LEZIONE 10
CLASS DIAGRAMS 2.2: Polimorfismo

Ingegneria del Software e Progettazione Web

Università degli Studi di Tor Vergata - Roma

Guglielmo De Angelis
guglielmo.deangelis@isti.cnr.it

operazione, metodo, messaggio

- operazione : specifica di un “*segnatura*” (prototipo) che un oggetto mette a disposizione di altri oggetti

- metodo : implementazione vera e propria dell'operazione di un oggetto

- messaggio : è la richiesta a run-time di invocazione di un metodo fornito da un oggetto $\mathcal{B}$ da parte di un oggetto $\mathcal{A}$

...

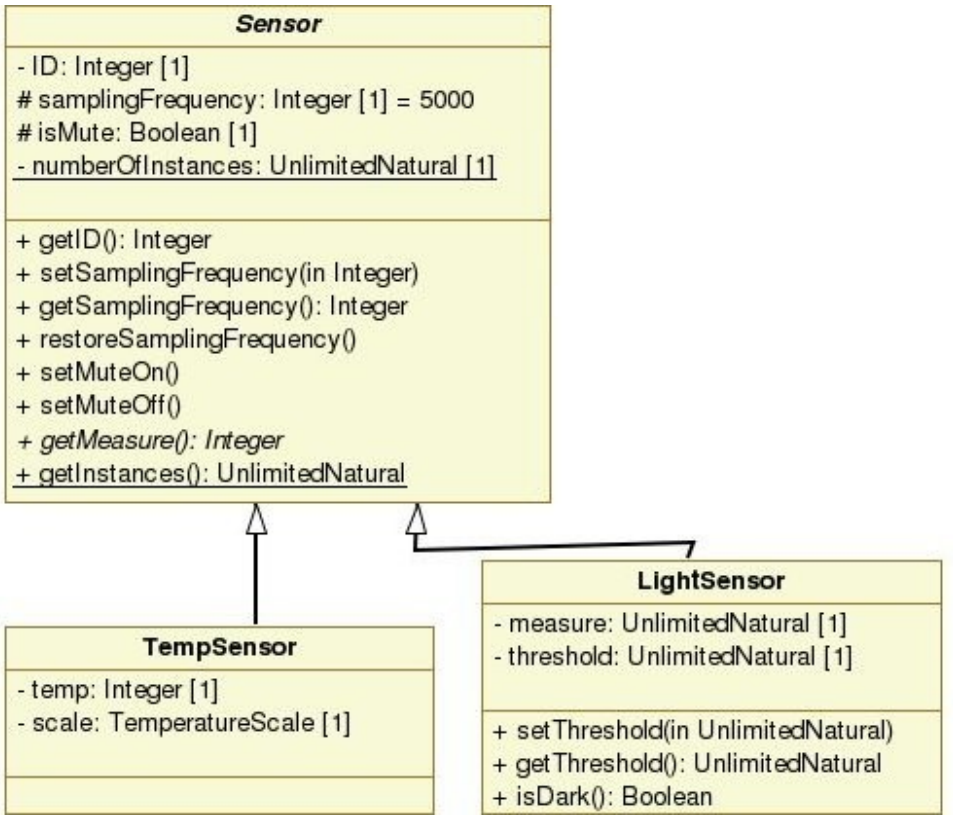
- *problema* :
  - modellare una famiglia di sensori. In particolare sensori di temperatura e di luminosità

generalizzazione e ...

- *problema* :
  - modellare una famiglia di sensori. In particolare sensori di temperatura e di luminosità
- *soluzione* :
  - definizione delle entità → classi
  - definizione delle caratteristiche comuni → operazioni ed attributi

generalizzazione e classi ...

- *problema* :
  - modellare una famiglia di sensori. In particolare sensori di temperatura e di luminosità
- *soluzione* :
  - definizione delle entità → classi
  - definizione delle caratteristiche comuni → operazioni ed attributi

