Implementazione dei numeri complessi

Questa realizzazione dà per scontata la conoscenza della teoria matematica dei numeri complessi.

Ci limitiamo a ricordare le principali proprietà

Dato un numero complesso z questo può essere rappresentato come somma della parte reale e della parte immaginaria oppure come modulo o argomento.

Nel primo caso abbiamo una rappresentazione cartesiana nel piano di Gauss nel secondo una rappresentazione polare.

Equazioni della rappresentazione cartesiana

Siano $z = a + i\,b$, allora

$re(z) = a$, $im(z)=b$;

$z^* = a - i\,b$;

$abs(z) = sqrt(z\,z^*) = sqrt(a^2+b^2)$;

$arg(z) = arctan(b/a)$, se $a\neq 0$, $\pi/2$, se $a=0$, $b>0$, $3/2\,\pi$ se $a=0$, $b<0$;

NB l'argomento è definito modulo 2π, per fissarne il valore lo rappresentiamo in $[0,2\pi)$.

$\alpha\, z = (\alpha\, a) + i\,(\alpha\, b)$;

$1/z = z^*/(z\,z^*) = (a - i\,b)/(a^2+b^2)$

Siano $z_1 = a_1 + i\,b_1$, $z_2 = a_2 + i\,b_2$, allora

$z_1 \pm z_2 = (a_1 \pm a_2) + i\,(b_1 \pm b_2)$;

$z_1\, z_2 = (a_1\, a_2 - b_1\, b_2) + i\,(a_1\, b_2 + a_2\, b_1)$;

$z_1 / z_2 = z_1\, z_2^* / (z_2\, z_2^*) = z_1\, [1/(z_2\, z_2^*)]\, z_2^*$.

NB $[1/(z_2\, z_2^*)]$ è un numero reale.

Equazioni della rappresentazione polare

Siano $z = \rho\, e^{i\theta}$, allora

$re(z) = \rho \cos(\theta),\ \ im(z) = \rho \sin(\theta)$;

$z^* = \rho\, e^{-i\theta}$;

$Abs(z) = \rho$, $Arg(z) = \theta$;

$\alpha\, z = (\alpha\, \rho)\, e^{i\theta}$;

$1/z = (1/\rho)\, e^{-i\theta}$

Siano $z_1 = \rho_1\, e^{i\theta_1}$, $z_2 = \rho\, e^{i\theta_2}$, allora

$z_1 \pm z_2 = (\rho_1 \cos(\theta_1) \pm \rho_2 \cos(\theta_2)) + i\, (\rho_1 \sin(\theta_1) \pm \rho_2 \sin(\theta_2))$;

$z_1\ z_2 = (\rho_1\, \rho_2)\, e^{i(\theta_1+\theta_2)}$;

$z_1 / z_2 = (\rho_1/\rho_2)\, e^{i(\theta_1-\theta_2)}$.

Prima implementazione

- una classe `Complex1` che implementa la rappresentazione cartesiana;
- una classe `Complex2` che implementa la rappresentazione polare.

Entrambe le classi implementano i metodi di istanza

`re()`, `im()`, `abs()`, `arg()`, `conjugate()`, `toString()`,

i metodi statici

`sum()`, `sub()`, `times()`, `div()`,

e il metodo privato `rd3()` che arrotonda un numero positivo alla terza cifra decimale.

Per comodità riportiamo parte delle specifiche del metodo `atan2` della classe `java.lang.Math`

```
public static double atan2(double y,double x)
```

Converts rectangular coordinates (x, y) to polar (r, theta). This method computes the phase theta by computing an arc tangent of y/x in the range of -π to π.

