

NON ABBIAMO ANCORA CORRETTO LE PROVETTE!!!

OO in Java: classi astratte, interfacce, classi interne

Stefano Mizzaro

Dipartimento di matematica e informatica
Università di Udine
<http://www.dimi.uniud.it/mizzaro/>
mizzaro@dimi.uniud.it
Programmazione, lezione 19
18 gennaio 2007

Riassunto: programmazione strutturata

- Mattoni di base
- Sequenza, selezione, iterazione
- Array
- Sottoprogrammi, metodi, passaggio parametri
- Ricorsione

Stefano Mizzaro - OO in Java

3

Riassunto: cos'è la OOP?

- TDA
 - Incapsulamento, interfaccia, implementazione
 - Composizione, uso
- Scambio messaggi
 - Oggetti (istanze) attivi che si "parlano"
 - Metodi/attributi d'istanza e di classe
 - Consente (ereditarietà e) polimorfismo
- Ereditarietà
 - Estensione, sovrascrittura, sottotipo, **extends**, **super**, relazioni fra classi...
- Polimorfismo
 - Maniglie, late binding, "talk to the base class"

Stefano Mizzaro - OO in Java

4

Cosa manca

- 6 lezioni
 - 1. OO in Java (ultima lez. ""teorica"")
 - API
 - 2. Sguardo dall'alto
 - 3. Eccezioni, File, ...
 - 4. e 5. AWT (per GUI); cenni applet
 - 6. Cenni computabilità
- (... e un sacco di altre cose che non vediamo...)

Stefano Mizzaro - OO in Java

5

Oggi

- OO in Java:
 - Classi astratte
 - Interfacce
 - Classi interne (cenni)

Stefano Mizzaro - OO in Java

6

Figure...

```

classDiagram
    class Figura {
        <draw()
    }
    class Cerchio {
        <draw()
    }
    class Punto {
        <draw()
    }
    class Linea {
        <draw()
    }
    Figura <|-- Cerchio
    Figura <|-- Punto
    Figura <|-- Linea
    
```

Stefano Mizzaro - OO in Java 7

Figura CON area () ?

- Aggiungiamo un metodo `area ()`
- Però: cerchi, punti e quadrati sanno calcolare la propria area
- E una figura generica?
- Uhm...

```

classDiagram
    class Figura {
        <area()
    }
    class Cerchio {
        <area()
    }
    class Punto {
        <area()
    }
    class Quadrato {
        <area()
    }
    Figura <|-- Cerchio
    Figura <|-- Punto
    Figura <|-- Quadrato
    Cerchio o--> Punto
    Quadrato o--> Punto
    
```

```

Cerchio c = new Cerchio();
Figura f = new Cerchio();
System.out.println(c.area());
System.out.println(f.area());
    
```

Stefano Mizzaro - OO in Java 8

Figura senza area () ?

- Una figura generica non sa calcolare la propria area ⇒ niente `area ()` in `Figura!`
- Ma se non metto `area ()` in `Figura...`

```

Cerchio c = new Cerchio();
Figura c1 = new Cerchio();
Figura f = new Figura();
System.out.println(c.area());
System.out.println(f.area());
System.out.println(c1.area());
    
```

```

classDiagram
    class Figura {
    }
    class Cerchio {
        <area()
    }
    class Punto {
        <area()
    }
    class Quadrato {
        <area()
    }
    Figura <|-- Cerchio
    Figura <|-- Punto
    Figura <|-- Quadrato
    
```

```

cannot resolve symbol
symbol : method area ()
location: class Figura
f.area ();
    
```

Stefano Mizzaro 9

Problema: Figura e area ()

- Se metto `area ()` in `Figura`
 - Non so cosa deve fare
- Se non metto `area ()` in `Figura`
 - No polimorfismo, perché non posso essere sicuro che le istanze di `Figura` (o sottoclasse) hanno `area ()`
- Soluzione: classi e metodi **astratti**