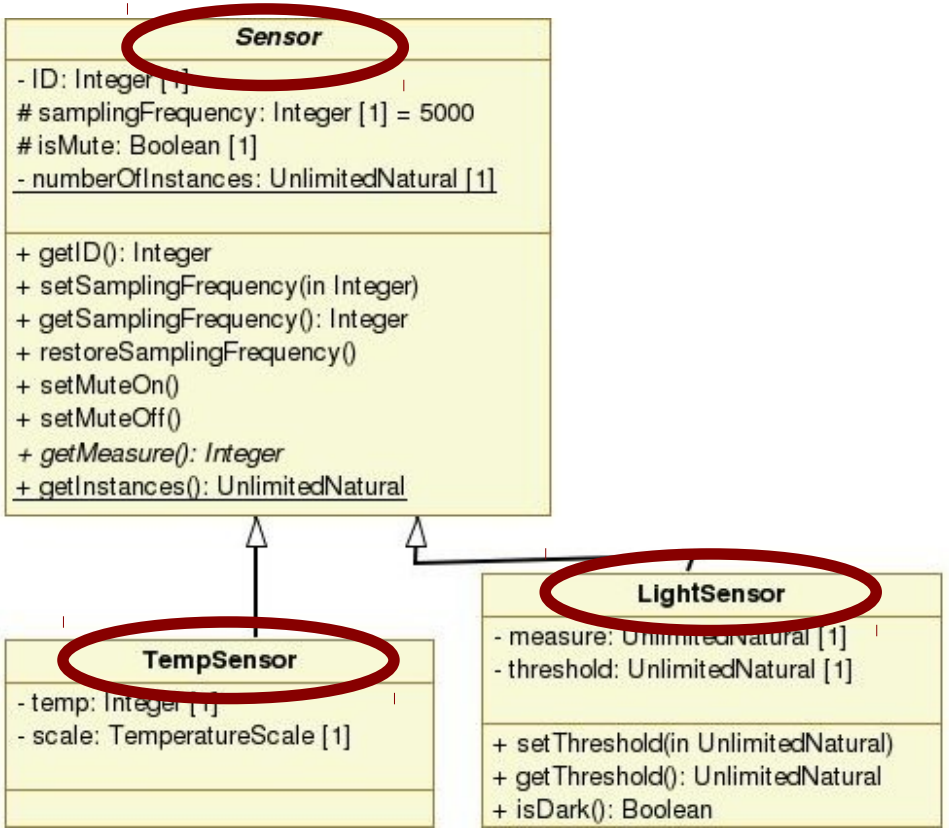


generalizzazione e classi …

(map in Java)

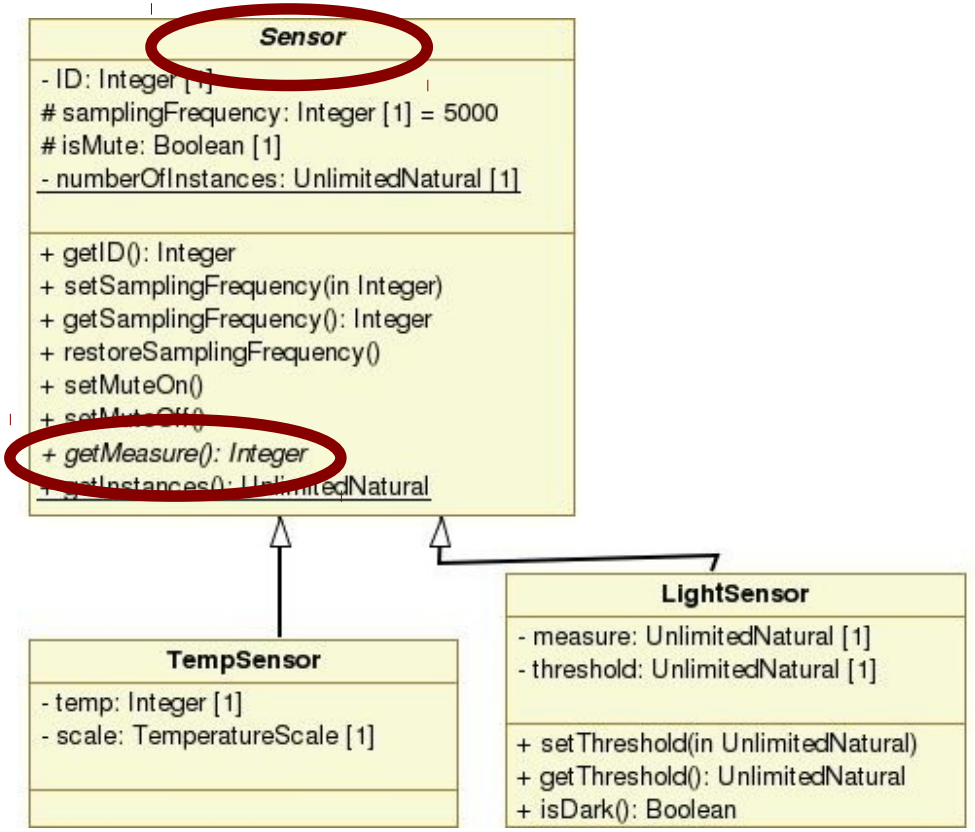
```
// Errore
Sensor s = new Sensor();
// OK
Sensor t = new TempSensor();
// OK
Sensor l = new LightSensor();
// OK
l.setMuteOn();
l.getThreshold();// ERRORE

l.getMeasure (); // OK
t.getMeasure (); // OK
```



