

# **Programmazione Orientata agli Oggetti**

---

Introduzione al Paradigma di  
Programmazione Orientato  
agli Oggetti

# Sommario

- Paradigmi di Programmazione
  - Il Paradigma Orientato agli Oggetti
  - Classi
  - Oggetti
  - Esercizio con Eclipse
  - Gli oggetti in *Rete*
- La notazione puntata
- Stato degli oggetti
  - Variabili di istanza
  - Inizializzazione
  - Campo d'Azione (Scope)
- Comportamento degli oggetti
  - Metodi

# Il Paradigma Orientato agli Oggetti (1)

- La divisione tra codice e operazioni è tipica dell'hardware moderno (memoria/CPU) secondo l'architettura di Von Neumann
- Ma gli esseri umani ragionano allo stesso modo?
- Ovvero, quando pensiamo ad un *oggetto* pensiamo solo al suo stato od anche alle operazioni che vi si possono compiere
- Ad esempio... un *televisore*...!?

# Il Paradigma Orientato agli Oggetti (2)

- Un problema è più naturalmente modellabile se pensato come popolato da una pluralità di *oggetti* che possiedono uno stato (*spento, acceso, sintonizzato* su un certo canale)

**Televisore, Persona, ecc...**

e che interagiscono ciascuno secondo i messaggi che sanno naturalmente interpretare:

**fabio.accende(tvInSalone)**

- Gli oggetti conoscono ed interagiscono con altri esemplari (anche dello stesso *tipo*)
  - Un oggetto **n11** di tipo **Stanza** conosce altri oggetti **Stanza**, ad esempio **n12**, nelle immediate adiacenze a formare un oggetto **Labirinto**

# Il Paradigma Orientato agli Oggetti (3)

- Ognuno di questi oggetti ha delle proprietà
  - *Peso, Nome, Altezza, ...*
- Ognuno di questi oggetti può svolgere delle azioni
  - Un oggetto **Macchina** può **accendersi()**
  - Un oggetto **Persona** può **salutare()**
- Risulta più naturale programmare se programmiamo in maniera più vicina a come naturalmente pensiamo

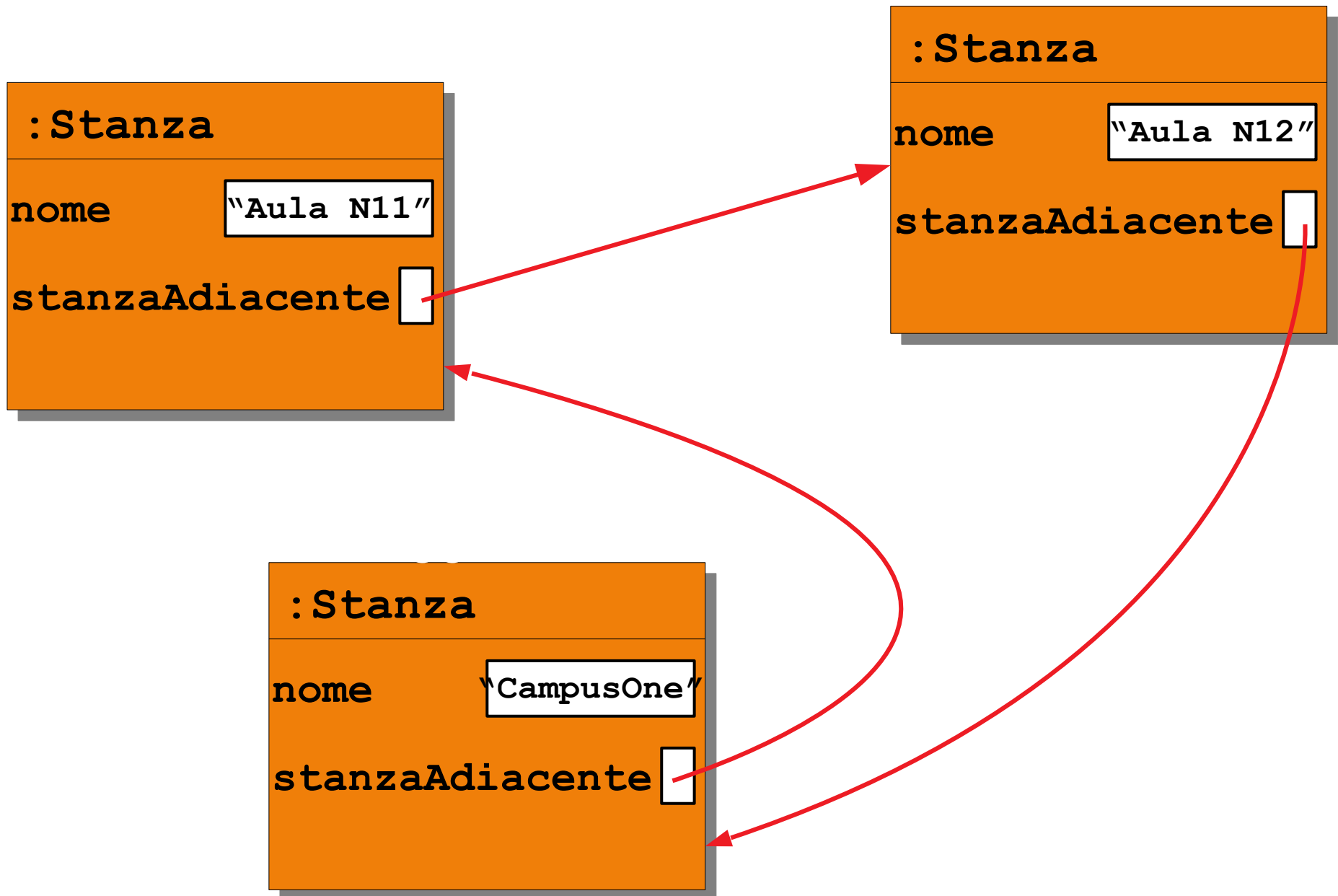
# Il Paradigma Orientato agli Oggetti (4)

