UNIVERSITÀ PARTHENOPE
DiST
Dipartimento di Scienze e Tecnologie

Programmazione 3 e Laboratorio di Programmazione 3

Polimorfismo

Proff. Angelo Ciaramella – Emanuel Di Nardo

Introduzione

- Polimorfismo
  - dal greco “molte forme”
  - consente di riferirci con un unico termine a “entità” diverse
  - e.g., sia un telefono fisso sia un portatile permettono di telefonare
    - telefonare può essere considerata un’azione polimorfica

Polimorfismo

Polimorfismo

Polimorfismo per metodi

Polimorfismo per dati

Overload

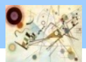
Override

Collezioni eterogenee

Parametri Polimorfi

Metodi virtuali

Polimorfismo in Java

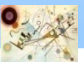


Reference

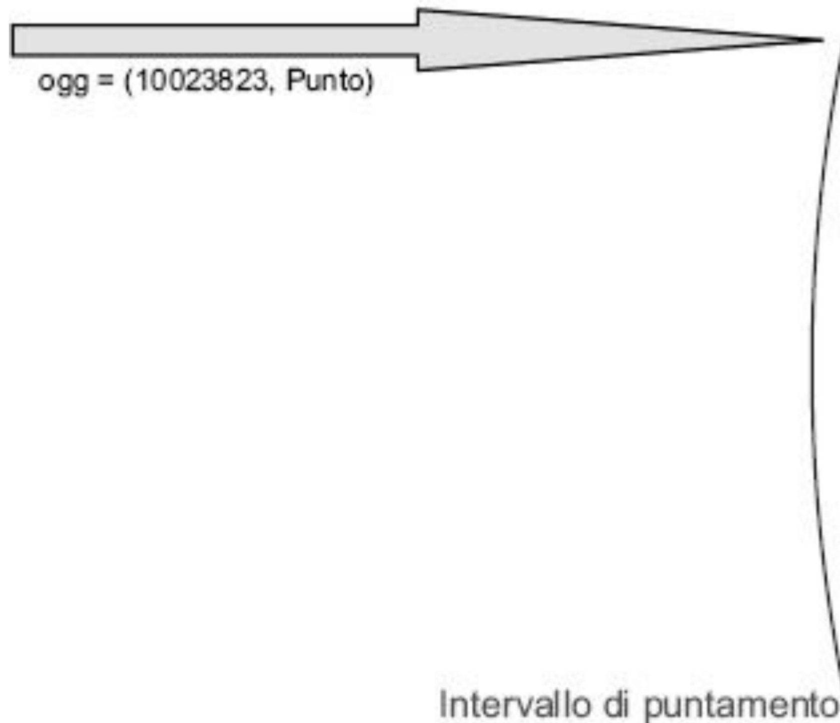
```
public class Punto {
private int x;
private int y;
public void setX(int x) {
this.x = x;
}
public void setY(int y) {
this.y = y;
}
public int getX() {
return x;
}
public int getY() {
return y;
}
}
```

```
Punto ogg = new Punto();
```

ogg è il reference

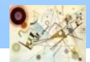


Reference



INTERFACCIA PUBBLICA	IMPLEMENTAZIONE INTERNA
setX()	
getX()	x
setY()	
getY()	y

Convenzione per i reference



Polimorfismo per metodi

- Realizzazione pratica sotto due forme
  - overload
    - che potremmo tradurre con "sovraccarico"
  - override
    - che potremmo tradurre con "riscrittura"

Overload

- Metodo
  - la coppia costituita dall'identificatore e dalla lista dei parametri è detta firma o segnatura
    - dall'inglese *signature*
  - in una classe possono convivere metodi con lo stesso nome ma con differente firma
  - Il tipo di ritorno non fa parte della firma di un metodo
    - *non ha importanza* per l'implementazione dell'overload

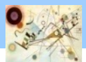
Overload

```
public class Aritmetica {
public int somma(int a, int b) {
return a + b;
}
public float somma(int a, float b) {
return a + b;
}
public float somma(float a, int b) {
return a + b;
}
public int somma(int a, int b, int c) {
return a + b + c;
}
public double somma(int a, double b, int c) {
return a + b + c;
}
}
```

tipale

posizionale

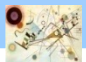
numerico



Varargs

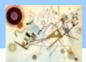
```
public class Aritmetica {
public double somma(double... doubles) {
double risultato = 0.0D;
for (double tmp : doubles) {
risultato += tmp;
}
return risultato;
}
}
```