Stefano Mizzaro - OO in Java 10

Figura e area () astratti

- Soluzione:
 - `area ()` in `Figura` è **astratto** (non invocabile)
 - anche `Figura` è **astratta** (non istanziabile)
- Astratto (metodo o classe) = non completamente specificato

```

classDiagram
    class Figura {
        <area()
    }
    class Cerchio {
        <area()
    }
    class Punto {
        <area()
    }
    class Quadrato {
        <area()
    }
    Figura <|-- Cerchio
    Figura <|-- Punto
    Figura <|-- Quadrato
    
```

Stefano Mizzaro - OO in Java 11

In Java

- Se dichiaro

```

abstract class Figura {
    abstract double area();
}
    
```

poi posso scrivere

```

Cerchio c = new Cerchio();
Figura f = new Cerchio();
System.out.println(c.area());
System.out.println(f.area());

f = new Figura();
System.out.println(f.area());
System.out.println((Figura)c.area());
    
```

```

classDiagram
    class Figura {
        <area()
    }
    class Cerchio {
        <area()
    }
    class Punto {
        <area()
    }
    class Quadrato {
        <area()
    }
    Figura <|-- Cerchio
    Figura <|-- Punto
    Figura <|-- Quadrato
    
```

Stefano Mizzaro - OO in Java 12

Classi e metodi astratti

- Una classe astratta non può essere istanziata
- Metodo astratto: lasciato non specificato
- Se una classe contiene metodi astratti ⇒ deve essere astratta (cosa succederebbe invocando un metodo astratto?)
- N.B. Non è detto che una classe astratta contenga metodi astratti
 - Se una classe è astratta ~~⇒~~ deve contenere metodi astratti

Stefano Mizzaro - OO in Java

13

Utilità delle classi astratte

- Scomposizione problema: alcune classi della mia gerarchia sono lasciate non specificate (e verranno poi specializzate)
- Impedisce la creazione di istanze di quella classe
- Permette il polimorfismo

Stefano Mizzaro - OO in Java

14

Postille

- Metodi **static** in classe **abstract**: OK (e si possono usare)
- Metodo **abstract static**: NO
- Come si può impedire la creazione di istanze di una classe?
 - Con costruttore privato e classe **final** (e quindi non estendibile)

Stefano Mizzaro - OO in Java

15

Scaletta

- OO in Java:
 - Classi astratte
 - Interfacce
 - Classi interne (cenni)

Stefano Mizzaro - OO in Java

16

Interfacce

- "Classi"
 - completamente astratte (non specificate)
 - senza attributi
 - solo metodi d'istanza astratti (e costanti)
- **interface I {...}**
- Una classe **A** può **implementare I**:


```
class A implements I {...}
```
- **A** definisce tutti i metodi di **I**

Stefano Mizzaro - OO in Java

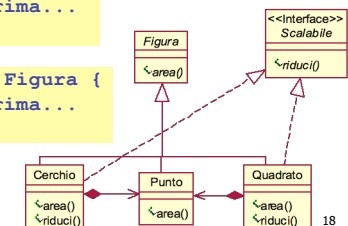
17

Figure scalabili (1/3)

```
interface Scalabile {
    public void riduci (int scala);
}
```

```
abstract class Figura {
    ...//tutto come prima...
}
```

```
class Punto extends Figura {
    ...//tutto come prima...
}
```



Stefano Mizzaro - OO in Java

18

Figure scalabili (2/3)

```
class Cerchio extends Figura implements Scalabile {
    ...//tutto come prima...
    public void riduci (int scala) {
        raggio = raggio * scala / 100;
    }
}

class Quadrato extends Figura
    implements Scalabile {
    ...//tutto come prima...
    public void riduci (int scala){
        ...
    }
}
```

Stefano Mizzaro - OO in Java 19

Figure scalabili (3/3)

■ Ora posso scrivere

```
Scalabile s1 = new Quadrato();
Scalabile s2 = new Cerchio();
s1.riduci(50);
Scalabile[] s = new Scalabile[10];
...
s[i] = new Cerchio();
s[j] = new Quadrato();
...
s[i].riduci(50);
s[j].riduci(20);
```

