Esercitazioni di Informatica 3

Programmazione object-oriented

Paolo Costa

paolo.costa@polimi.it

Lucidi a cura di M.Pradella – P.Spoletini (2002)

Ereditarietà

A partire dalla classe 'persona' definire le classi 'studente' e 'professore':

```
class Persona {
      String nome;
      Persona(String nome) {
            this.nome = nome;
      }
      public void print() {
            System.out.println("Il mio nome è" +
            nome);
      }
}
```

Classe studente

```
class Studente extends Persona {
        float media;
        Studente(String nome, float media) {
                super(nome);
                this.media = media;
        }
        public void print() {
                System.out.println("Il mio nome è
                " + nome + " e la mia media è " +
                media);
        }
}
```

Classe professore

```
class Professore extends Persona {
      int pubblicazioni;
      Studente(String nome,int pubblicazioni){
            super(nome);
            this.pubblicazioni=pubblicazioni;
      }
      public void print() {
            System.out.println("Il mio nome è
            " + nome + " e ho " +
            pubblicazioni " articoli");
      }
}
```

Esempio: main()

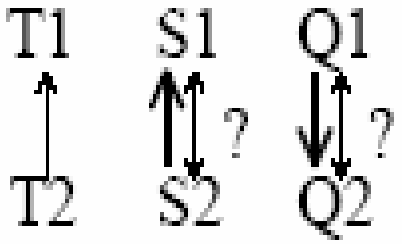
```
public static void main(String args[]) {
      Persona p = new Persona("Geometra Filini");
      p.print();
      p = new Studente("Pierino", 18);
      p.print();
      p = new Professore("Oronzo Canà", 14);
      p.print();
}
```

NOTA: In java i metodi sono ridefinibili tramite override per default (in C++ è necessaria la parola chiave `virtual`). In tal modo è possibile realizzare il polimorfismo costringendo il sistema a eseguire un link dinamico dello stesso metodo

Inheritance and type system

- We wish a subclass to define a subtype
- NB: type, set (better: *algebra*), class should be the same thing but ...
- The relation between a type and a subtype is *substitutability*
- If we wish to achieve strong typing and polymorphism, the subtype can
  - add new operations
  - redefine operations preserving contravariance of input (new domain is a superset of old one) and covariance of output (new range is a subset of old one) parameters
    - in C++, Java, Object Pascal, and Modula3 they must be the same type;
    - Eiffel and Ada require covariance of both parameter and result

Covariance and Contravariance



s1 = t.f(q1);

- The dynamic type of the result must be defined by a subclass of S1 (*covariance*)
- The dynamic type of q1 must be defined by a superclass of Q1 (*contravariance*)

OO: Type Checking

```
class a {                                   class b extends a {
       int a1;                                     int b1;
       a2 f(a3 p) {…}                              b2 f(b3 p) {…}
}                                           }
1. a x = new a; b y = new b;
2. h1 = x. f(…);
3. x = y;
4. h2 = x. f(…);
5. y.b1 = x.a1;
6. x.a1 = 0;
7. ...
8. y = x;
9. ....
10. a x = new b;
11. h3 = x. f(…);
12. x.b1 = 0;
```

- Linguaggio object-oriented fortemente tipizzato in cui gli oggetti sono dinamicamente allocati nello heap e valgono le regole di polimorfismo e binding dinamico.
- Ciò che non appare dichiarato qui, come le classi a2, a3, b2, b3 e le variabili h1, h2, h3, è dichiarato altrove e è visibile staticamente al frammento.

$\rightarrow$Analisi statica: istruzioni scorrette dal punto di vista del type checking o corrette solo sotto certe ipotesi da fare sulle parti di programma qui non mostrate.

