INTERFACCE E PROGETTO

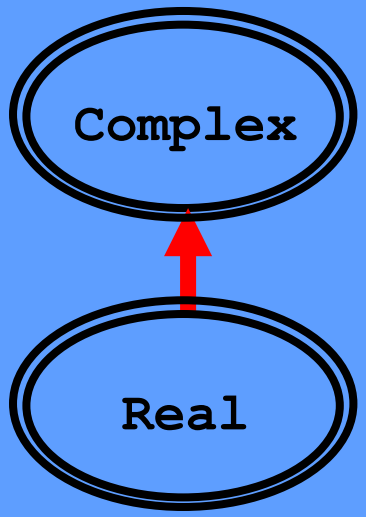
Le interfacce inducono un diverso *modo di concepire il progetto*

- **prima si definiscono le interfacce che servono, e si stabiliscono le relazioni tra loro**
  - **la gerarchia delle interfacce riflette *scelte di progetto* *(pulizia concettuale)***
- **poi si creano le classi che implementano tali interfacce**
  - **la gerarchia delle classi riflette *scelte implementative* *(efficienza ed efficacia)***

IL CASO DI STUDIO Reali / Complessi

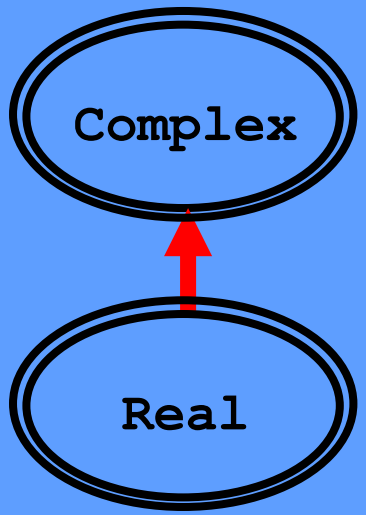
- **Usiamo *interfacce* per definire le *proprietà osservabili* degli ADT e le *relazioni tra loro***

- **Poiché nella realtà i reali sono un sottoin-sieme dei complessi, stabiliamo le seguenti relazioni *fra interfacce*:**
  - **`Complex` è l'interfaccia-base**
  - **`Real` la specializza**



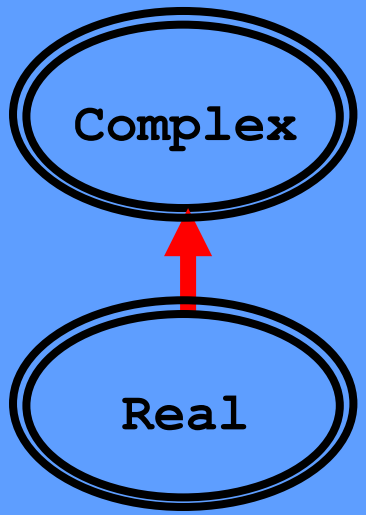
L' INTERFACCIA Complex

```
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);
}
```



