LEZIONE 1 - CLASS DIAGRAM

Catia Trubiani

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica (DISIM) - Università degli Studi dell'Aquila

catia.trubiani@univaq.it

Class Diagram

Il Class Diagram viene usato per modellare la struttura statica di un sistema ed è costituito da:

entità + relazioni

Una **classe** descrive una **entità** da modellare, ed è caratterizzata:

(1) da un nome

(2) degli attributi

(3) delle operazioni sugli attributi

Classe

Modella una famiglia di entità del dominio di applicazione

le proprietà (**attributi**)

il comportamento (**operazioni**)

Name
Attributes
Operations

Raggruppa un insieme coeso di entità

Alcuni esempi:

mammifero, autoveicolo, grafo...

Un esempio di classe: 'car' (1/2)

Car
registration number data speed direction

un esempio di classe
e i suoi attributi

un esempio di entità concrete (ISTANZE) appartenenti alla stessa classe

Un esempio di classe: 'car' (2/2)

Car
registration number : String data : CarData speed : Integer direction : Direction

attributi di una classe e tipo di dato associato agli attributi

Car
registration number : String data : CarData speed : Integer direction : Direction
drive (speed : Integer, direction : Direction) getData () : CarData

un esempio di classe e le sue operazioni

Altri esempi di classe

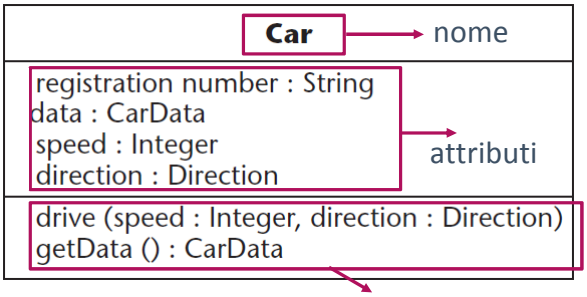
Invoice
amount : Real date : Date = Current date customer : String specification : String administrator : String = "Unspecified" number of invoices : Integer

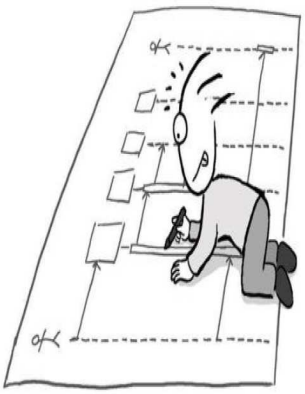
attributo di classe: il valore di questo attributo è uguale per tutte le istanze perché l'attributo è condiviso.

Figure
size : Size pos : Position figcounter : Integer
draw () getCounter () : Integer

operazione di classe: il valore restituito dall'operazione è uguale per tutte le istanze perché l'operazione è condivisa.

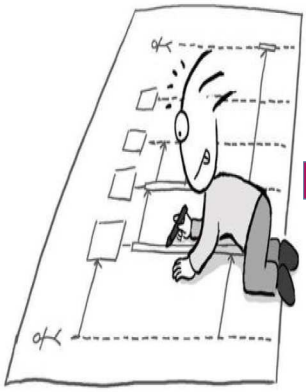
Cosa rappresenta un class diagram?! È un 'disegno'?! A cosa serve?!



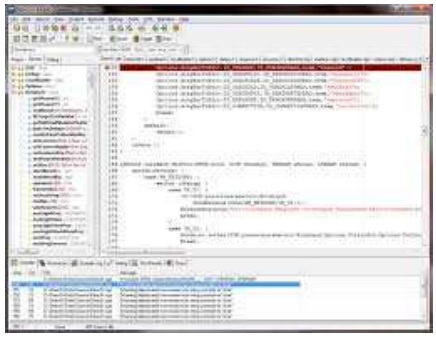


Perché modellare un sistema??