generalizzazione e classi … astratte

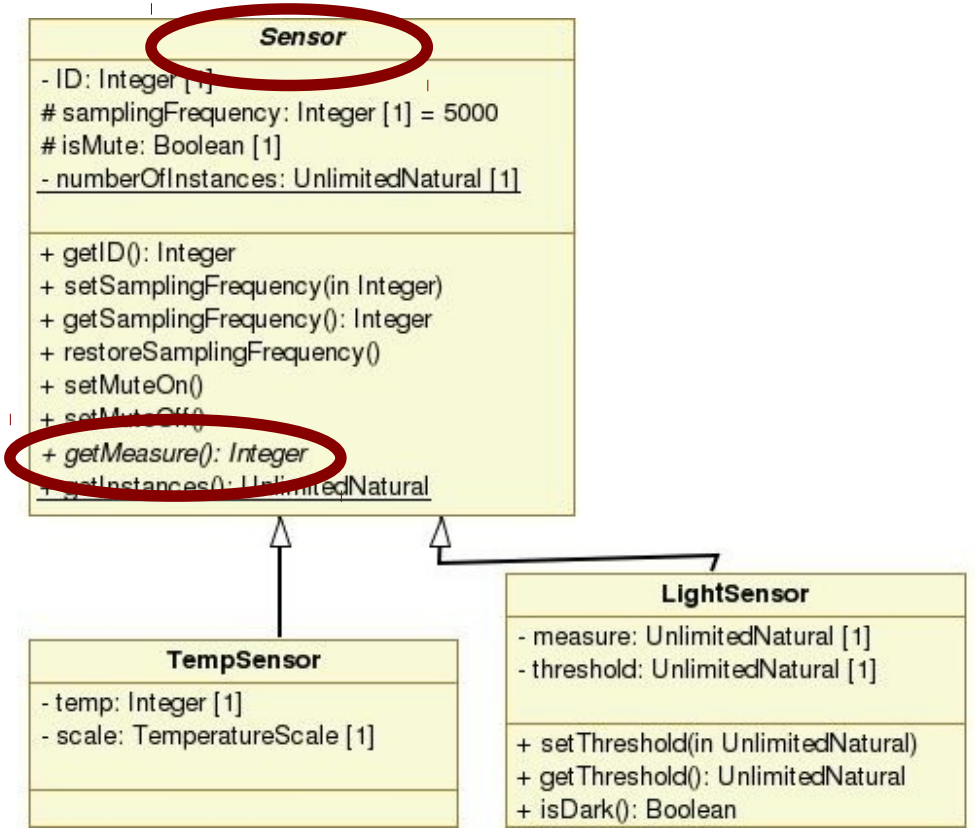
- *nuovo problema* :
  - alcune delle caratteristiche comuni che vengono raggruppate nella superclasse devono manifestarsi in modo diverso dipendentemente della sottoclasse considerata



- nello specifico :
  - il modo in cui avviene rilevamento di una temperatura presumibilmente sarà diverso dal modo in cui avviene il rilevamento di una intensità luminosa

generalizzazione e classi … astratte (map in Java)

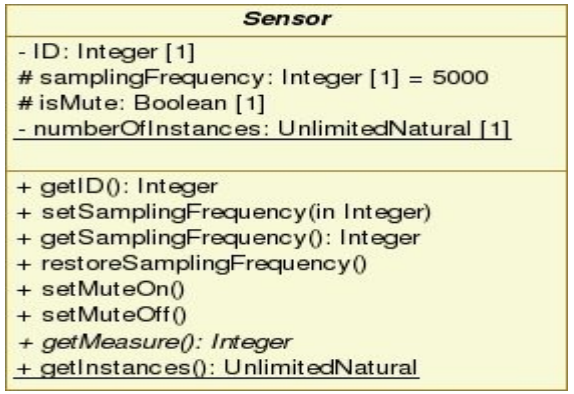
```
// Errore
Sensor s = new Sensor();
// OK
Sensor t = new TempSensor();
// OK
Sensor l = new LightSensor();
// OK
l.setMuteOn();
l.getThreshold();// ERRORE
```

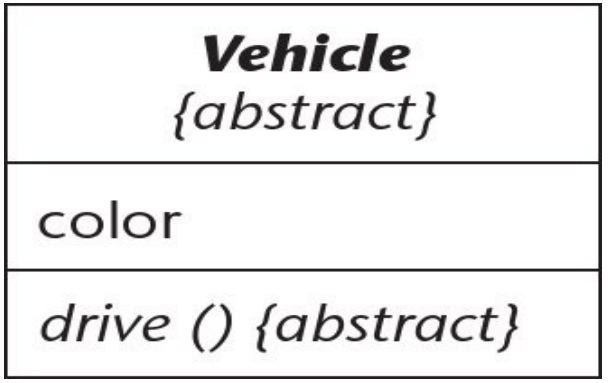


`l.getMeasure ();` ATTENZIONE: anche se `l` e `t` sono dello
`t.getMeasure ();` stesso tipo, le chiamate si riferiscono a implementazioni diverse

classi astratte

- sono classi parzialmente definite
- sono utili per modellare contesti in cui un insieme di classi include operazioni con la stessa semantica ma che saranno implementate da metodi differenti
- in Java si usa il modificatore `abstract` per classi ed operazioni
- in UML non c'è una simbologia grafica univoca