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);
}
```

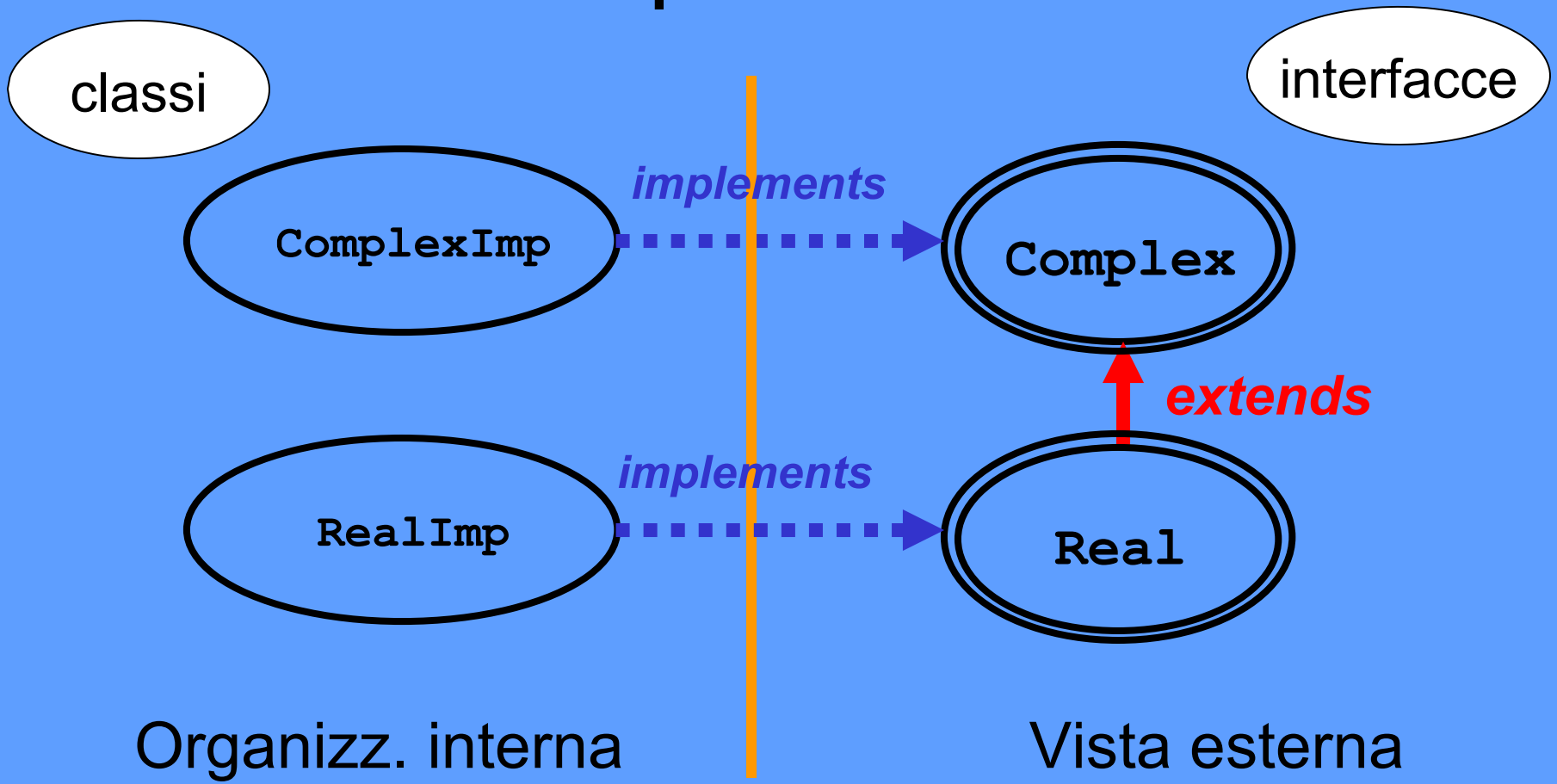


IL PROGETTO DELLE CLASSI

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

REALI E COMPLESSI: IL PROGETTO

L'architettura complessiva:



LA CLASSE `RealImp`

```
public class RealImp implements Real {
 protected double re;
 public RealImp() { re=0; }
 public RealImp(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 RealImp(re + x.getReal());}
 public Real sub(Real x){return new RealImp(re - x.getReal());}
 public Real mul(Real x){return new RealImp(re * x.getReal());}
 public Real div(Real x){return new RealImp(re / x.getReal());}
 ...
```

LA CLASSE `RealImp`

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

LA CLASSE ComplexImp

```
public class ComplexImp implements Complex {
 protected double re, im;
 public ComplexImp() { re = im = 0; }
 public ComplexImp(double x) { re = x; im = 0; }
 public ComplexImp(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 ComplexImp(re, -im); }
 public Complex divByFactor(double x) {
   return new ComplexImp(re/x, im/x); }
 public Complex sum(Complex z) {
   return new ComplexImp( re+z.getReal(), im+z.getIm()); }
 public Complex sub(Complex z) {
   return new ComplexImp( re-z.getReal(), im-z.getIm()); }
 ...
```

LA CLASSE ComplexImp

```
...
public Complex mul(Complex z) {
  return new ComplexImp(
  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 ComplexImp
  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;
}
}
```

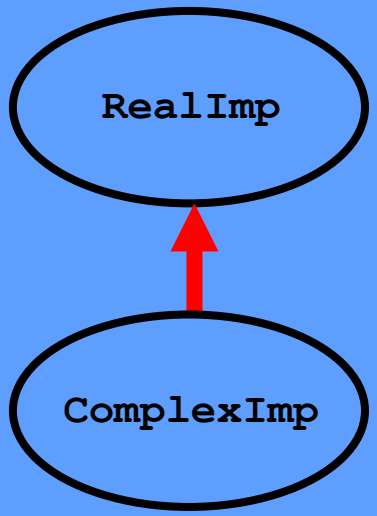
UNA PROVA