Il 'disegno' del sistema diventa CODICE che lo implementa

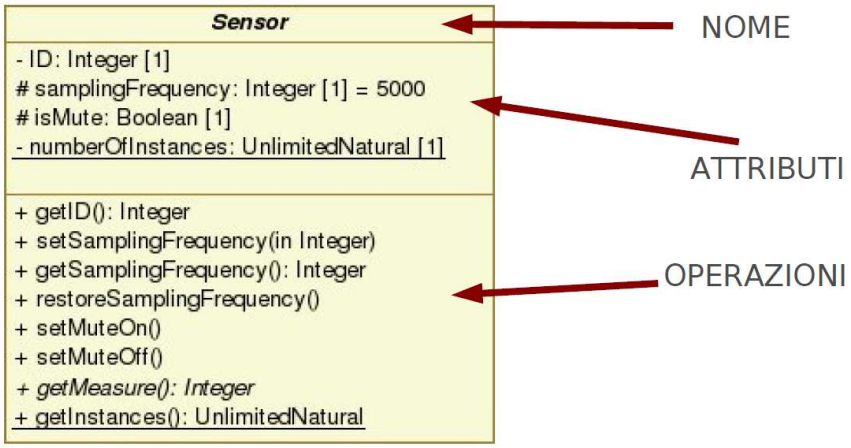


Class Diagram



Codice Java (per esempio)

Un altro esempio di classe: 'sensor' e la sua traduzione in Java



Nome di una classe

Definisce il nome di un'entità

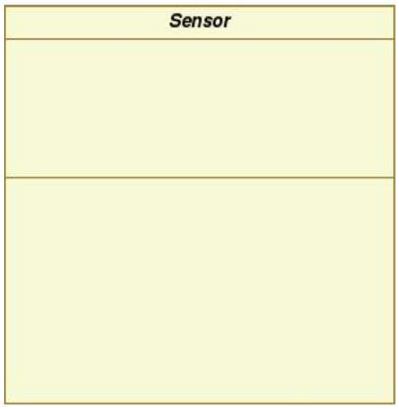
- stringa di testo
  - semplice stringa
  - path (prefisso + "::" + <nome_classe>)

Convenzione

- lettera iniziale maiuscola

Come rappresentare il nome di una classe in Java

```
public class Sensor {
  /** Corpo della
    *     classe
  */
}
```



Attributi di una classe

Gli attributi modellano le proprietà di una classe e per ogni attributo viene definito l'insieme di valori (tipo) che esso può assumere

Le proprietà sono condivise tra tutti gli oggetti appartenenti a quella classe, vale a dire che tutte le istanze hanno quella classe come tipo

Nel nostro esempio:

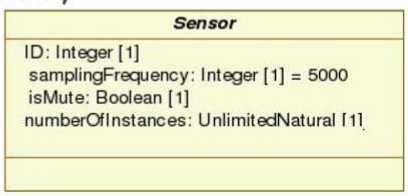
un sensore è caratterizzato da

identificativo (univoco)

frequenza di rilevamento della misura

Come rappresentare gli attributi di una classe in Java

```
public class Sensor {
  int ID;
  int samplingFrequency = 5000;
  boolean isMute;
  int numberOfInstances;
}
```



Operazioni di una classe

Le operazioni manipolano lo stato degli oggetti (ovvero il valore degli attributi)

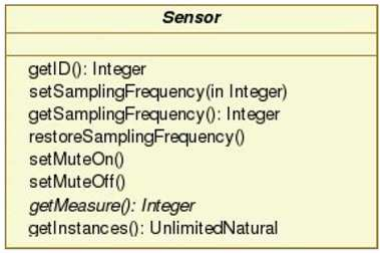
Hanno una segnatura: un tipo, un nome e una lista di parametri

Operations compartment specifica cosa può fare una classe (e non come), ovvero che servizi offre

Ad esempio il nostro sensore :

