r1=Math.sqrt(x1*x1+x2*x2+x3*x3);
                                    solido1.punti[i][0]=x1*30/r1;
                                    solido1.punti[i][1]=x2*30/r1;
                                    solido1.punti[i][2]=x3*30/r1;
                                    // Definizione dei colori delle facce e degli spigoli
                                    solido1.colorePunti[i]=new Color(coloreR+(int)(Math.random()*56),coloreG+(int)(Math.random()*56),coloreB+(int)(Math.random()*56));
                            }
                            solido1.CalcolaRaggio();
                            //solido1.CalcolaBaricentro();
                            solido1.CalcolaBaricentroPoligono();
                            solido1.ScalaPunti(Math.random()*1.2+0.6);
                            //solido1.poligono=new int[][] {{0,1},{0,2},{0,3},{0,4},{0,5},{0,6},{0,7},{0,8},{0,9},{0,10},{0,11},{0,12},{0,13},{0,14},{0,15},{0,16},{0,17},{0,18},{0,19},{0,20},{0,21},{0,22},{0,23},{0,24},{0,25},{0,26},{0,27},{0,28},{0,29},{0,30}};// Inizializza poligoni
                    }
                                                        break;
                    default: break;
                }
            insieme1.add(indice1,solido1);
                    }
                    // Ordino i solidi a seconda della distanza dei baricentri dall'osservatore
                    public static void OrdinaOggetti(double distanza,double puntoOsservazioneX,double puntoOsservazioneY,double puntoOsservazioneZ){
                            int x;
                            double y;
                            FiguraSolidaNelloSpazio solidoZ;
                            for(int i=0;i<InsiemeDiOggetti.insieme1.size();i++) {
                                    solidoZ=insieme1.get(i);

                                distanzeOggetti[i]=FiguraNelloSpazio.CalcolaDistanza(solidoZ.punti[0][0]*distanza/(distanza+solidoZ.punti[0][2]),solidoZ.punti[0][1]*distanza/(distanza+solidoZ.punti[0][2]),distanza*solidoZ.punti[0][2]/(distanza+solidoZ.punti[0][2]), puntoOsservazioneX, puntoOsservazioneY,distanza+puntoOsservazioneZ); //solidoZ.punti[0][2];
                                    ordinaOggetti[i]=i;
                            }
                            for(int i=0;i<InsiemeDiOggetti.insieme1.size()-1;i++) for(int j=i+1;j<InsiemeDiOggetti.insieme1.size();j++) if(distanzeOggetti[i]<distanzeOggetti[j])
                            {
                                    y=distanzeOggetti[i];
                                    distanzeOggetti[i]=distanzeOggetti[j];
                                    distanzeOggetti[j]=y;
                                    x=ordinaOggetti[i];
                                    ordinaOggetti[i]=ordinaOggetti[j];
                                    ordinaOggetti[j]=x;
                            }
                    }
            }
/** Rimbalzo Lastre(Animation) via custom thread */
@SuppressWarnings("serial")
class ProvaJava extends JFrame {
    public FiguraSolidaNelloSpazio solidoX=new FiguraSolidaNelloSpazio();
    public FiguraSolidaNelloSpazio solidoY=new FiguraSolidaNelloSpazio();
          protected static Image img;
          public int maxx=1000;
          public int maxy=500;
          public int maxz=500;
          private final int distanza=400;
          final double puntoOsservazioneX=maxx/2;
          final double puntoOsservazioneY=maxy/2;
          final double puntoOsservazioneZ=50;
    /** Constructor to setup the GUI components */
    ProvaJava() {
    // Define named-constants
    final int LARGHEZZA = 1440;
    final int ALTEZZA = 900;
    final int INTERVALLO = 50; // milliseconds
```

```
    final Disegna disegna;  // the drawing canvas (extends JPanel)
    // Inizializza solidi
    for(int i=0;i<(int)(Math.random()*30)+20;i++) InsiemeDiOggetti.AggiungiSolido(i,(int)(Math.random()*5));
    disegna = new Disegna();
    disegna.setPreferredSize(new Dimension(LARGHEZZA,ALTEZZA));
    this.setContentPane(disegna);
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    this.pack();
    this.setTitle("Figure in movimento");
    this.setVisible(true);
    // Create a new thread to run update at regular interval
    Thread updateThread;
            updateThread = new Thread() {
                @Override
                //@SuppressWarnings("SleepWhileInLoop")
                public void run() {
                    while (true) {
                        update();   // update the (x, y) position
                        repaint();  // Refresh the JFrame. Called back paintComponent()
                        try {
                            // Delay and give other thread a chance to run
                            Thread.sleep(INTERVALLO);  // milliseconds
                        } catch (InterruptedException ignore) {}
                    }
                }
            };
        updateThread.start(); // called back run()
    }

    /** Update the (x, y) position of the moving object */
    void update() {
            /* ----------------------------------------------Rimbalzo tra
solidi----------------------------------------------------------------*/
            InsiemeDiOggetti.VerificaCollisioni();
            for(int i=0;i<InsiemeDiOggetti.insieme1.size();i++){
                    solidoX=InsiemeDiOggetti.insieme1.get(i);

//solidoX.Sposta(solidoX.velocitaBaricentroX,solidoX.velocitaBaricentroY,solidoX.velocitaBaricentroZ);