Parameters: y - the ordinate coordinate, x - the abscissa coordinate

Returns: the theta component of the point (r, theta) in polar coordinates that corresponds to the point (x, y) in Cartesian coordinates.

```
import java.text.*;
public class Complex1 {
    double reP, imP;

    public Complex1(double reP, double imP){
        this.reP = reP;
        this.imP = imP;
    }

    public static Complex1 sum (Complex1 x, Complex1 y) {
        return new Complex1(x.reP + y.reP, x.imP + y.imP);
    }

    public static Complex1 sub (Complex1 x, Complex1 y) {
        return new Complex1(x.reP - y.reP,  x.imP - y.imP);
    }

    public static Complex1 times (Complex1 x, Complex1 y) {
        double reP = x.reP*y.reP - x.imP*y.imP;
        double imP = x.imP*y.reP + x.reP*y.imP;
        return new Complex1(reP, imP);
    }
```

```
public static Complex1 times (double a, Complex1 x) {
    return new Complex1(a*x.reP, a*x.imP);
}
public static Complex1 div (Complex1 x, Complex1 y) {
    return times(x, times(1/y.abs()/y.abs(), y.conjugate()));
}
public Complex1 conjugate(){
    return new Complex1(reP, -imP);
}
public double re(){return reP;}
public double im(){return imP;}
public double abs(){return Math.sqrt(reP*reP+imP*imP);}
public double arg(){
 return (2*Math.PI + Math.atan2(imP, reP)) % (2*Math.PI);
}
```

```
    public String toString(){
     String op = "+";
     if (imP<0) op="-";
     return rd3(reP)+" "+op+" i "+rd3(Math.abs(imP));
    }

    private NumberFormat nf = new DecimalFormat("0.###");
    private String rd3(double x){return nf.format(x);}
}
```

```
import java.text.*;
public class Complex2 {

  double rho, phi;

  public Complex2(double rho, double phi){
    this.rho = rho;
    this.phi = (2*Math.PI + phi % (2*Math.PI))% (2*Math.PI);
  }

  public static Complex2 sum (Complex2 x, Complex2 y) {
    double rez =
        x.rho * Math.cos(x.phi) + y.rho * Math.cos(y.phi);
    double imz =
        x.rho * Math.sin(x.phi) + y.rho * Math.sin(y.phi);
    return new Complex2(
        Math.sqrt(rez*rez+imz*imz),Math.atan2(imz, rez));
  }
```

```
 public static Complex2 sub (Complex2 x, Complex2 y) {
   double rez =
      x.rho * Math.cos(x.phi) - y.rho * Math.cos(y.phi);
   double imz =
      x.rho * Math.sin(x.phi) - y.rho * Math.sin(y.phi);
   return new Complex2(
      Math.sqrt(rez*rez+imz*imz),Math.atan2(imz, rez));
 }
public static Complex2 times (Complex2 x, Complex2 y) {
  return new Complex2(x.rho*y.rho, (x.phi+y.phi));
}

public static Complex2 times (double a, Complex2 x) {
  return new Complex2(a*x.rho, x.phi);
}
public static Complex2 div (Complex2 x, Complex2 y) {
  return new Complex2(x.rho/y.rho, (x.phi-y.phi));
}
```

```
public Complex2 conjugate(){return new Complex2(rho, -phi);}
 public double re(){return rho * Math.cos(phi);}
 public double im(){return rho * Math.sin(phi);}
 public double abs(){return rho;}
 public double arg(){return phi;}

 public String toString(){
   return rd3(rho)+" * exp(i " +rd3(phi)+")";
 }

 private NumberFormat nf = new DecimalFormat("0.###");
 private String rd3(double x){return nf.format(x);}
}
```

Seconda implementazione

- una classe astratta `AbstrComplex` che dichiara i metodi astratti

  `re()`, `im()`, `abs()`, `arg()`, `conjugate()`

  e implementa

  `toString()` e `rd3()`.

- una classe `CartesianComplex` che implementa la rappresentazione cartesiana;
- una classe `TrigComplex` che implementa la rappresentazione polare.

```
public abstract class AbstrComplex {

    public abstract AbstrComplex conjugate();
    public abstract double re();
    public abstract double im();
    public abstract double abs();
    public abstract double arg();

    public String toString(){
     String op = "+";
     if (this.im()<0) op="-";
     return rd3(this.re())+" "+op+" i "+rd3(Math.abs(this.im()));
    }

    private NumberFormat nf = new DecimalFormat("0.###");
    private String rd3(double x){return nf.format(x);}
}
```

```
public class CartesianComplex extends AbstrComplex{
  private double reP, imP;

  public CartesianComplex(double reP, double imP){
    this.reP = reP;
    this.imP = imP;
  }

  public static AbstrComplex sum(AbstrComplex x, AbstrComplex y){
    return new CartesianComplex(
        x.re() + y.re(), x.im() + y.im());
  }

  public static AbstrComplex sub (AbstrComplex x, AbstrComplex y){
    return new CartesianComplex(
        x.re() - y.re(),  x.im() - y.im());
  }

  public static AbstrComplex div (AbstrComplex x, AbstrComplex y){
    return times(x, times(1/y.abs()/y.abs(), y.conjugate()));
  }
```

```
  public static AbstrComplex times(
          AbstrComplex x, AbstrComplex y){
     double reZ = x.re()*y.re() - x.im()*y.im();
     double imZ = x.im()*y.re() + x.re()*y.im();
     return new CartesianComplex(reZ, imZ);
  }

  public static AbstrComplex times (double a, AbstrComplex x) {
    return new CartesianComplex(a*x.re(), a*x.im());
  }
  public AbstrComplex conjugate(){
    return new CartesianComplex(reP, -imP);
  }

  public double re(){return reP;}
  public double im(){return imP;}
  public double abs(){return Math.sqrt(reP*reP+imP*imP);}
  public double arg(){return Math.atan2(imP,reP);}
}
```

```
public class TrigComplex extends AbstrComplex {
  private double rho, phi;

  public TrigComplex(double rho, double phi){
    this.rho = rho;
    this.phi = (2*Math.PI + phi % (2*Math.PI))% (2*Math.PI);
  }

  public static AbstrComplex sum(AbstrComplex x, AbstrComplex y){
   double rez = x.abs()*Math.cos(x.arg())+y.abs()*Math.cos(y.arg());
   double imz = x.abs()*Math.sin(x.arg())+y.abs()*Math.sin(y.arg());
   return new TrigComplex(
        Math.sqrt(rez*rez+imz*imz),Math.atan2(imz,rez));
  }

  public static AbstrComplex sub(AbstrComplex x, AbstrComplex y){
   double rez =
        x.abs()*Math.cos(x.arg())-y.abs()*Math.cos(y.arg());
   double imz =
        x.abs()*Math.sin(x.arg())-y.abs()*Math.sin(y.arg());
   return new TrigComplex(
        Math.sqrt(rez*rez+imz*imz),Math.atan2(imz, rez));
  }
```

```
 public static AbstrComplex times(
         AbstrComplex x, AbstrComplex y){
   return new TrigComplex(x.abs()*y.abs(), x.arg()+y.arg());
 }
 public static AbstrComplex times (double a, AbstrComplex x) {
   return new TrigComplex(a*x.abs(), x.arg());
 }
 public static AbstrComplex div (
         AbstrComplex x, AbstrComplex y) {
   return new TrigComplex(x.abs()/y.abs(), x.arg()-y.arg());
 }
 public AbstrComplex conjugate(){
   return new TrigComplex(rho, -phi);
 }
 public double re(){return rho * Math.cos(phi);}
 public double im(){return rho * Math.sin(phi);}
 public double abs(){return rho;}
 public double arg(){return phi;}
}
```