Vargas risolve alcuni problemi di overload



Override

- Override
  - le sottoclassi possono ridefinire un metodo ereditato da una superclasse
  - il metodo riscritto nella sottoclasse deve avere la stessa firma del metodo della superclasse
  - il tipo di ritorno del metodo della sottoclasse deve coincidere con quello del metodo che si sta riscrivendo o deve essere di un tipo che estende il tipo di ritorno del metodo della superclasse
  - Il metodo ridefinito nella sottoclasse non deve essere meno accessibile del metodo originale della superclasse
    - e.g., metodo ereditato è dichiarato protetto, non si può ridefinire privato, semmai pubblico

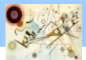


Override

```
public class Punto {
private int x, y;
public void setX(int x) {
this.x = x;
}

public int getX() {
return x;
}
public void setY(int y) {
this.y = y;
}
public int getY() {
return y;
}
public double distanzaDallOrigine() {
int tmp = (x*x) + (y*y);
return Math.sqrt(tmp);
}
}
```

Override

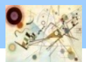
```
public class PuntoTridimensionale extends Punto {
private int z;
public void setZ(int z) {
this.z = z;
}
public int getZ() {
return z;
}

public double distanzaDallOrigine() {
int tmp = (getX()*getX()) + (getY()*getY())
+ (z*z); // N.B. : x ed y non sono ereditate
return Math.sqrt(tmp);
}
}
```

Annotazione

```
public class PuntoTridimensionale {
@Override                    //annotazione
public double distanzadallOrigine() {
. . .
}
}
```

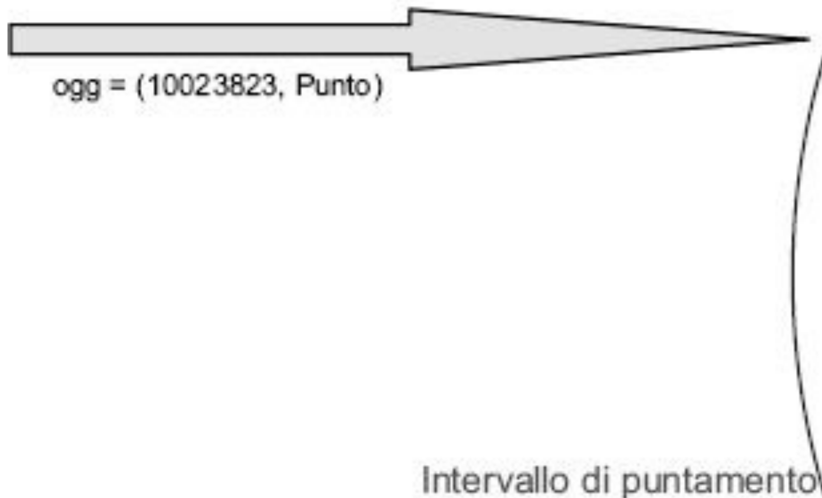
```
method does not override a method from its superclass
@Override public double distanzadallOrigine() {
^
1 error
```

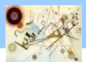
Polimorfismo per dati

- Permette di poter assegnare un reference di una superclasse ad un'istanza di una sottoclasse

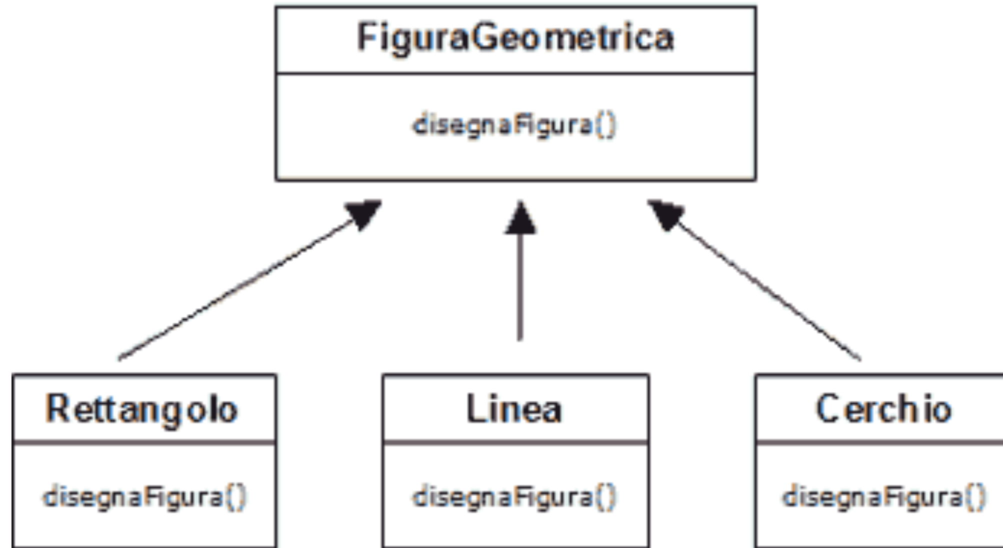
```
Punto ogg = new PuntoTridimensionale();
```