//solidoX.Ruota(solidoX.momentoRotazioneSolidoX,solidoX.momentoRotazioneSolidoX,solidoY.momentoRotazioneSolidoZ
,solidoX.momentoRotazioneModulo);
                    solidoX.AggiornaPosizione();
                    if(solidoX.tipoSolido>0)solidoX.CalcolaBaricentro();
                solidoX.rimbalzoAiBordi(maxx,maxy,maxz);
                        solidoX.CalcolaBaricentroPoligono();
                        solidoX.OrdinaPoligono(puntoOsservazioneX,puntoOsservazioneY,puntoOsservazioneZ);
                        InsiemeDiOggetti.insieme1.set(i,solidoX);
                        // Rimbalzo tra solidi
                        for(int j=i+1;j<InsiemeDiOggetti.insieme1.size();j++)
                                if(InsiemeDiOggetti.verificaCollisioni[i][j][0]&&!
InsiemeDiOggetti.verificaCollisioni[i][j][1]) {
                                        InsiemeDiOggetti.RimbalzoTraDueSolidi(i,j);
                                        InsiemeDiOggetti.verificaCollisioni[i][j][1]=true;
                                        InsiemeDiOggetti.verificaCollisioni[j][i][1]=true;
                                }
            }
            InsiemeDiOggetti.OrdinaOggetti(distanza,puntoOsservazioneX,puntoOsservazioneY,puntoOsservazioneZ);
    }
    /** DrawCanvas (inner class) is a JPanel used for custom drawing */
    class Disegna extends JPanel {
        private final int[] x=new int[5];
             private final int[] xpoints=new int[5];
             private final int[] ypoints=new int[5];
             private int npoints,z;
        @Override
        protected
        void paintComponent(Graphics g) {
           super.paintComponent(g);  // paint parent's background
           g.drawImage(img,0,0,1440,900,null);
           // Disegno i solidi in prospettiva lineare 3D
           for(int i=0;i<InsiemeDiOggetti.insieme1.size();i++){
             z=InsiemeDiOggetti.ordinaOggetti[i];
             solidoX=InsiemeDiOggetti.insieme1.get(z);
             switch (solidoX.tipoSolido) {
                        case 0:
                   // Disegno la sfera
                   g.setColor(solidoX.coloreSpigoli);
                   g.fillOval(maxx/2+(int) (solidoX.punti[0][0]*distanza/(distanza+solidoX.punti[0][2])),maxy/2+
(int) (solidoX.punti[0][1]*distanza/(distanza+solidoX.punti[0][2])),(int) (solidoX.raggio*distanza/
(distanza+solidoX.punti[0][2])),(int) (solidoX.raggio*distanza/(distanza+solidoX.punti[0][2])));
                   g.setColor(Color.BLACK);
                   g.drawOval(maxx/2+(int) (solidoX.punti[0][0]*distanza/(distanza+solidoX.punti[0][2])),maxy/2+
(int) (solidoX.punti[0][1]*distanza/(distanza+solidoX.punti[0][2])),(int) (solidoX.raggio*distanza/
(distanza+solidoX.punti[0][2])),(int) (solidoX.raggio*distanza/(distanza+solidoX.punti[0][2])));
                   break;
                        case 4:

                      for(int k=1;k<solidoX.nPunti+1;k++){
                               g.setColor(solidoX.colorePunti[k]);
                               g.drawLine((int) (maxx/2+solidoX.punti[0][0]*distanza/(distanza+solidoX.punti[0]
[2])),maxy/2+(int) (solidoX.punti[0][1]*distanza/(distanza+solidoX.punti[0][2])),maxx/2+(int) (solidoX.punti[k]
[0]*distanza/(distanza+solidoX.punti[k][2])),maxy/2+(int) (solidoX.punti[k][1]*distanza/
(distanza+solidoX.punti[k][2])));
                      }
                      break;
                       default:
                      for(int j=0;j<solidoX.nPoligoni;j++){
                                                    int p=solidoX.ordinaPoligoni[j];
                           npoints=solidoX.nPuntiPoligono+1;
                           for(int k=0;k<solidoX.nPuntiPoligono;k++){
                               x[k]=solidoX.poligono[p][k];
                               xpoints[k]=500+(int) (solidoX.punti[x[k]][0]*distanza/
(distanza+solidoX.punti[x[k]][2]));
                               ypoints[k]=500+(int) (solidoX.punti[x[k]][1]*distanza/
(distanza+solidoX.punti[x[k]][2]));
                           }
                           xpoints[solidoX.nPuntiPoligono]=xpoints[0];
                           ypoints[solidoX.nPuntiPoligono]=ypoints[0];
                           g.setColor(solidoX.colorePoligono[p]);
                           g.fillPolygon(xpoints, ypoints,npoints);
                           g.setColor(solidoX.coloreSpigoli);
                           for(int k=0;k<solidoX.nPuntiPoligono;k++){
                              g.drawLine(xpoints[k],ypoints[k],xpoints[k+1],ypoints[k+1]);
                           }
                                                }
                      break;
                   }
                }
             }
    }

    /** The entry main method */
    //    @SuppressWarnings("Convert2Lambda")
    public static void main(String[] args) throws AWTException {
        // capture the whole screen
        final Image salvaschermo;
        salvaschermo = new Robot().createScreenCapture(
                new Rectangle(Toolkit.getDefaultToolkit().getScreenSize()) );
      // Run GUI codes in Event-Dispatching thread for thread safety
      SwingUtilities.invokeLater(new Runnable() {
         @Override
         public void run() {
            new ProvaJava(); // Let the constructor do the job
            ProvaJava.img=salvaschermo;
         }
      });
    }
}
```