



Overloading - Overriding

Overloading:

Funzioni con uguale nome e diversa firma possono coesistere.

`move(int dx, int dy)`

`move(int dx, int dy, int dz)`

Overriding:

Ridefinizione di una funzione in una sottoclasse (mantenendo immutata la firma)

Es. `estrai()` in Coda e Pila



Esempi

Persona – Studente - Docente

Veicolo – Auto - Moto



Modificatori: abstract

Classi dichiarate abstract non possono essere istanziate, e devono essere subclassate.

Metodi dichiarati abstract devono essere sovrascritti

Una class non abstract non può contenere abstract methods



Modificatori: final

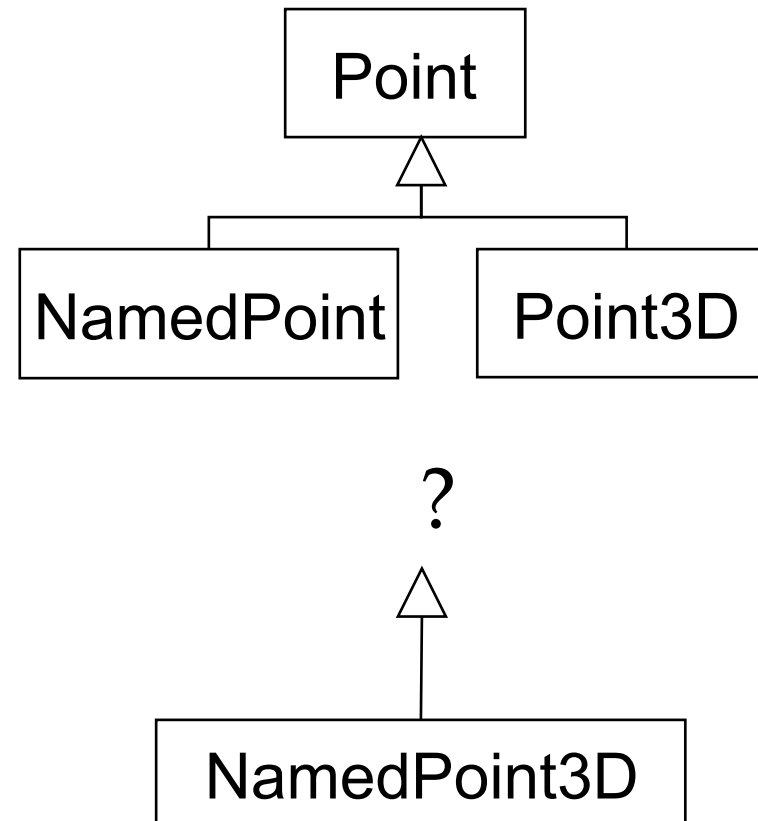
Variabili dichiarate final sono costanti.

**Metodi dichiarati final non possono essere
sovrascritti**

**Classi dichiarate final non possono essere
subclassate.**



Problemi con l'ereditarietà





Polimorphysm

- Una funzione può comportarsi in maniera diversa a seconda
- del tipo che le viene passato
 - del tipo di dato su cui è chiamata

```
Class A() {  
    do(int a) {System.out.println("1");}  
    do(String a) {System.out.println("2");}  
}  
Class B extends A {  
    do(int a) {System.out.println("3");}  
}
```

```
A a=new A();  
a.do(1);  
a.do("1");  
a=new B();  
a.do(1);
```



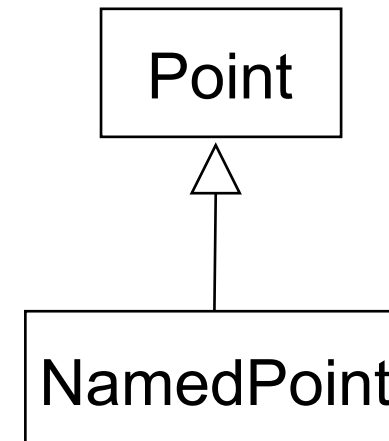
Liskov substitution principle

Se S è un sottotipo of T ,
allora oggetti di tipo T in un programma
possono essere sostituiti da oggetti di tipo S
senza alterare alcuna proprietà desiderabile del programma.

```
Point p=new Point();  
p.move(3,4);
```

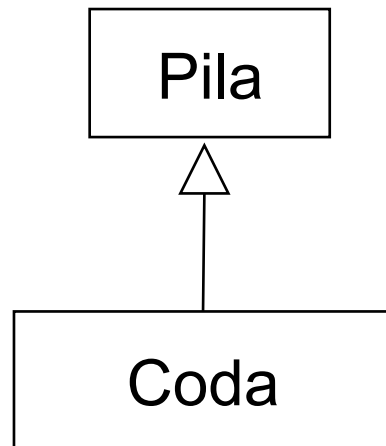
Ovunque c'e' un Point posso
mettere un NamedPoint

```
Point p=new NamedPoint();  
p.move(3,4);
```





Decisioni al volo...



```
public static void main(String a[]) {
    Pila p;
    // leggi k
    if (k==1) p=new Pila();
    else p=new Coda();
    p.inserisci(1);
    p.inserisci(2);
    p.estrai();
}
```

Il vero tipo della variabile p viene deciso A RUNTIME!
(dynamic binding).