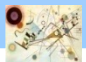
INTERFACCIA PUBBLICA	IMPLEMENTAZIONE INTERNA
setX()	
getX()	x
setY()	
getY()	
distanza...()	y
setZ()	
getZ()	



Polimorfismo per dati



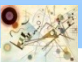
Esempio di ereditarietà



Polimorfismo per dati

- Il sistema è in grado di capire autonomamente quale figura geometrica debba essere disegnata ed invocarne direttamente il metodo `disegnaFigura` appartenente alla classe figlia coinvolta

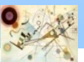
- In un sistema non ad oggetti
  - costrutto tipo `switch`



Polimorfismo per dati

```
class Albero
{
 protected void cresce()
   {
   System.out.println("Metodo cresce della classe Albero");
   }
   public static void main(String args[])
    {
        Albero al = new Abete();
        al.cresce();
    }
}
class Abete extends Albero
{
    public void cresce()
    {
        System.out.println("Metodo cresce della classe Abete");
    }
}
```

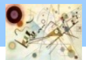
```
Metodo cresce della classe Abete
```

Parametri polimorfi

- In un metodo un parametro di tipo *reference* si dice parametro polimorfo
  - può puntare ad un oggetto istanziato da una sottoclasse
  - metodo `println()` che prende un parametro di tipo `Object`

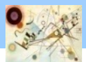
```
Punto p1 = new Punto();

System.out.println(p1);
```

Collezioni eterogenee

- Una collezione eterogenea è una collezione composta da oggetti diversi
  - e.g., un array di `Object` che in realtà immagazzina oggetti diversi
  - anche la possibilità di sfruttare collezioni eterogenee è garantita dal polimorfismo per dati

```
Object arr[] = {new Punto(), "Hello World!", new
Date()};
```

instanceof

- instanceof
  - ci permette di testare a quale *tipo di istanza* punta un reference
  - *true* se il primo operando è un reference che punta ad un oggetto istanziato dal secondo operando o ad un oggetto istanziato da una sottoclasse

```
public class Dipendente {
...
};
```

```
public class Programmatore extends Dipendente {
...
}
```

```
public void pagaDipendente(Dipendente dip) {
if (dip instanceof Programmatore) {
...
}
```

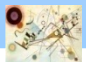
Casting di oggetti

- il reference che punta ad un oggetto istanziato da una sottoclasse non può accedere ai membri dichiarati nelle sottoclassi stesse
  - ristabilire la piena accessibilità all'oggetto tramite il meccanismo del casting di oggetti

```
if (dip instanceof Programmatore) {
Programmatore pro = (Programmatore) dip;
. . .
```

reference a `Programmatore` che punta all'indirizzo di memoria dove punta il reference `dip`

- assegnare al reference `pro` l'indirizzo di `dip` senza l'utilizzo del casting
  - errore in compilazione ed un relativo messaggio che ci richiede un casting esplicito



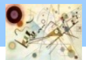
Casting di oggetti

dip = (10023823, Dipendente)

pro = (10023823, Programmatore)

nome
stipendio
matricola
dataDiNascita
dataDiAssunzione
linguaggiConosciuti
anniDiEsperienza