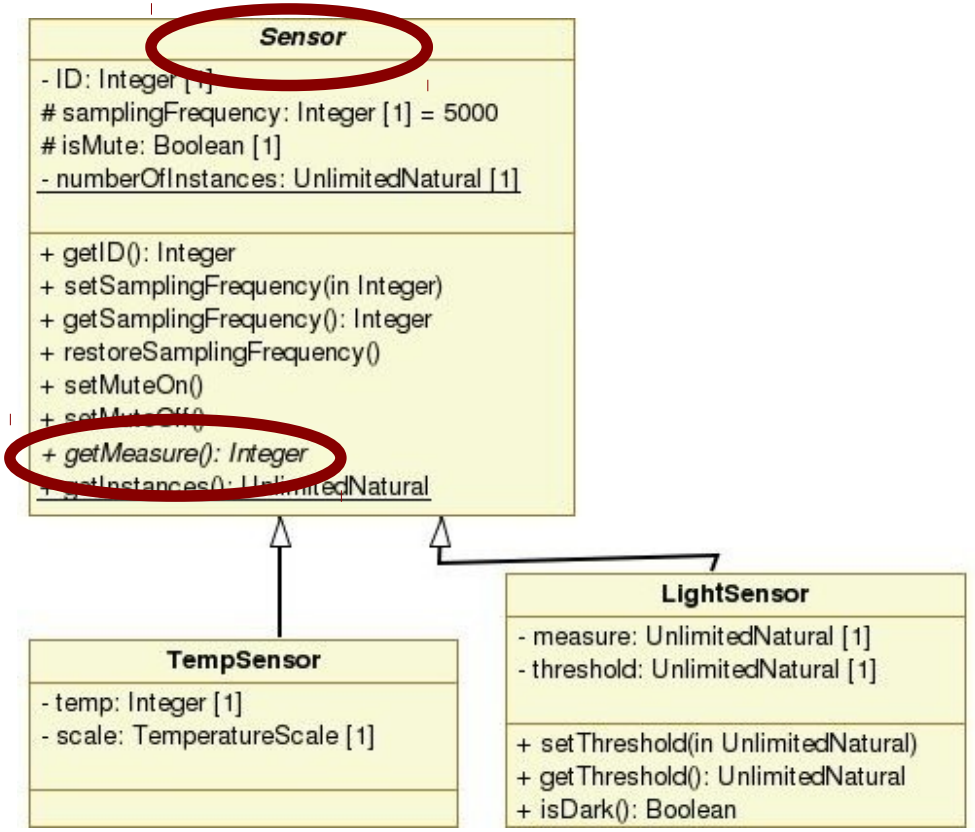




va per la maggiore!!

generalizzazione e classi … astratte (map in Java)

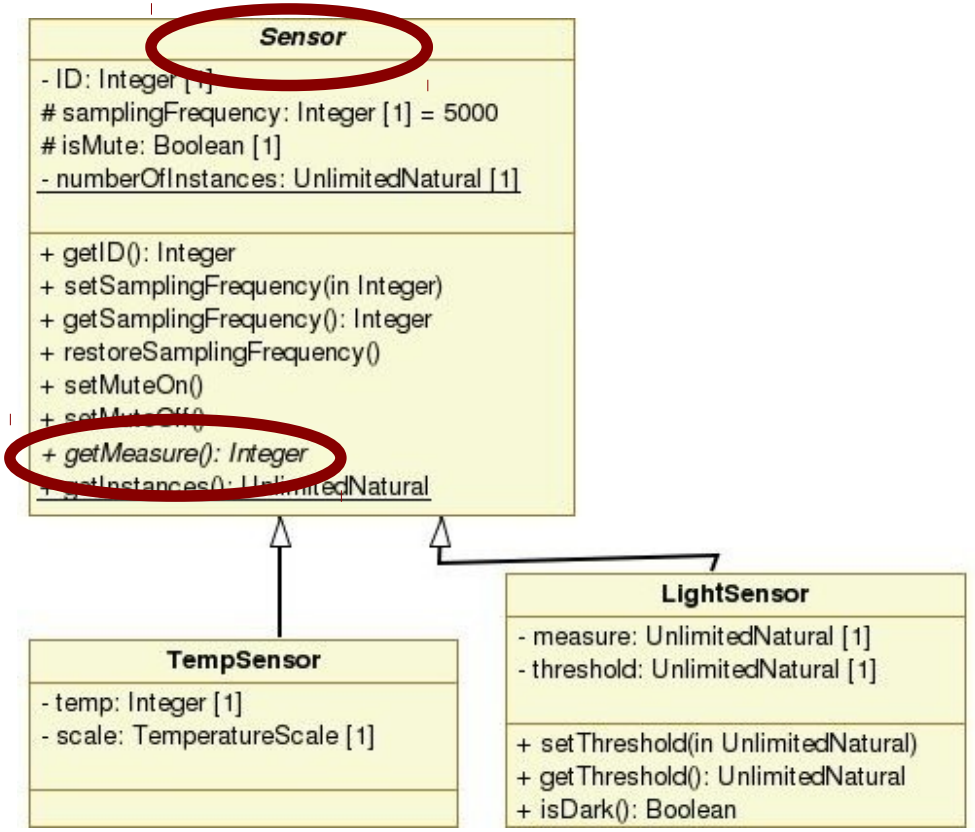
```
// Errore
Sensor s = new Sensor();
// OK
Sensor t = new TempSensor();
// OK
Sensor l = new LightSensor();
// OK
l.setMuteOn();
l.getThreshold();// ERRORE
```



`l.getMeasure ();` ATTENZIONE: anche se `l` e `t` sono dello
`t.getMeasure ();` stesso tipo, le chiamate si riferiscono a implementazioni diverse

generalizzazione, classi astratte e … polimorfismo

- non è mai possibile istanziare una classe astratta in una variabile
- normalmente le variabili su classi astratte si usano per mantenere riferimenti ad istanze (concrete) di sottoclassi
- **polimorfismo** : capacità di una classe di "comportarsi" in modi differenti

