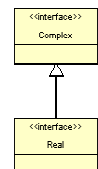


Il caso di studio Complessi /Reali: modello con interfacce

L' INTERFACCIA Real

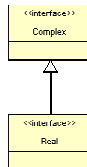
```
public interface Real extends Complex {
    public Real sum(Real x);
    public Real sub(Real x);
    public Real mul(Real x);
    public Real div(Real x);
}
```

Nota: non compare `toString()`, in quanto la sua signature è identica a quella presente in `Complex`.



IL CASO DI STUDIO Reali / Complessi

- Usiamo **interfacce** per definire le **proprietà osservabili** degli ADT e le **relazioni tra loro**
- Poiché nella realtà i reali sono un sottoinsieme dei complessi, stabiliamo le seguenti **relazioni fra interfacce**:
 - `Complex` è l'interfaccia-base
 - `Real` la specializza



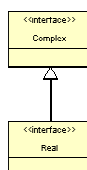
IL PROGETTO DELLE CLASSI

- Usiamo **classi** per implementare le interfacce **in modo efficiente** dal punto di vista dell'uso delle risorse
 - `RealNum` implementa `Real`
 - `ComplexNum` implementa `Complex`
- Non è detto che `RealNum` e `ComplexNum` siano in una qualche relazione fra loro: anzi, potrebbero benissimo essere due classi del tutto indipendenti
- ... oppure anche derivate una dall'altra.

L' INTERFACCIA Complex

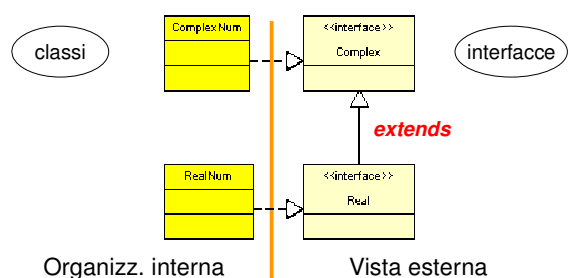
```
public interface Complex {
    public double getReal();
    public double getIm();
    public double module();
    public Complex cgt();
    public Complex divByFactor(double x);
    public Complex sum(Complex z);
    public Complex sub(Complex z);
    public Complex mul(Complex z);
    public Complex div(Complex z);
    public String toString();
}
```

Non ci sono dati, ma è bene prevedere metodi di accesso alle informazioni caratterizzanti (anche modulo e arg... ?)



REALI E COMPLESSI: IL PROGETTO

L'architettura complessiva:



LA CLASSE RealNum

```
public class RealNum implements Real {
    protected double re;
    public RealNum() { re=0; }
    public RealNum(double x) { re=x; }
    //
    public double getReal() { return re; }
    public double getIm() { return 0; }
    //
    public double module() { return re<0 ? -re : re; }
    // operazioni con entrambi operandi reali
    public Real sum(Real x){return new RealNum(re + x.getReal());}
    public Real sub(Real x){return new RealNum(re - x.getReal());}
    public Real mul(Real x){return new RealNum(re * x.getReal());}
    public Real div(Real x){return new RealNum(re / x.getReal());}
    ...
}
```

LA CLASSE ComplexNum

```
...
public Complex mul(Complex z) {
    return new ComplexNum(
        re*z.getReal()-im*z.getIm(),
        re*z.getIm()+im*z.getReal()); }
public Complex div(Complex z) {
    double mod = z.module();
    return mul(z.cgt()).divByFactor(mod*mod);
}
public String toString() { // stampa di un ComplexNum
    String res;
    if (re==0.0 && im==0.0) return "0";
    if (re==0.0) res = ""; else
    { res = Double.toString(re); if (im>=0.0) res += "+"; }
    res += (im==1 || im==-1 ? "" : Double.toString(im)) + "i";
    return res;
}
}
```

LA CLASSE RealNum

```
...
// operazioni con this reale, secondo operando Complex
public Complex sum(Complex z) {
    return new ComplexNum( re+z.getReal(), z.getIm() ); }
public Complex sub(Complex z) {
    return new ComplexNum( re-z.getReal(), z.getIm() ); }
public Complex mul(Complex z) {
    return new ComplexNum( re*z.getReal(), re*z.getIm() ); }
public Complex div(Complex z) {
    return new ComplexNum( re/z.getReal(), re/z.getIm() ); }
public Complex cgt() { return this; }
public Complex divByFactor(double x) {
    return new RealNum(re/x); }
public String toString() {
    return Double.toString(re); }
}
```

Il coniugato di un Real è il Real stesso

Il risultato è in effetti un Real, ma l'interfaccia prevede un generico Complex

UNA PROVA

```
public class Prova {
    public static void main(String args[]){
        Real r1 = new RealNum(18.5), r2 = new RealNum(3.14);
        Complex c1 = new ComplexNum(-16, 0), c2 = new ComplexNum(3, 2),
        c3 = new ComplexNum(0, -2);
        Real r = r1.sum(r2);
        Complex c = c1.sum(c2);
        System.out.println("r1 + r2 = " + r); // il reale 21.64
        System.out.println("c1 + c2 = " + c); // il complesso -13+2i
        c = c.sum(c3);
        System.out.println("c + c3 = " + c); // il complesso -13+0i
        c = r;
        System.out.println("c = r; c = " + c); // Qui c è reale
        // POLIMORFISMO: scatta la toString dei reali --> 21.64
    }
}
```

```
C:\esercizi>java Prova
r1 + r2 = 21.64
c1 + c2 = -13.0+2.0i
c + c3 = -13.0+0.0i
c = r; c = 21.64
```