Due tipi di accessi allo stesso oggetto



Invocazione virtuale dei metodi

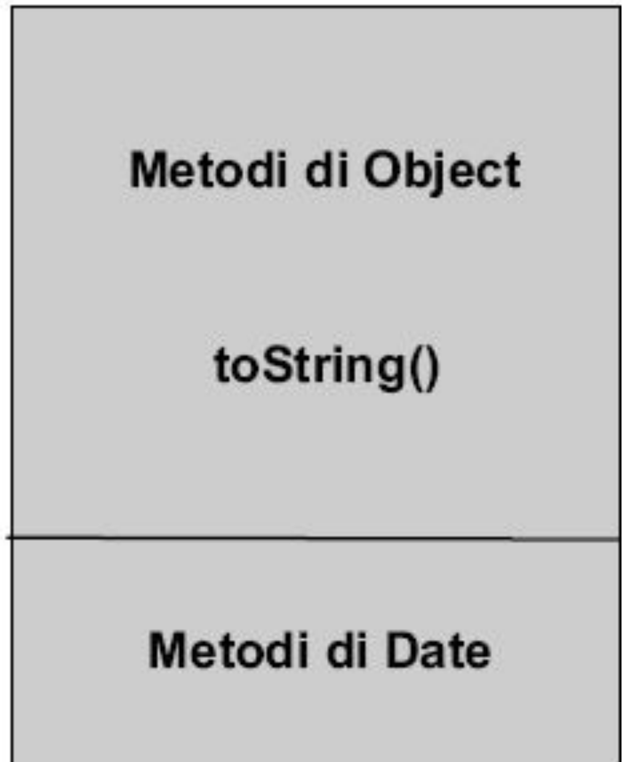
- Invocazione ad un metodo `m` può definirsi virtuale
  - `m` è definito in una classe A
  - ridefinito in una sottoclasse B (*override*)
  - invocato su un'istanza di B tramite un reference di A (*polimorfismo per dati*)

- Viene invocato il metodo ridefinito nella classe B

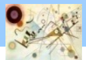
Invocazione virtuale dei metodi

```
// classe Date del package java.util
. . .
Object obj = new Date();
String s1 = obj.toString();
. . .
```

reference a Date

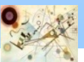


Lo spazio dei metodi di un oggetto Date che sovrascrive `toString()` della classe Object



Esempio di polimorfismo

```java
public abstract class Veicolo {
public abstract void accelera();
public abstract void decelera();
}
public class Aereo extends Veicolo {
public void decolla() {
//. . .
}
public void atterra() {
//. . .
}

@Override
public void accelera() {
//. . .
}

@Override
public void decelera() {
//. . .
}
//. . .
```

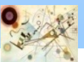



Esempio di polimorfismo

```
}
public class Automobile extends Veicolo {
@Override
public void accelera() {
//. . .
}
@Override
public void decelera() {
//. . .
}
public void innestaRetromarcia() {
//. . .
}
//. . .
}
public class Nave extends Veicolo {
@Override
public void accelera() {
//. . .
}
@Override
public void decelera() {
```

Esempio di polimorfismo

```
/. . .
}
public void gettaAncora() {
//. . .
}
//. . .
}

public class Viaggiatore {
public void viaggia(Automobile a) {
a.accelera();

//. . .
}
public void viaggia(Aereo a) {
a.accelera();
//. . .
}
public void viaggia(Nave n) {
n.accelera();
//. . .
}
//. . .
}
```

Esempio di polimorfismo

```
public class Viaggiatore {
public void viaggia(Veicolo v) { //param. Polimorfo
v.accelera(); //invocazione metodo virtuale
//. . .
}
//. . .
}

// Esempio 1
Viaggiatore claudio = new Viaggiatore();
Automobile fiat500 = new Automobile();
// avremmo potuto istanziare anche una Nave o un Aereo
claudio.viaggia(fiat500);

//Esempio 2
Viaggiatore claudio = new Viaggiatore();
Aereo piper = new Aereo();
claudio.viaggia(piper);
```

Sfruttando un parametro polimorfo ed un metodo virtuale

Polimorfismo e interfacce

```
public interface Volante {
void atterra();
void decolla();
}

public class Aereo extends Veicolo implements Volante {
@Override
public void atterra() {
// override del metodo di Volante
}
@Override
public void decolla() {
// override del metodo di Volante
}
@Override
public void accelera() {
// override del metodo di Veicolo
}
@Override
public void decelera() {
// override del metodo di Veicolo
}
}
```

Polimorfismo e interfacce

```
public class TorreDiControllo {

//parametri polimorfi per sfruttare l'interfaccia Volante

public void autorizzaAtterraggio(Volante v) {
v.atterra();
}
public void autorizzaDecollo(Volante v) {
v.decolla();
}
}



// possiamo passare a questi metodi degli "oggetti
// volanti" creati da classi che implementano
// l'interfaccia Volante
```