Stefano Mizzaro - OO in Java 20

Interfacce

- Oltre alle classi: interfacce
- Classi che implementano interfacce devono implementarne i metodi:
 - se `c implements I` siamo sicuri (anche il compilatore!) che gli oggetti di `c` hanno i metodi di `I`
 - (oppure `c` è astratta)

```
abstract class NonSoComeRidurmi implements Scalabile{
    // senza riduci()
}
```

Stefano Mizzaro - OO in Java 21

Interfacce ed eredità

- Un'interfaccia `J` può ereditare da un'interfaccia `I`
- Una classe (non astratta) `C` che implementa `J` deve implementare i metodi anche di `I`

Stefano Mizzaro - OO in Java 22

Eredità singola e multipla

- Eredità singola
 - Ogni classe è sottoclasse diretta di un'unica superclasse
- Eredità multipla
 - Le classi possono avere più superclassi dirette

Stefano Mizzaro - OO in Java 23

Eredità multipla: pro...

- Gerarchie nel mondo reale: non sempre a eredità singola
- `Cerchio` sottoclasse di
 - `Figura`
 - `Scalabile` (oggetti che possono ridursi)
- `Cerchio` eredita i metodi sia di `Figura` sia di `Scalabile`

Stefano Mizzaro - OO in Java 24

...e contro

```

classDiagram
    class B
    class C
    class A
    B --|> A
    C --|> A
    
```

- Se sia **B** sia **C** definiscono **m()**
- Da chi viene ereditato **m()**?
- Quale **m()** va eseguito alla chiamata **x.m()**?

Stefano Mizzaro - OO in Java 25

Eredità multipla in Java?

- In Java: **eredità singola**
- In C++: eredità multipla
- Però in Java l'eredità multipla "rientra dalla finestra"

```

class A extends B, C {
    ...
}


```

Stefano Mizzaro - OO in Java 26

Eredità multipla con interfacce

```

interface I1 {...}
interface I2 {...}
class C implements I1, I2 {
    ...
}

interface I3 extends I1, I2 {}
class D implements I3 {...}

```

Stefano Mizzaro - OO in Java 27

Eredità multipla con interfacce

- Quindi:
 - Un'interfaccia può estendere più interfacce
 - Una classe può implementare più interfacce
- (interfacce ≠ classi astratte)
- Perché?
 - No eredità multipla dell'implementazione

Stefano Mizzaro - OO in Java 28

Utilità delle interfacce (1/3)

- Un'altra possibilità per il polimorfismo
- Es.: array di oggetti scalabili

```

Scalabile[] s = new Scalabile[10];
s[i] = new Cerchio();
s[j] = new Quadrato();
...
s[i].riduci(50);
s[j].riduci(20);

```

Stefano Mizzaro - OO in Java 29

Utilità delle interfacce (2/3)

- Definizione costanti
- Oltre a metodi d'istanza astratti, anche attributi!
- Però **static** e **final** (ossia: costanti)
- Più comodo di **CostantiUtili.PI**

```

interface CostantiUtili{
    public static final double PI = 3.14159;
    public static final double E = 2.71828;
}

class C implements CostantiUtili{
    ... PI ... E ...
}

```

Stefano Mizzaro - OO in Java 30

Utilità delle interfacce (3/3)

- **Marcatori** di oggetti, per sapere se un oggetto è istanza di una classe che implementa un'interfaccia
- Ad esempio la clonabilità è gestita con


```
interface Cloneable { }
```
- Per verificare se una classe è clonabile:


```
if (x instanceof Cloneable) {
    ... operazioni di duplicazione ...
} else System.out.println("Non clonabile");
```

Stefano Mizzaro - OO in Java 31

Utilità classi astratte - bis (1/2)

- "Pattern" comune:
 - Classi astratte che forniscono implementazioni parziali o banali di un'interfaccia
- Esempio:
 - Classe c che deve implementare un'interfaccia I con tanti metodi...
 - ... ma è interessata solo a uno di questi (gli altri vuoti)
 - Se c'è una classe "adattatore"...