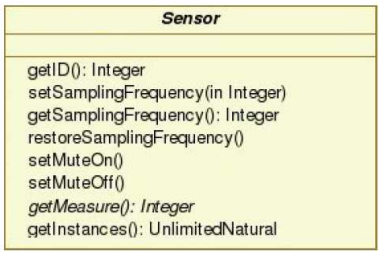
setSamplingFrequency,

getMeasure, …

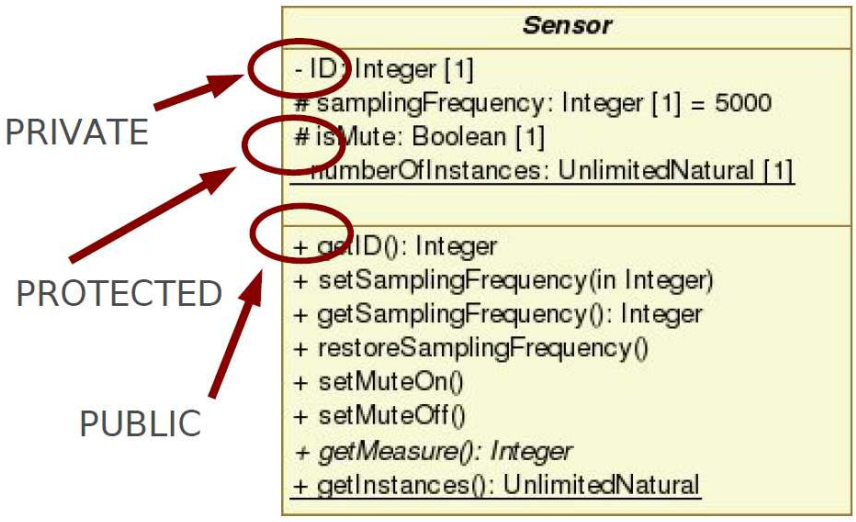


Come rappresentare le operazioni di una classe in Java

```
public class Sensor {
   int getID(){}
   void setSamplingFrequency(int i){}
   int getSamplingFrequency(){}
   void setMuteOn (){}
   void setMuteOff (){}
   int getMeasure(){}
   int getInstances (){}
}
```



Visibilità di un elemento della classe



Visibilità per attributi/metodi

PUBLIC: attributo/metodo che può essere usato ed è visualizzato all'esterno della classe nella quale è definito.

PRIVATE: attributo/metodo che NON può essere usato e NON è visualizzato all'esterno della classe nella quale è definito.

PROTECTED: attributo/metodo che può essere usato ed è visualizzato solo dalle classi che hanno una *relazione di generalizzazione* con la classe nella quale è definito.

Public, private, protected?!

Obiettivo: nascondere le scelte che possono essere soggette a cambiamenti

- REGOLA PRATICA : gli attributi in genere sono privati, le operazioni possono essere pubbliche

Chiaramente

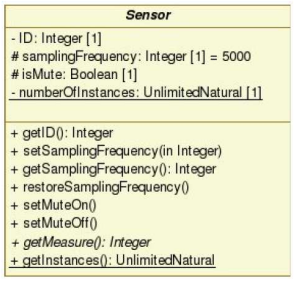
- l'interfaccia delle operazioni deve essere stabile
- vanno definite operazioni di get/set per accedere agli attributi privati

Principio di information hiding (encapsulation)

- separazione tra "interfaccia" e implementazione

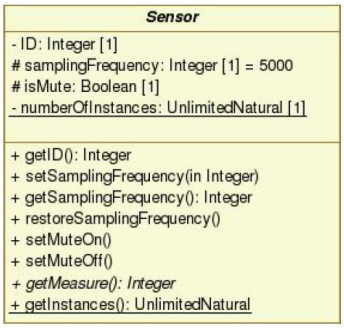
Come rappresentare la visibilità (public/private) in Java

```
public class Sensor {
  public void Sensor(){
    this.numberOfInstances ++;
     // OK
  }
}
public class testClass {
  public void testMethod(){
    Sensor s = new Sensor();
    s.setSamplingFrequency(23);// OK
    s.ID = 45; // ERRORE
  }
}
```



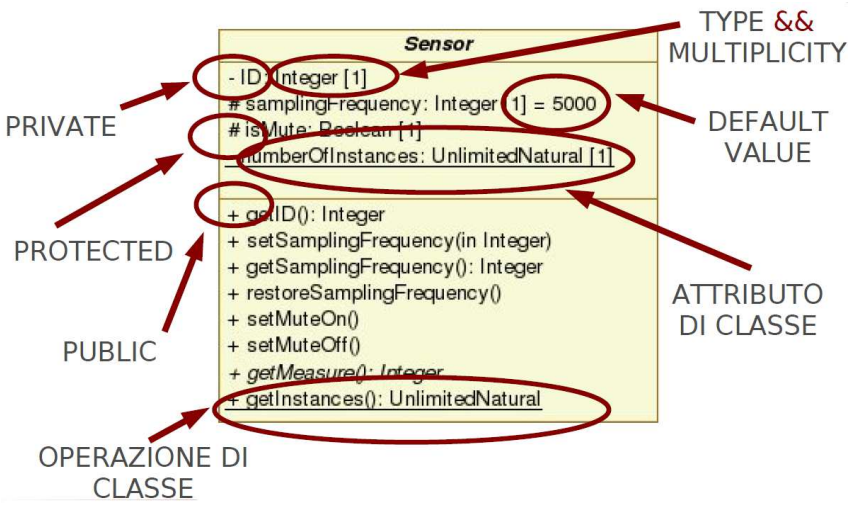
Come rappresentare la visibilità (protected) in Java

```
public class Sensor {
  public void setMuteOn(){
     this.isMute = true;// OK
  }
}
public class testClass {
  public void testMethod(){
    Sensor s = new Sensor();
    s.isMute = true ; // ERRORE
  }
}
```



NOTA BENE: **protected** e **private** sembrano equivalenti, non è cosi.
La differenza c'è con la *relazione di generalizzazione*.