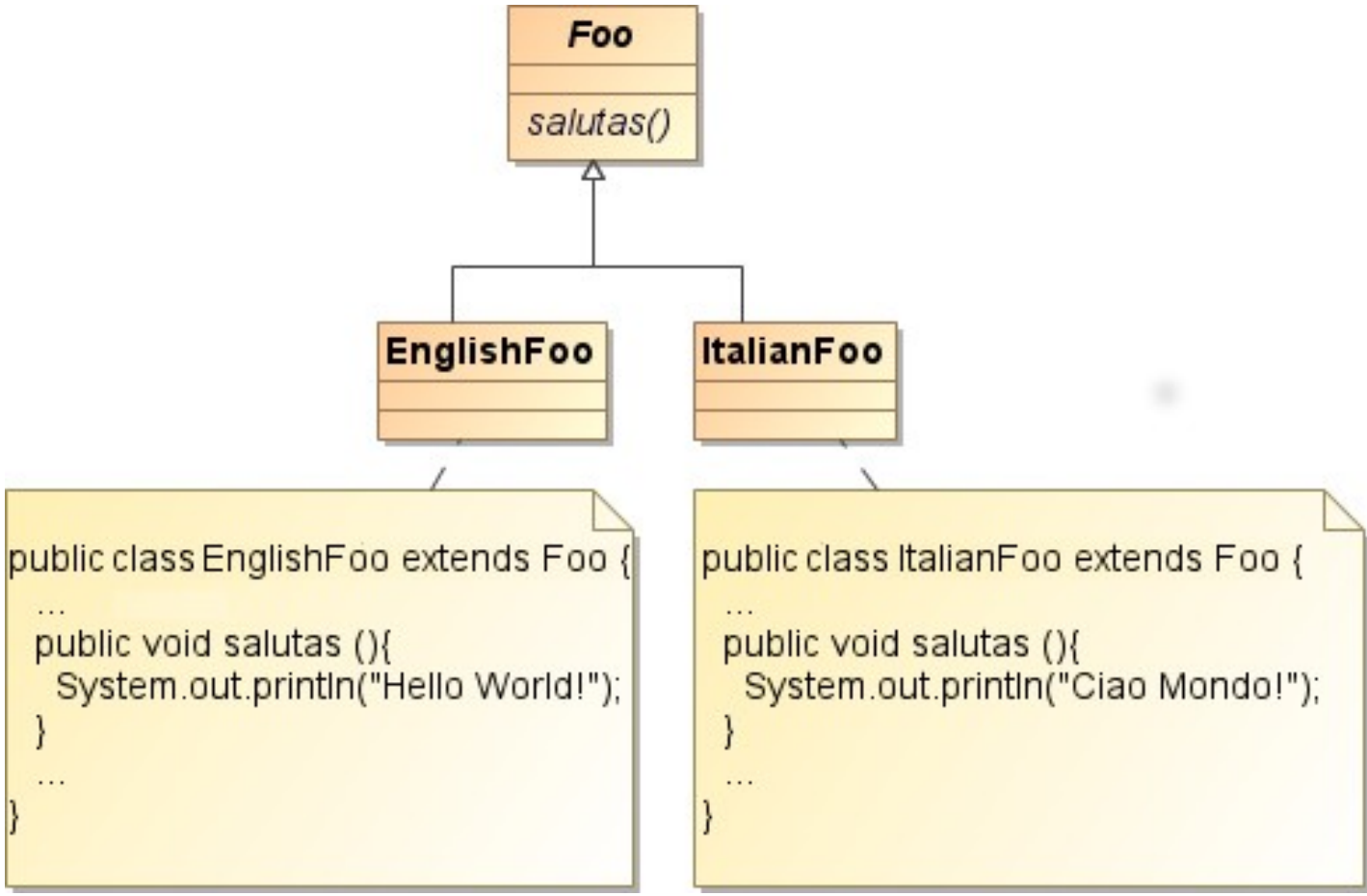


1. gerarchia di classi
2. variabile (e.g. **`s`**) con tipo la superclasse astratta
3. meccanismo di scelta che crea una istanza per una sottoclasse e ne associa il riferimento ad **`s`**
4. linguaggio con supporto di binding dinamico (vedi lezione 2 e lezione 15-18)