Stefano Mizzaro - OO in Java 32

Utilità classi astratte - bis (2/2)

```
interface I {
    public void m1();
    public void m2();
    ...
    public void mn();
}
```

```
abstract class A
implements I {
    public void m1(){}
    public void m2(){}
    ...
    public void mn(){}
}
```

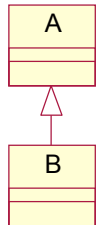
```
class Cn extends A {
    public void mn(){...}
}
class C2 extends A {
    public void m2(){...}
}
class C1 extends A {
    public void m1(){...}
}
```

```
class C3 implements I {
class C2 implements I {
class C1 implements I {
    public void m1(){...}
    public void m2(){...}
    ...
    public void mn(){...}
}
```

Stefano Mizzaro - OO in Java 33

Conversioni di tipo e cast

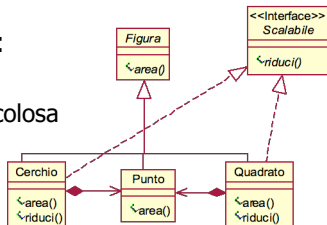
- Con i tipi predefiniti:
 - Promozione (implicita): da `byte` ad `int`
 - Cast (esplicito): da `int` a `byte`
- Con le classi
 - Promozione (implicita): da `B` ad `A`
 - Cast (esplicito): da `A` a `B`
- Implicito: non crea problemi



Stefano Mizzaro - OO in Java 34

Cast rivisitato: upcast

- Upcast/Promozione:
 - Di solito implicita
 - Ok perché non pericolosa



```
Figura[] f = new Figura[...];
f[i] = new Punto();
f[i] = (Figura) new Punto();
```

Stefano Mizzaro - OO in Java 35

Cast rivisitato: downcast

- Esplicito e pericoloso
- Se so che `f[i]` è un `Punto`, posso chiamare i suoi metodi, previo cast
 - Ma se poi `f[i]` non è un `Punto`, errore in esecuzione
- A volte downcast non fino "in fondo" alla gerarchia (array di `Object`, contenente una `Figura`...)
- Ex.: completare e sperimentare

```
Figura[] f = new Figura[...];
f[i] = new Punto();
f[j] = new Cerchio();
f[i].setX(2.0);
f[j].setX(2.0);
(Punto) f[i].setX(2.0);
(Punto) f[j].setX(2.0);
```

Stefano Mizzaro - OO in Java 36

Scaletta

- OO in Java:
 - Classi astratte
 - Interfacce
 - Classi interne (cenni)

Stefano Mizzaro - OO in Java

37

Classi interne

- Classi dentro a un'altra classe
- Classi **membro**
- Classi **locali** (a un metodo)
 - Classi **anonime**
- Le uniche differenze sono sulla visibilità
- Consiglio: per ora non usatele...

Stefano Mizzaro - OO in Java

38

Esempio

```
class Classe {
    class Membro extends A {
        ...
    }
    public void m() {
        class Locale extends A {
            ...
        }
        Membro m = new Membro();
        Locale l = new Locale();
        A x = (new A() {
            ...
        });
    }
}
```

- Anche se **A** è un'interfaccia!
(**extends**→**implements**)

Stefano Mizzaro - OO in Java

39

Riassunto

- OO in Java:
 - Classi astratte
 - Interfacce
 - Classi interne (cenni)
- Argomenti solo accennati
 - Vedremo esempi
 - Li approfondirete nei prossimi corsi
 - Non mi aspetto che siate in grado di scrivere programmi che usano qs. cose
 - Mi aspetto che siate in grado di capire semplici programmi che le usano...
 - ... e di rispondere a domande teoriche

Stefano Mizzaro - OO in Java

40