- E' infatti possibile definire classi di oggetti che
  - mantengono uno stato
  - offrono operazioni che lo modificano/interrogano in maniera logicamente coesa
- La realizzazione di un programma consiste nella definizione di opportune classi di oggetti che si scambiano messaggi che sanno come interpretare
- Java è solo uno dei tanti linguaggi ideati per supportare la programmazione secondo il paradigma orientato agli oggetti
- Anzi... a ben vedere esistono fonti che spiegano come programmare in maniera orientata agli oggetti anche in linguaggio C (>>)

# La Rete di Oggetti (1)

- L'esecuzione di un programma Java si risolve nello scambio di messaggi tra oggetti che aggiornano ed interrogano il proprio stato
- Ad esempio, se si volesse rappresentare la topologia delle aule della nostra università, useremmo degli oggetti **Stanza** (**n10**, **n11**, **campusOne**, ...)
  - Per potersi scambiare messaggi, questi oggetti devono conoscersi tramite dei *riferimenti*
    - si crea una *Rete di Oggetti*

# La Rete di Oggetti (2)

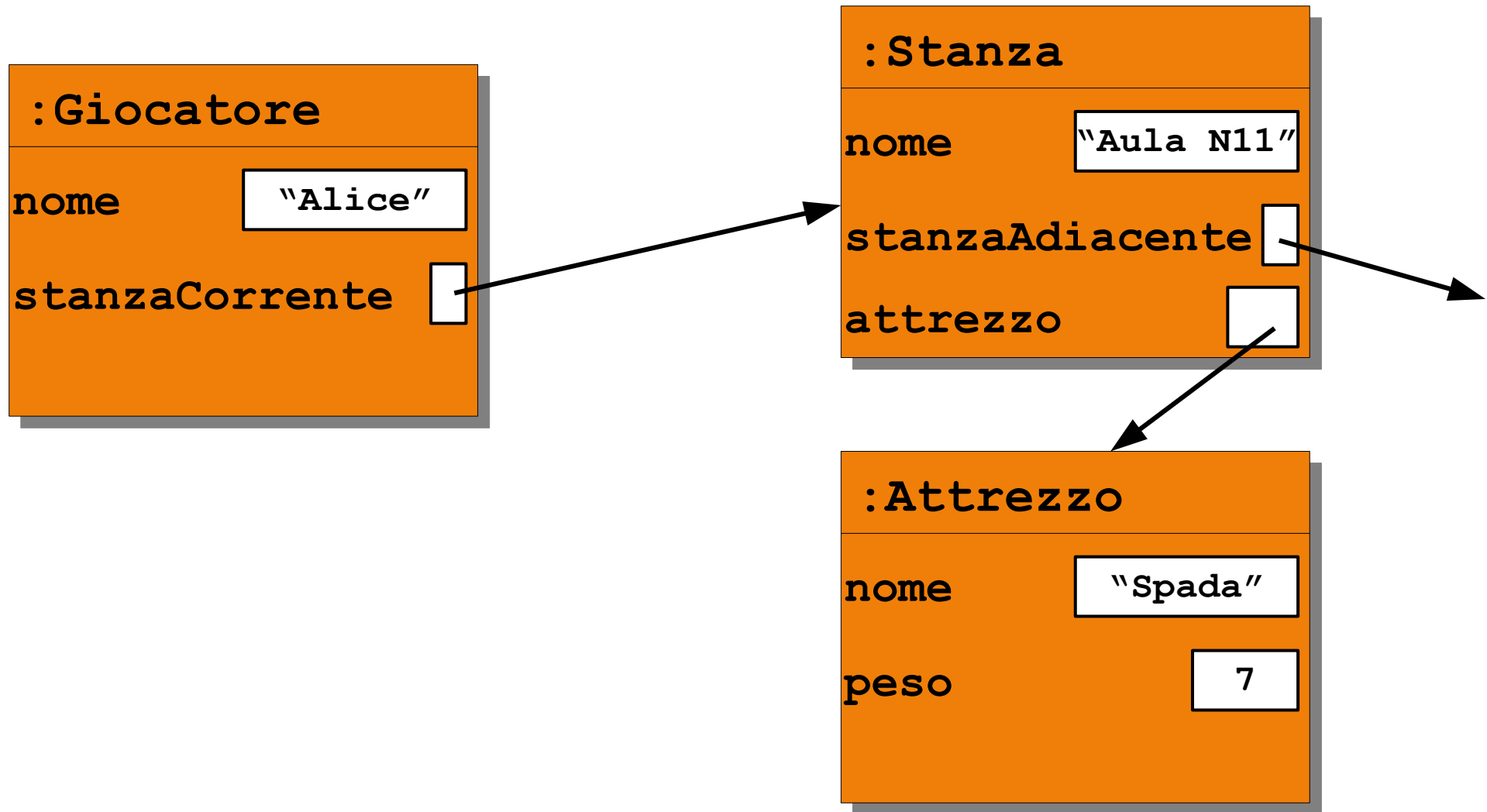




# La Rete di Oggetti (3)

- Grazie ad un riferimento è possibile rappresentare l'oggetto **Stanza** corrente di un ipotetico oggetto **Giocatore**
- Grazie ai collegamenti tra oggetti è possibile spostarsi all'interno di un oggetto **Labirinto** che *conosce* certamente le stanze
- Un oggetto **Stanza** può “contenere” un oggetto **Attrezzo**
- L'oggetto **Giocatore** potrebbe “chiedere” all'oggetto **Stanza** corrente quale oggetto **Attrezzo** possiede, usando operazioni come **prendiAttrezzo()** e **rimuoviAttrezzo()**
- Allo scopo deve scambiare messaggi con l'oggetto **Stanza**