polimorfismo

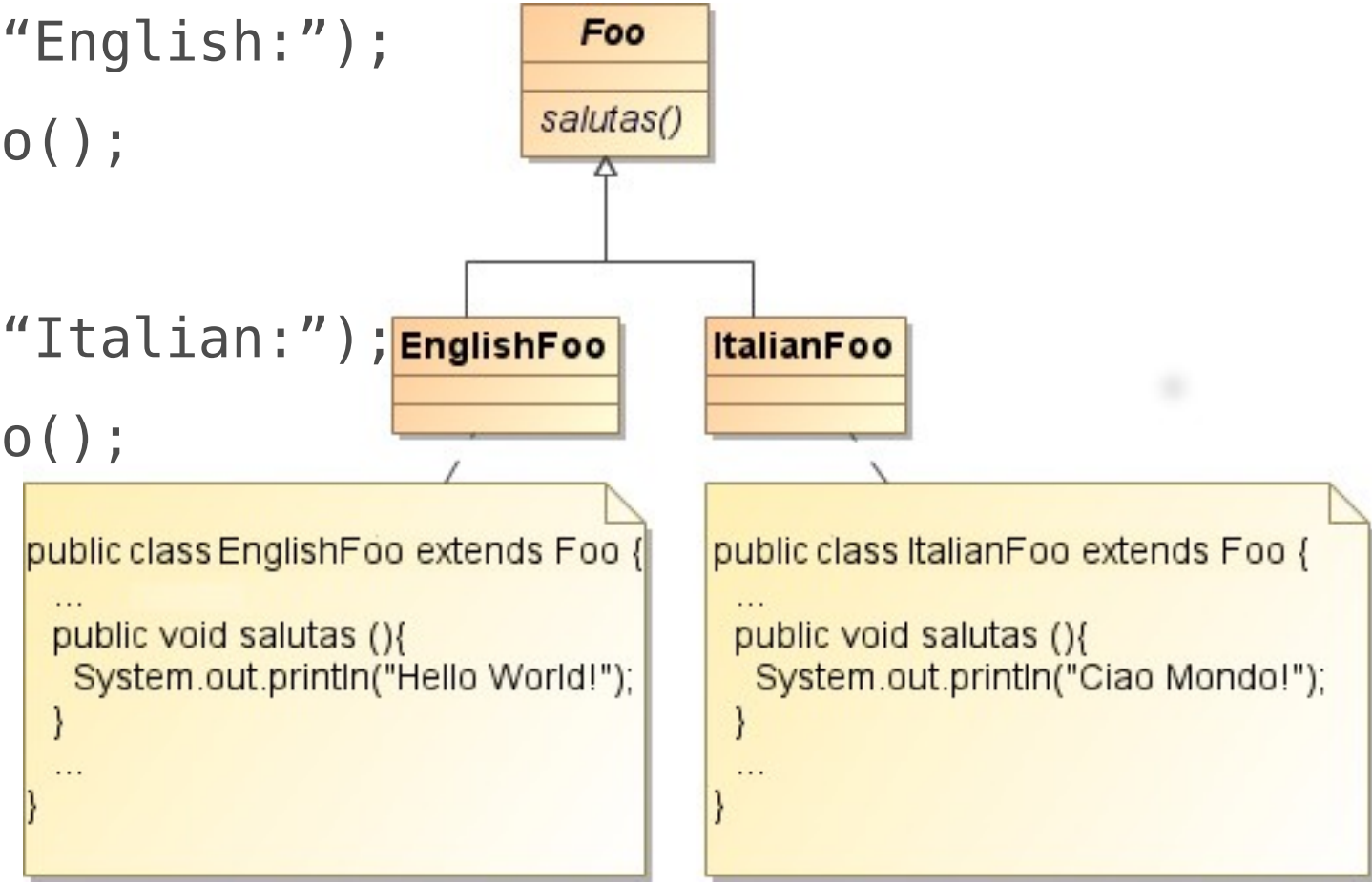
- consente di progettare/realizzare sistemi che “perfezionano” il loro comportamento solo a tempo di esecuzione
- i sistemi polimorfici sono:
  - molto flessibili
  - facili da estendere : basta aggiungere nuove sottoclassi e combinarle con nuovi meccanismi di scelta
- il polimorfismo consente alle istanze di classi differenti di rispondere allo stesso messaggio in modi differenti
  - cioè consente che all'invocazione di stesse operazioni, riferisca a metodi diversi

generalizzazione e polimorfismo



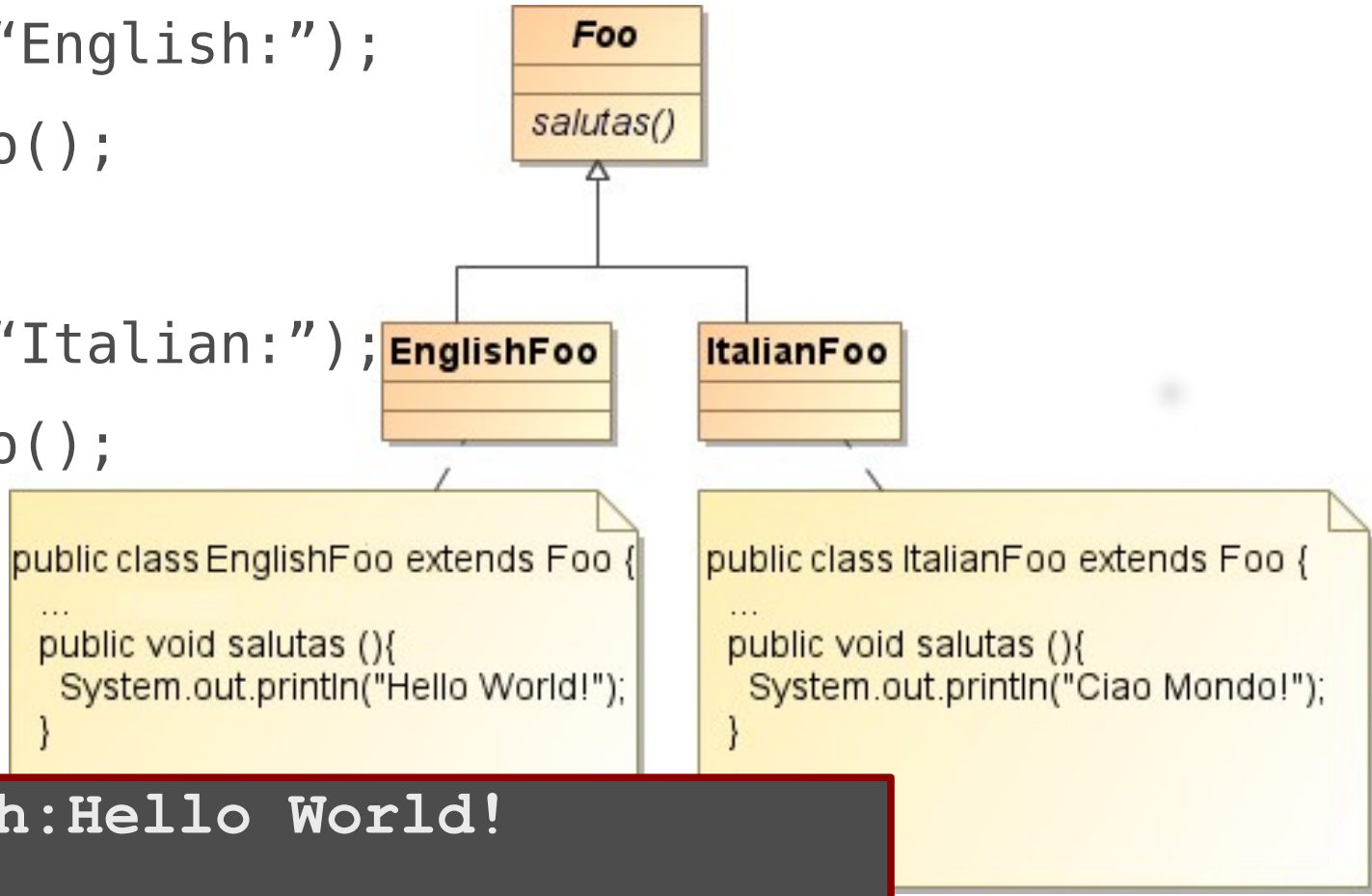
generalizzazione e polimorfismo

```
Foo f;
if (<test>){
  System.out.print(“English:”);
  f = new EnglishFoo();
} else {
  System.out.print(“Italian:”);
  f = new ItalianFoo();
}
f.salutas();
```