Implementazione finale

Il tipo Numero Complesso può essere implementato correttamente in Java nel modo seguente:

- un'interfaccia `Complex` che definisce le firme dei metodi `re()`, `im()`, `abs()`, `arg()` e `conjugate()`;
- una classe astratta (non istanziabile) `AbstractComplex` che implementa `Complex` e realizza i metodo `toString()` e `rd3()` (questa volta implementato in modo pulito);
- due classi concrete `CartesianComplex` e `TrigComplex` che estendono `AbstractComplex` e rappresentano i complessi rispettivamente come parte reale e parte immaginaria oppure come modulo e argomento (l'argomento è sempre compreso tra 0 e 2π).
- una classe `Complexes` non istanziabile che realizza i metodi statici.

Per creare un numero complesso è necessario scegliere una rappresentazione (conviene sceglierla a seconda di come si hanno i dati) poi durante le operazioni si passa da una rappresentazione all'altra. La cosa non è efficiente ma questo esempio è puramente didattico.

Interfaccia Complex

```
public interface Complex {
    public double re();
    public double im();
    public double abs();
    public double arg();
    public Complex conjugate();
}
```

Classe AbstractComplex

```
import java.text.*;
public abstract class AbstractComplex implements Complex{
  public AbstractComplex(){nf.setMaximumFractionDigits(2);}
  public String toString(){
    if (this.im()==0) return rd3(this.re());
    String op = "+";
    if (this.im()<0) op="-";
    return  rd3(this.re())+" "+op+" i "+ rd3(Math.abs(this.im()));
  }
  private NumberFormat nf = new DecimalFormat("0.###");
  private String rd3(double x){return nf.format(x);}
}
```

Metodi statici

```
public class Complexes {
  private Complexes() {}
  public static Complex add(Complex x, Complex y){
    return new CartesianComplex(x.re()+y.re(),x.im()+y.im());
  }
  public static Complex sub(Complex x, Complex y){
    return new CartesianComplex(x.re()-y.re(), x.im()-y.im());
  }
  public static Complex times(Complex x, Complex y) {
    return new TrigComplex(x.abs()*y.abs(), x.arg()+y.arg());
  }
  public static Complex times(double x, Complex y) {
    return new TrigComplex(x*y.abs(), y.arg());
  }
  public static Complex div(Complex x, Complex y) {
    return new TrigComplex(x.abs()/y.abs(), x.arg()-y.arg());
  }
}
```

Implementazione cartesiana

```
public class CartesianComplex extends AbstractComplex {
    private static final double duepi = Math.PI*2;
    private double realPart, imgPart;

    public CartesianComplex(Complex x) {
      this.realPart=x.re();
      this.imgPart=x.im();
    }

    public CartesianComplex(double realPart, double imgPart) {
      this.realPart=realPart;
      this.imgPart=imgPart;
    }
```

```
    public double re (){return this.realPart;}
    public double im (){return this.imgPart;}
    public double abs() {
      return Math.sqrt((this.realPart * this.realPart)+
                       (this.imgPart * this.imgPart));
    }
    public double arg() {
      return (duepi+Math.atan2(this.imgPart,this.realPart))%duepi;
    }
    public Complex conjugate() {
      return new CartesianComplex(this.realPart, -this.imgPart);
    }
}
```

Implementazione trigonometrica

```
public class TrigComplex extends AbstractComplex {
    private static final double duepi = Math.PI*2;
    private double rho, phi;
    public TrigComplex(Complex x) {
      this.rho = x.abs();
      this.phi = x.arg();
    }
    public TrigComplex(double rho, double phi){
      this.rho=rho;
      this.phi=(duepi+phi%duepi)%duepi;
    }
```

```
    public double re(){return this.rho*Math.cos(phi);}
    public double im(){return this.rho*Math.sin(phi);}
    public double abs({return this.rho;}
    public double arg(){return this.phi;}
    public Complex conjugate() {
      return new TrigComplex(this.rho, -this.phi);
    }
}
```