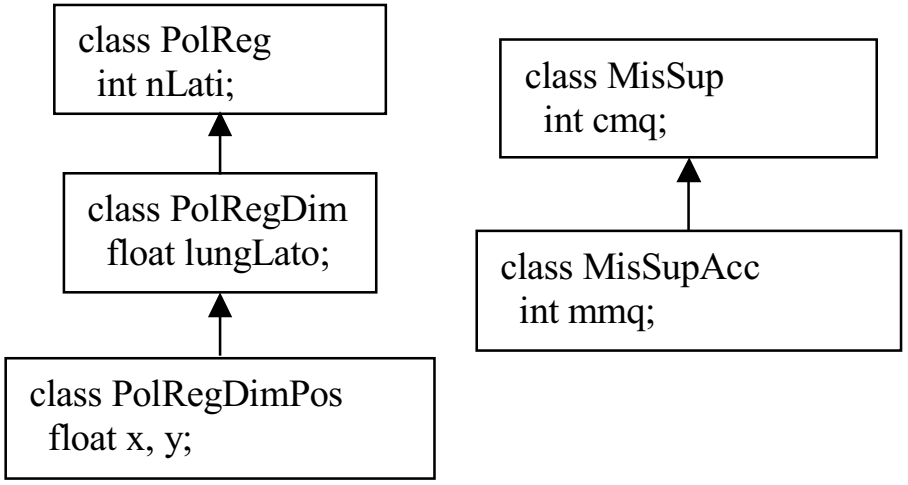
Type checking (2)

Hp.: metodi e attributi delle classi pubblici per evitare errori in compilazione.

- La ridefinizione di f in b deve soddisfare alle regole di covarianza per i risultati e controvarianza per i parametri. Solo sotto questa ipotesi, a runtime le istruzioni 4 e 11 (durante la cui esecuzione x risulta di tipo dinamico b) non generano errori.
- Le variabili h1, h2 e h3 devono essere di tipo a2 o di un supertipo di a2.
- L'istruzione 8 è scorretta: non si può assegnare un oggetto di un tipo a una variabile che fa riferimento a un suo sottotipo.
- L'istruzione 10 è problematica. Se si trova nello stesso "scope" della precedente dichiarazione di a, questa è una doppia dichiarazione (e quindi un errore).
- L'istruzione 12 è scorretta, in quanto il type checking viene fatto in base al tipo statico di x (che è a). La classe a non ha un attributo pubblico b1.

OO

In un ipotetico linguaggio orientato agli oggetti si definiscono le tre classi *PolReg, PolRegDim* e *PolRegDimPos*, che intendono rappresentare tre nozioni di poligono regolare, via via più precise, aggiungendo attributi relativi alla dimensione del lato e alla posizione (coordinate cartesiane) del centro. Similmente le due classi *MisSup*, *MisSupAcc* danno rispettivamente una misura di superficie in centimetri quadrati e una più accurata, aggiungendo un attributo per i millimetri quadrati.



Si consideri ora la classe *Geometria*, con un metodo pubblico *supPol* per calcolare la superficie di un poligono regolare:

```
 class Geometria  { ...
      public  MisSup supPol (PolRegDim pol){...}
... }
```

e si immagini di voler introdurre una classe erede, da essa derivata, che ridefinisca il metodo *supPol*, nei seguenti quattro modi

```
 class GeometriaDerivata1 inherits Geometria { ...
      public  MisSupAcc supPol (PolRegDimPos pol){...}
... }
 class GeometriaDerivata2 inherits Geometria { ...
      public  MisSupAcc supPol (PolReg pol){...} ... }
 class GeometriaDerivata3 inherits Geometria { ...
      public  MisSup supPol (PolReg pol){...} ... }
 class GeometriaDerivata4 inherits Geometria { ...
      public  MisSup supPol (PolRegDim pol){...} ... }
```

Si indichi, scrivendo SI o NO nella seguente tabella, se ognuna delle classi GeometriaDerivata1, GeometriaDerivata2, GeometriaDerivata3 e GeometriaDerivata4 è corretta per quanto riguarda la ridefinizione del metodo, secondo il principio di sostuibilità, o secondo le regole dei linguaggi C++ e Java.

	SOSTITUIBILITÀ	**C++ e Java**
GeometriaDerivata1	***NO***	***NO***
GeometriaDerivata2	***SI***	***NO***
GeometriaDerivata3	***SI***	***NO***
GeometriaDerivata4	***SI***	***SI***