generalizzazione e polimorfismo

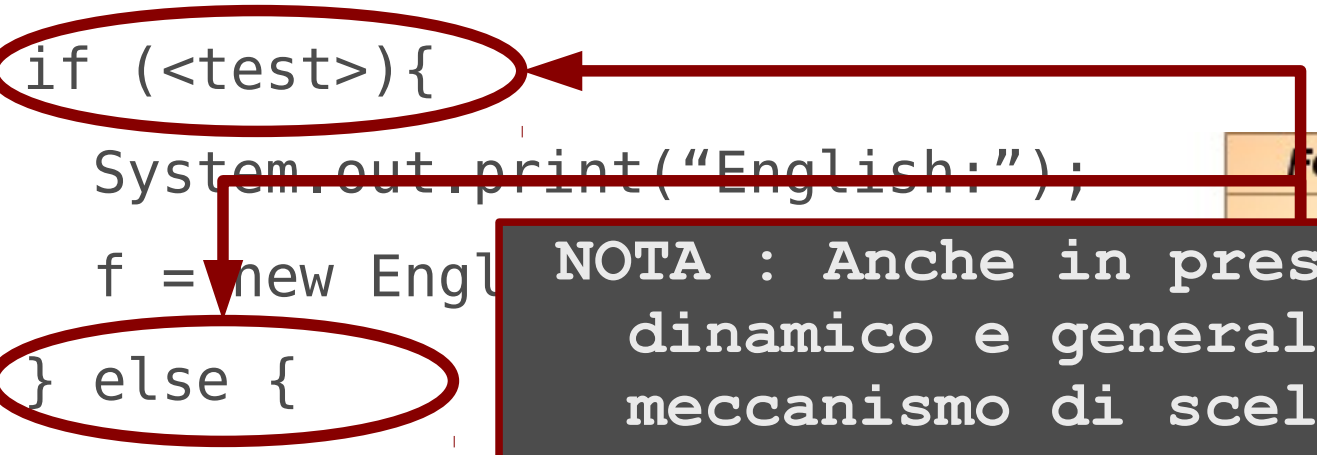
```
Foo f;
if (<test>){
  System.out.print(“English:”);
  f = new EnglishFoo();
} else {
  System.out.print(“Italian:”);
  f = new ItalianFoo();
}
f.salutas();
```




```
<test>   : English:Hello World!
!<test>  : Italian:Ciao Mondo!
```

generalizzazione e polimorfismo

```
Foo f;
if (<test>){
   System.out.print(“English:”);
   f = new Engl
} else {
   System.out.p
   f = new Ital
}
f.salutas();
```



NOTA : Anche in presenza di binding dinamico e generalizzazione, senza un meccanismo di scelta, questo codice avrebbe ha sempre lo stesso comportamento … quindi non si tratterebbe di una soluzione POLIMORFA

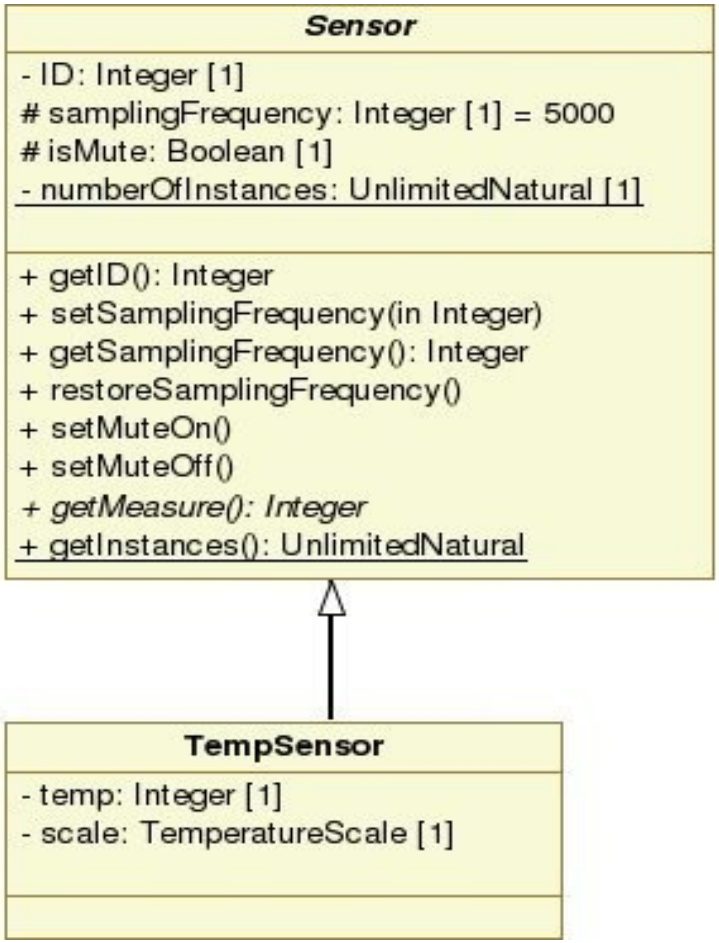
```
public class EnglishFoo extends Foo {
  …
  public void salutas (){
    System.out.println("Hello World!");
  }
```

```
public class ItalianFoo extends Foo {
  …
  public void salutas (){
    System.out.println("Ciao Mondo!");
  }
```