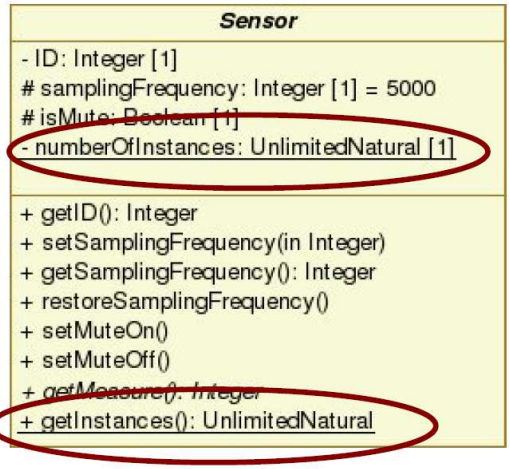
Visibilità degli elementi di una classe



Attributi ed operazioni delle classi

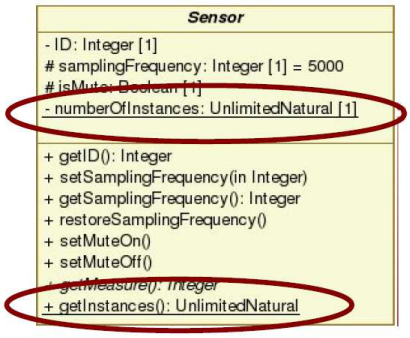
attributi : il valore è condiviso tra tutte le istanze

operazioni : non richiedono una istanza della classe per essere invocate



Come rappresentare attributi ed operazioni di una classe in Java

```
public class Sensor {
   ...
   private static int
     numberOfInstances;
   ...
   public static int
     getInstances (){}
}
```



come accederli da un'altra classe

```
int n1 = Sensor.getInstances();
...
Sensor s = new Sensor();
int n2 = s.getInstances();
```


Class Diagram

Il Class Diagram viene usato per modellare la struttura statica di un sistema ed è costituito da:

- entità + relazioni

Le **relazioni** di base in un class diagram

- corrispondono alla definizione delle possibili interazioni tra le classi di un modello
- una relazione tra una classe A ed una classe B significa che (in qualche modo) A può comunicare con B

Il tipo di relazione definisce il modo di comunicazione tra due classi

Class Diagram

Mostra un insieme di classi, interfacce e le relazioni tra loro

> *dipendenza, associazione, aggregazione, composizione, generalizzazione*

Può essere visto come un grafo dove i nodi sono classi/interfacce e gli archi sono relazioni

Possono contenere anche **package** o sottosistemi (usati per raggruppare elementi)

Class Diagram

Modella la struttura statica di una applicazione

> elementi specificati e/o composti a design time

Si usa per modellare:

> gli elementi di una applicazione (vocabolario) una classe è l'**astrazione** di un elemento nel dominio del problema

Semplici collaborazioni

> una classe non vive da sola ma si **relaziona** con altre al fine di fornire, "cooperativamente", un comportamento complesso

Relazioni in un Class Diagram

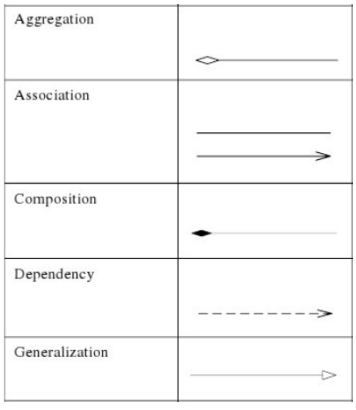
Una relazione rappresenta una “connessione” tra gli elementi di uno o più domini

Nei class diagram si specificano relazioni tra :

classi, oggetti, interfacce, package, etc...