```
public class Prova {
 public static void main(String args[]){
   Real        r1 = new RealImp(18.5),      r2 = new RealImp(3.14);
   Complex     c1 = new ComplexImp(-16, 0), c2 = new ComplexImp(3, 2),
               c3 = new ComplexImp(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
```

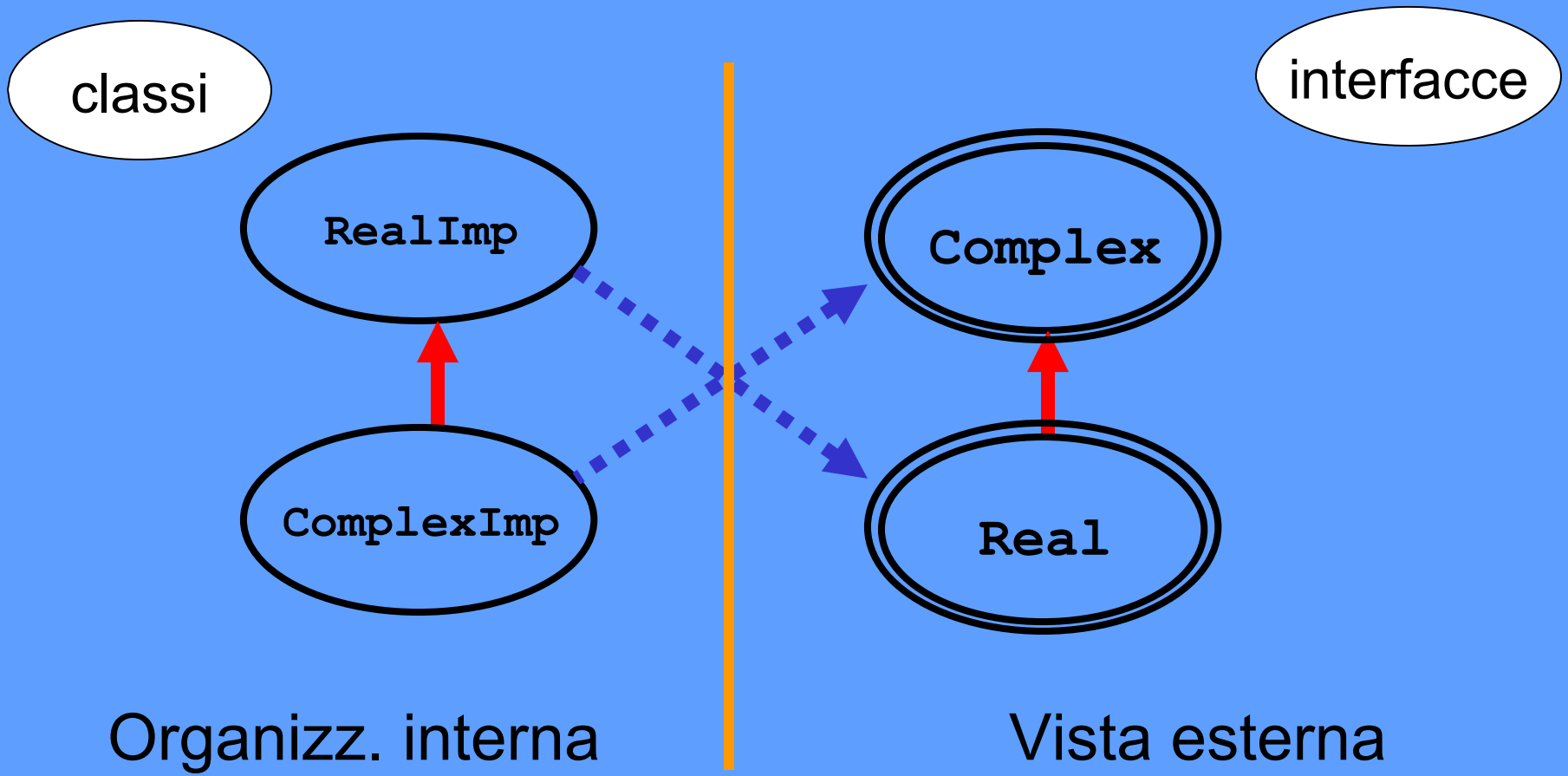
UN PROGETTO ALTERNATIVO

- In alternativa si può decidere di derivare una classe dall'altra. Va però tenuto *ben presente* che la relazione fra tali classi non riflette una realtà, ma solo una *scelta implementativa*
- Perciò, tale relazione *non dovrà essere pubblicizzata* all'esterno: da fuori si dovranno *usare sempre e solo le interfacce.*
- Sotto questo vincolo, una possibile realizzazione concreta può prevedere
  - `RealImp` come classe-base
  - `ComplexImp` come specializz.



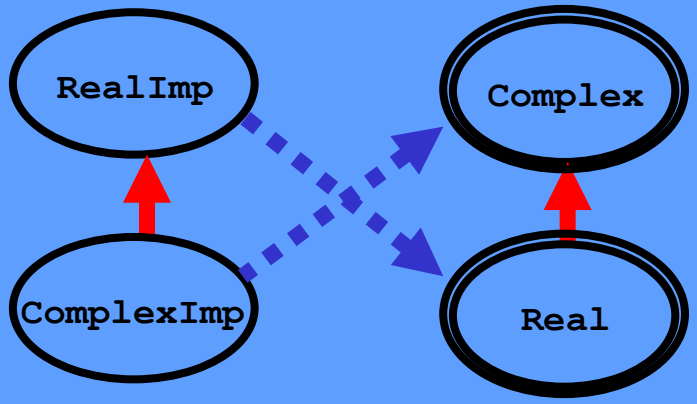
UN PROGETTO ALTERNATIVO

L'architettura che ne deriva:



CLASSI: COSA CAMBIA

- **`RealImp` implementa `Real`**
  - definisce e usa un solo float internamente, come prima
  - ***non cambia niente, va già bene così com'è***
- **`ComplexImp` implementa `Complex`**
  - eredita già un float (parte reale) da RealImp
  - **i costruttori vanno adattati** *(super)*
  - ***alcuni metodi ereditati vanno già bene così*** *(sum, sub, getReal)*
  - **altri vanno specializzati.**
- **`Prova` usa solo interfacce**
  → *resta identica*



LA NUOVA CLASSE `ComplexImp`