<test> : English:Hello World!

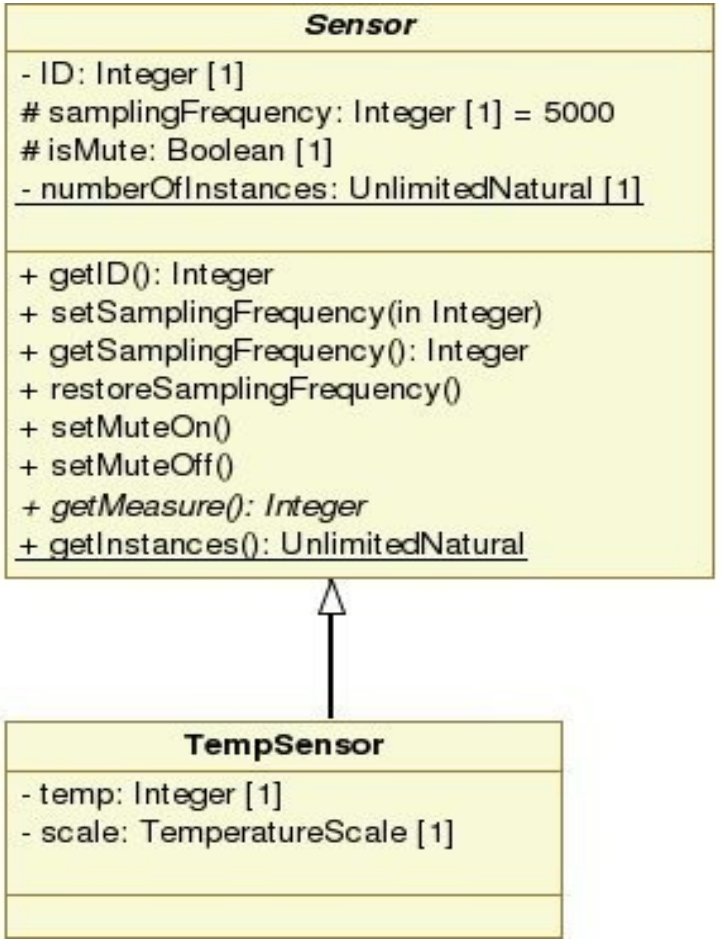
!<test> : Italian:Ciao Mondo!

generalizzazione – visibilità (map in Java)



generalizzazione – visibilità (map in Java)

```
public class TempSensor extends Sensor {
   public void setMuteOn(){
      int k = this.ID; // ERRORE
      this.isMute = true;// OK
   }
}
public class TestClass {
   public void testMethod(){
     Sensor s = new TempSensor();
     s.isMute = true ; // ERRORE
   }
}
```



SOLO le sottoclassi POSSONO riferire gli elementi dichiarati `protected` nella superclasse.

overriding VS overloading

overriding VS overloading

- overriding : sovrascrivo il comportamento di una operazione cambiando il metodo a cui essa è associata
  - precondizioni a overriding tra `A` e `B` su `m`
    - gerarchia di classi (e.g. `A` generalizza `B`)
    - stessa segnatura di `A.m` e `B.m`
    - `A.m` è `public` o `protected`, oppure `B.m` sovrascrive un altro metodo che comunque sovrascrive `A.m`

overriding VS overloading

- overriding : sovrascrivo il comportamento di una operazione cambiando il metodo a cui essa è associata
  - precondizioni a overriding tra `A` e `B` su `m`
    - gerarchia di classi (e.g. `A` generalizza `B`)
    - stessa segnatura di `A.m` e `B.m`
    - `A.m` è `public` o `protected`, oppure `B.m` sovrascrive un altro metodo che comunque sovrascrive `A.m`
- overloading : aggiunge funzionalità ad una classe fornendole una nuova operazione
  - due metodi (definiti localmente o ereditati) in una classe hanno lo stesso nome ma non la stessa segnatura

overriding VS overloading

```
package it.uniroma2.dicii.myfirsttest;

public class A {

  public int test (int p){
      return p;
  }

}
```

overriding VS overloading

```java
package it.uniroma2.dicii.myfirsttest;

public class A {

   public int test (int p){
        return p;
   }

}
```

```java
package it.uniroma2.dicii.myfirsttest;

public class B extends A {

    @Override
    public int test (int p){
        return p+1;
    }

    public int test (float p){
        return super.test(Math.round(p));
    }
}
```