Una relazione fornisce un “percorso di comunicazione” tra gli elementi del diagramma

Relazioni in un Class Diagram



Generalizzazione

Relazione tra una classe più generale ed una più specifica

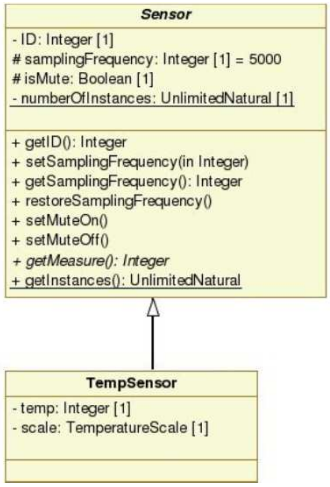
padre → super-classe

figlia → sotto-classe

Rappresentata da una freccia con triangolo bianco (verso la super-classe)

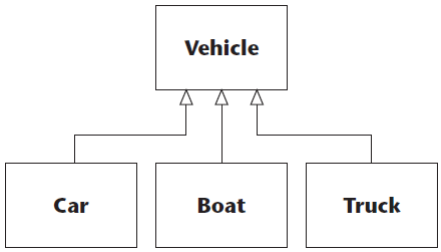
Una sotto-classe eredita tutte le caratteristiche della super-classe

is-a-kind-of



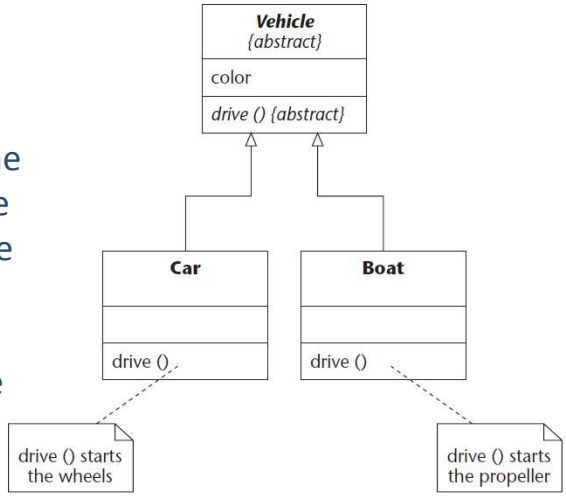
Alcuni esempi di generalizzazione

Il veicolo è una classe generale (super-classe) derivata per classi specifiche (sotto-classi) tramite l'ereditarietà (generalizzazione).



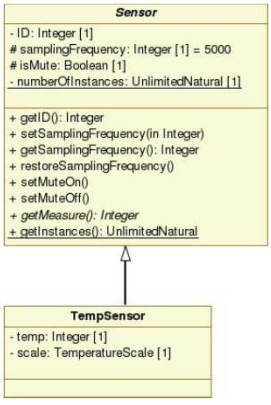
Generalizzazione e polimorfismo

La classe 'Car' eredita l'attributo 'color' e l'operazione 'drive'. L'operazione viene ridefinita nelle classi 'car' e 'boat'. La classe 'vehicle' è astratta, ed è anche segnato.



Generalizzazione in Java

```
public class TempSensor extends Sensor {
        private int temp;
        private TemperatureScale scale;
        ...
}
```



Quando usare la generalizzazione

PRINCIPIO DI SOSTITUIBILITA'

> Se q(x) è una proprietà che si può dimostrare essere valida per oggetti x di tipo T, allora q(y) deve essere valida per oggetti y di tipo S dove S è un sottotipo di T.

Che riformulato in termini più quotidiani:

> Data una super-classe T di S, in tutti i contesti in cui si usa un'istanza di T, deve essere possibile utilizzare una qualsiasi istanza di S (o di una qualunque altra sotto-classe a qualsiasi livello) la sotto-classe deve avere la **stessa semantica** della super-classe

Generalizzazione e classi…

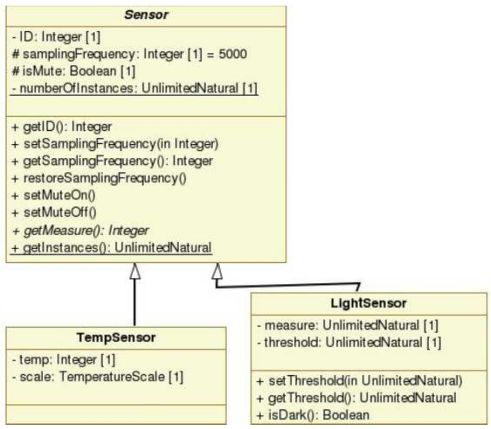
Problema :