```
public class ComplexImp extends RealImp implements Complex {
 protected double im;
 public ComplexImp() { super(); im=0; }
 public ComplexImp(double x) { super(x); im = 0; }
 public ComplexImp (double x, double y) { super(x); im = y; }
 public double getIm() { return im; }
 // risparmia di riscrivere getReal()
 public double module() { return Math.sqrt(re*re+im*im); }
 public Complex cgt() { return new ComplexImp(re, -im); }
 public Complex divByFactor(double x) {
   return new ComplexImp(re/x, im/x); }
 // risparmia di riscrivere sum() e sub()
  ...
```

LA NUOVA CLASSE `ComplexImp`

```
 ...
 public Complex mul(Complex z) {
 return new ComplexImp(
   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() {
   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 + "+";}
   res += (im==1 || im==-1 ? "" : Double.toString(im)) + "i";
   return res;
 }
}
```

LA STESSA PROVA

```
public class Prova {
 public static void main(String args[]){
   Real       r1 = new RealImp(18.5),      r2 = new RealImp(3.14);
   Complex    c1 = new ComplexImp(-16, 0), c2 = new ComplexImp(3, 2),
   c3 = new ComplexImp(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)));
   // il complesso -13+i
   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
```

CONCLUSIONE

- **Un progetto basato su interfacce consente di *concentrarsi sul modello della realtà***
  - **senza però *pagare inefficienze***
  - ***le classi* che implementano le interfacce possono essere definite curando l'aspetto dell'efficienza**
- **Non è molto utile "intrecciare" le gerarchie**
  - **nell'esempio visto, il guadagno nel secondo caso è minimo, ma la comprensibilità ne risente alquanto**
- **Un progetto basato su interfacce è possibile *anche in modelli privi del concetto di classe***
  - **le classi sono *implementazioni di tipi* più che definizioni di tipi -- le interfacce definiscono ADT**