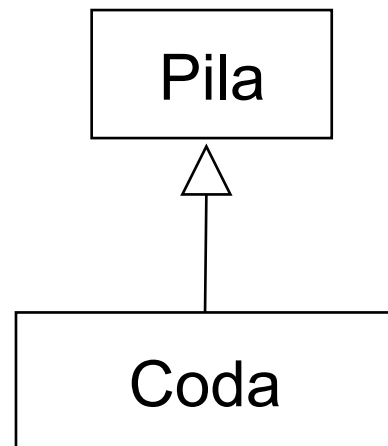
Decisioni al volo...

Il C++ offre al programmatore complessi meccanismi per decidere se usare **dynamic binding** (decisione del tipo a runtime) o **static binding** (decisione del tipo a compile time).

In Java le decisioni sono sempre fatte a runtime, salvo quando ci sono le condizioni per decidere automaticamente a compile time



Determinazione del tipo



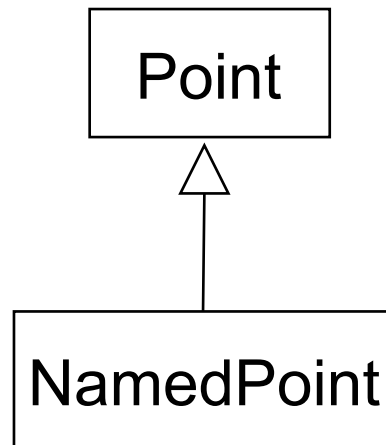
```
public static void main(String a[]){
    Coda p=new Coda();
    System.out.println(p instanceof Coda);
    System.out.println(p instanceof Pila);
}
```

true
true

Il vero tipo della variabile p viene deciso A RUNTIME!
(dynamic binding).



Determinazione del tipo



```
public static void main(String a[]){
    Point p;
    // leggi k
    if (k==1) p=new Point(2,2);
    else p=new NamedPoint(3,3,"A");

    // p.getName(); SBAGLIATO!

    if (p instanceof NamedPoint)
        ((NamedPoint)p).getName();
}
```



Static and Dynamic binding

Cosa succede quando si chiama un metodo su di un oggetto (esempio: `C obj; ... obj.f(args)`) ?

1. il compilatore cerca tra i metodi dell'oggetto `obj` i metodi `f(...)` e li enumera
2. il compilatore determina i tipi di parametri passati. Se trova un match unico con i metodi della classe `C` (anche tramite cast!) ne prende nota – **overloading resolution** – altrimenti genera un messaggio di errore
3. se il metodo è `private`, `static`, `final` o un costruttore il compilatore sa esattamente che metodo chiamare (**static binding**)
4. altrimenti il metodo dipende da qual'è la classe a cui `obj` appartiene a runtime ed il compilatore deve delegare alla JVM la determinazione del metodo a run time (**dynamic binding**)



Static and Dynamic binding

Se il programma usa il dynamic binding, la JVM deve chiamare il metodo appropriato al particolare tipo di oggetto `obj`.

- Es. `obj` sia di tipo `D`, classe derivata da `C`. Se esiste un metodo `f(int)` in `D`, questo sarà il metodo chiamato, altrimenti verrà chiamato il metodo `f(int)` di `C`
- Eseguire ogni volta questi controlli non è efficiente., per cui la virtual machine calcola in anticipo un `method table` per ogni classe che raccoglie tutte le `signatures` dei metodi



Polimorfismo – esempio

```
abstract class OP {  
    int f(int a,int b);  
}  
  
class Somma extends OP {  
    int f(int a,int b){  
        return a+b;  
    }  
}  
  
class Sottrazione extends OP {  
    int f(int a,int b){  
        return a-b;  
    }  
}
```



Polimorfismo - esempio

```
class Test {  
    public static void main(String[] a) {  
        new Test;  
    }  
    public Test() {  
        OP o;  
        ...  
        if (i!=0) o=new Somma();  
        else o=new Sottrazione();  
        ...  
        o.f(3,2);  
    }  
}
```

Che metodo viene chiamato qui?