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



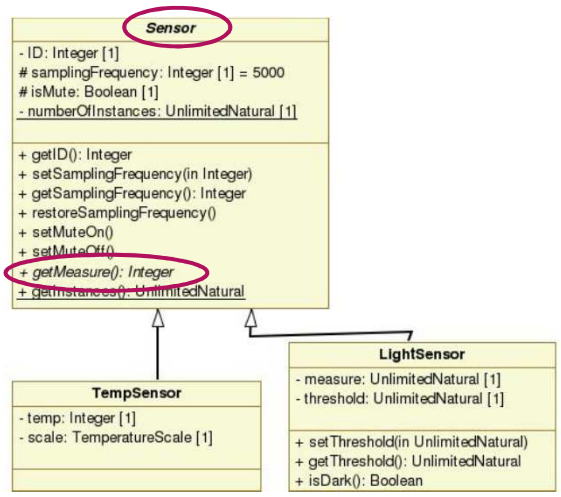
Generalizzazione e classi astratte

Parzialmente definite

modellare **operazioni** che hanno la stessa semantica ma che saranno implementate da **metodi** differenti

Non è mai possibile istanziare una classe astratta

Si possono (si devono) dichiarare puntatori classi astratte → **polimorfismo**



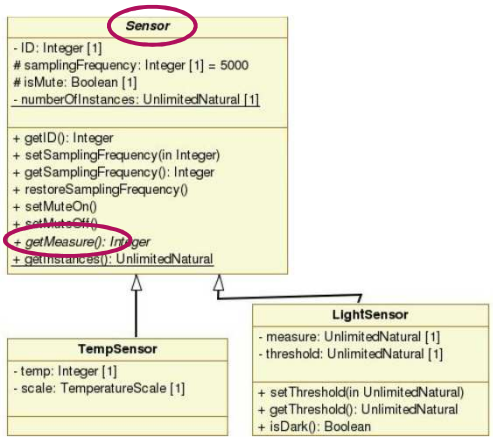
Generalizzazione e polimorfismo in Java

```
// Errore
Sensor s = new Sensor();

// OK
Sensor t = new TempSensor();

// OK
Sensor l = new LightSensor();

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

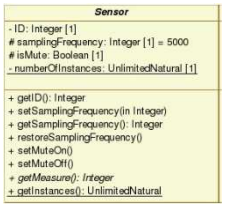


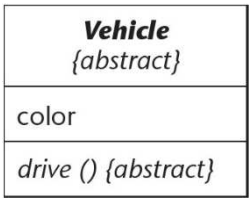
ATTENZIONE: anche se l e t sono dello stesso tipo, le chiamate si riferiscono a implementazioni diverse

Generalizzazione e classi astratte

Una classe è astratta se il flag *abstract* dell'elemento è impostato a true

Non c'è una simbologia grafica univoca

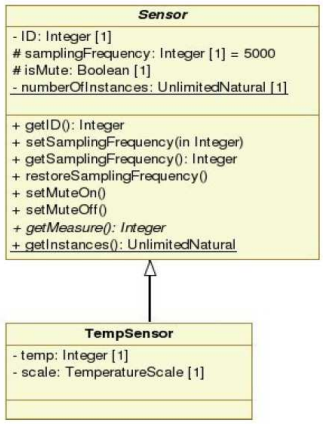




va per la maggiore!!

Generalizzazione e visibilità in Java

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



SOLO le sottoclassi POSSONO riferire gli elementi dichiarati **protected** nelle superclasse.

Interfacce

Collezioni di operazioni che sono utilizzate per specificare un servizio di una classe o di un componente

Definisce solo la segnatura delle operazioni

Le operazioni possono avere attributi di visibilità (come nelle classi)

Non tutti i linguaggi hanno interfacce

C++ NO , Java SI

Associazione

Relazione strutturale tra oggetti di classi differenti

Può essere simmetrica (navigabile nelle due direzioni)

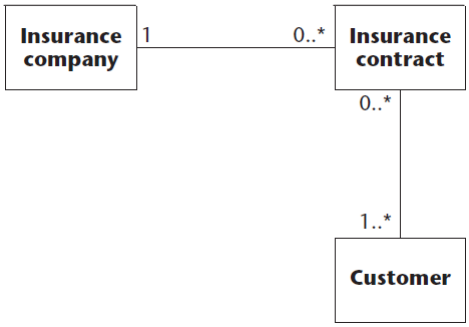
Una associazione può essere ricorsiva, ovvero tra oggetti della stessa classe

binaria o N-aria (è poco usata)

Rappresentata mediante una linea continua che collega le classi in relazione

Di fatto indica la possibilità che una classe possa inviare messaggi a quelle associate

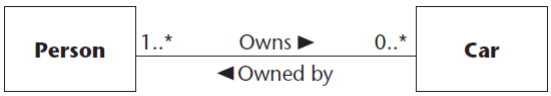
Un esempio di associazione tra classi



Una compagnia di assicurazioni ha molti contratti di assicurazione (zero-a-molti).
Un cliente può avere molti contratti di assicurazione (zero-a-molti).
Un contratto di assicurazione è connesso ad un'unica compagnia di assicurazioni.
Il contratto di assicurazione è legato a molti (uno-a-molti) clienti.

Altri esempi di associazione

Un autore utilizza un computer.
La classe Author è in **associazione** con la classe Computer.



Una persona può avere molte (zero-a-molti) automobili.
Una macchina può essere di proprietà di molte (uno-a-molti) persone.

Un'associazione navigabile dice che una persona può possedere molte vetture, ma non dice nulla su quante persone possono possedere una macchina.

Associazione in Java

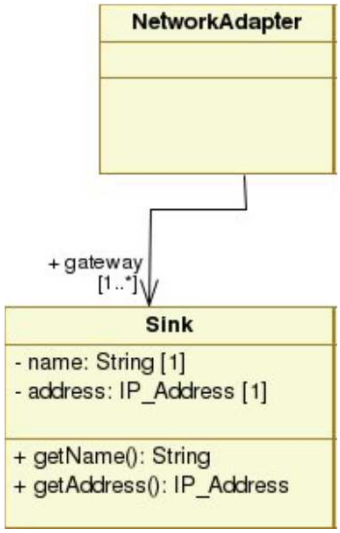
Un'associazione può avere:

un nome,

il ruolo degli operandi,

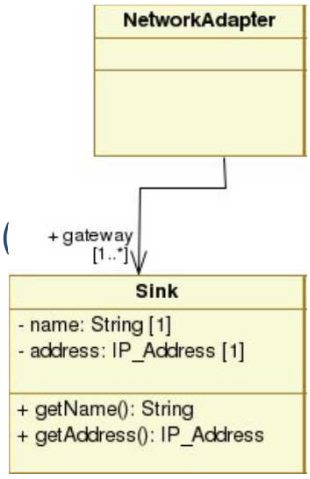
stereotipo

cardinalità



Associazione in Java