LA CLASSE ComplexNum

```
public class ComplexNum implements Complex {
    protected double re, im;
    public ComplexNum() { re = im = 0; }
    public ComplexNum(double x) { re = x; im = 0; }
    public ComplexNum(double x, double y) { re = x; im = y; }
    public double getReal() { return re; }
    public double getIm() { return im; }
    public double module() { return Math.sqrt(re*re+im*im); }
    public Complex cgt() { return new ComplexNum(re, -im); }
    public Complex divByFactor(double x) {
        return new ComplexNum(re/x, im/x); }
    public Complex sum(Complex z) {
        return new ComplexNum( re+z.getReal(), im+z.getIm()); }
    public Complex sub(Complex z) {
        return new ComplexNum( re-z.getReal(), im-z.getIm()); }
    ...
}
```

UNA RIFLESSIONE

- Nel programma precedente, si usano *quasi sempre solo interfacce...* **tranne che all'atto della creazione degli oggetti:**

```
Real r2 = new RealNum(3.14);
Complex c2 = new ComplexNum(3,2);
Real r = r1.sum(r2);
Complex c = c1.sum(c2);
```

- **Dover specificare le classi toglie flessibilità al progetto, perché ingessa la scelta di usare quell'implementazione..**
- ...quando tutto il resto sarebbe indipendente!

UN APPROCCIO ALTERNATIVO

- Sarebbe utile che il programma cliente **usasse sempre e solo interfacce**, **nascondendo la fase di creazione degli oggetti**

```
Real    r2 = il reale di valore 3.14
Complex c2 = il complesso di valore 3+2i
Real    r = r1.sum(r2);
Complex c = c1.sum(c2);
```

- In tal modo è possibile **cambiare in ogni momento l'implementazione**, senza impatto sui clienti.
- Ciò si ottiene **delegando la creazione dell'oggetto a una FABBRICA (FACTORY)**

LA CLASSE "FACTORY"

La classe "factory" NumFactory

- è l'unica **classe pubblica** del package
- fornisce **funzioni statiche** che **incapsulano la new delle istanze di ComplexNum e RealNum**

NumFactory
<pre>+getNewReal() : Real +getNewReal(x:double) : Real +getNewComplex() : Complex +getNewComplex(x:double) : Complex +getNewComplex(x:double, y:double) : Complex</pre>

NumFactory restituisce formalmente riferimenti a Real o Complex: le classi RealNum o ComplexNum fuori non si vedono mai !

IL PATTERN "FABBRICA" (factory)

- La **fabbrica nasconde agli utenti la costruzione degli oggetti**
- Dunque, **l'utente può non sapere di che classe è l'oggetto: basta che ne conosca l'interfaccia!**
 - le classi NON sono più PUBBLICHE!
 - la funzione statica che restituisce l'oggetto richiesto, già pronto per l'uso, specifica nella sua signature solo l'INTERFACCIA



LA CLASSE "FACTORY"

```
package myNumbers;

public class NumberFactory {

    public static Real getNewReal() {
        return new RealNum();
    }

    public static Real getNewReal(double x) {
        return new RealNum(x);
    }

    public static Complex getNewComplex() {
        return new ComplexNum();
    }

    public static Complex getNewComplex(double x) {
        return new ComplexNum(x);
    }

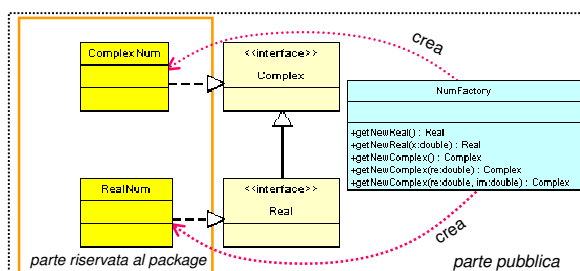
    public static Complex getNewComplex(double x, double y) {
        return new ComplexNum(x,y);
    }

}
```

NumFactory restituisce formalmente riferimenti a Real o Complex: le classi RealNum o ComplexNum fuori non si vedono mai !

UN PROGETTO ALTERNATIVO

L'architettura complessiva (package)



LA CLASSE DI PROVA

```
import myNumbers.*;
public class Prova {
    public static void main(String args[]){
        Real    r1 = NumFactory.getNewReal(18.5),
               r2 = NumFactory.getNewReal(3.14);
        Complex c1 = NumFactory.getNewComplex(-16, 0),
               c2 = NumFactory.getNewComplex(3, 2),
               c3 = NumFactory.getNewComplex(0, -2);

        Real    r = r1.sum(r2);      Complex c = c1.sum(c2);
        System.out.println("r1 + r2 = " + r);      // il reale 21.64
        System.out.println("c1 + c2 = " + c);      // il complesso -13+2i
        System.out.println("c1 + c2 - i = " + c.sub(new Complex(0,1)));
        c = c.sum(c3);
        System.out.println("c + c3 = " + c);      // il complesso -13+0i
        c = r;
        System.out.println("c = r; c = " + c);    // Qui c è reale
    }
}
```

Creazione indiretta di oggetti

```
C:\esercizi>java Prova
r1 + r2 = 21.64
c1 + c2 = -13.0+2.0i
c + c3 = -13.0+0.0i
c = r; c = 21.64
```