```
public class Sink {
        private String name;
        private IP_Address address;
        public String getName();
        public IP_Address getAddress(
}
public class NetworkAdapter {
        private Sink s;
}
```



Aggregazione

E’ una relazione di tipo gerarchico :

una classe che rappresenta una “entità autonoma”

una classe che rappresenta che aggrega l'“entità autonoma”

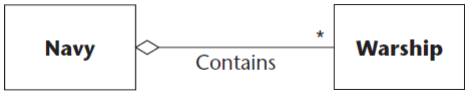
Esempio:

auto = motore + 4 ruote + scocca + ... altre cose

Denominata anche: relazione “whole-part”

Rappresentata mediante una linea tra la classe aggregante e quella aggregata. All'estremità della classe aggregante, la linea ha un rombo bianco

Alcuni esempi di aggregazione



La marina militare contiene molte navi da guerra. Alcune navi da guerra possono essere rimosse ed è ancora una marina. Alcune navi da guerra possono essere aggiunte, ed è ancora una marina.

Una squadra è composta da membri del team. Una persona potrebbe essere un membro di molte squadre. Si tratta di un esempio di aggregazione condivisa, dove le persone sono le parti condivise.

Aggregazione

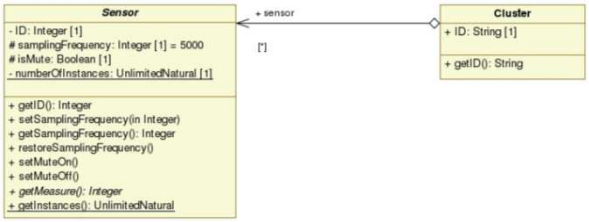
l'aggregazione non impone vincoli sul ciclo di vita degli elementi aggregati

le aggregazioni circolari sono semanticamente errate

A aggrega B

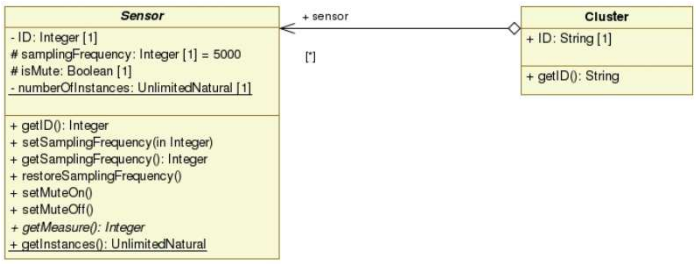
B aggrega C

C aggrega A



Aggregazione in Java

```
public class Cluster {
        private Vector<Sensor> vSensor;
        public Cluster (Vector<Sensor> s){
        this.vSensor= s;
        }
}
```



Composizione

E’ un’aggregazione forte

Le istanze composte non devono essere create con l’istanza della classe “componente”

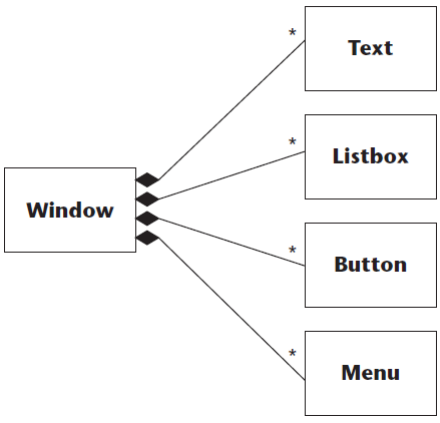
Una volta create, le istanze composte seguono il ciclo di vita dell’istanza che le compone

Dipendentemente dall'ambiente di sviluppo:

- la classe composita elimina le parti in un momento antecedente alla propria distruzione (in C++)
- l'implementazione è strutturata in modo che le istanze create con la composizione siano distrutte (con il garbage collector in Java)

Alcuni esempi di composizione

Il rombo scuro mostra la relazione di composizione, che vuol dire che ‘finestra’ contiene (è aggregata di) molti ‘menu’, ‘pulsanti’, ‘caselle di riepilogo’, e ‘descrizione testuale’. Tutti i tipi di composizione possono avere un nome.



Composizione

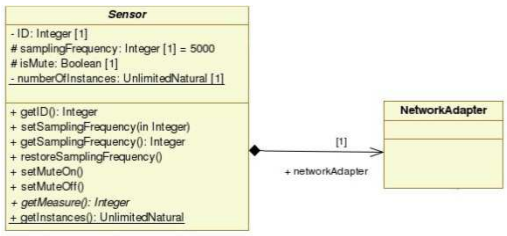
Conseguenze :

Un oggetto può essere parte di un solo oggetto composto alla volta, mentre nella relazione di aggregazione una parte può essere condivisa tra più oggetti composti.

È rappresentata mediante una linea tra la classe che compone (whole) e quella componente (part). All'estremità della classe che compone, la linea ha un rombo nero.

Composizione in Java

```
public abstract class Sensor {
 private NetworkAdapter na;
        public Sensor (NetworkAdapter net){
                this.na = new NetworkAdapter();
                /* Copy the state of net
                * into this.na
                */
        }
}
```

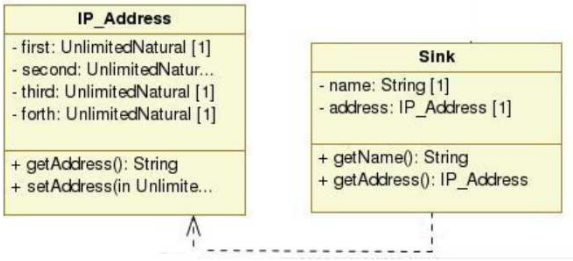


Dipendenza

È una relazione tra due elementi dove il cambiamento di un elemento può causare il cambiamento nell'altro

non necessariamente è implicato il viceversa

Rappresentata mediante una linea tratteggiata che collega le classi in relazione



Questions?

catia.trubiani@univaq.it