# La Rete di Oggetti (4)



# Oggetti: Stato + Comportamento (1)

- Secondo il paradigma OO l'esecuzione di un programma avviene con la creazione di una rete di oggetti che si scambiano messaggi, aggiornano ed interrogano il proprio stato durante l'esecuzione
- Ogni oggetto
  - *possiede* uno stato interno
  - *offre* operazioni agli altri oggetti
- Gli oggetti sono dotati di:
  - *Stato*: informazioni memorizzate in **variabili di istanza** (o campi o attributi)
  - *Comportamento*: **metodi** che si possono invocare sull'oggetto
  - *Identità*: un oggetto può essere distinto dagli altri (anche e soprattutto da *esemplari* della stessa tipologia)

# Oggetti: Stato + Comportamento (2)

- Ad esempio l'oggetto **Giocatore** possiede come stato il suo nome e la stanza in cui si trova
- È possibile chiedere all'oggetto **Giocatore** di compiere delle azioni
  - ad es. **getStanzaCorrente()** per ottenere l'oggetto che rappresenta la stanza corrente
  - ad es. **setStanzaCorrente()** per spostarsi nella prossima stanza
- Può servire quindi interrogare l'oggetto **Stanza**
  - ad es. **getStanzaAdiacente()** per ottenere l'oggetto che rappresenta la stanza adiacente a quella considerata *corrente*

# Classe (1)

- Possono esistere vari oggetti dello stesso tipo
  - diverse stanze; attrezzi; giocatori
- Tutti gli oggetti di un certo *tipo* possiedono le informazioni dello stesso *tipo*
  - tutte le stanze hanno un **nome** ed una o più **stanzaAdiacente**
- Tutti gli oggetti di un certo *tipo* offrono le stesse operazioni
  - A tutti i giocatori è possibile chiedere di spostarsi nella prossima stanza o di raccogliere un oggetto **Attrezzo**

# Classe (2)

- È necessario un meccanismo per costruire oggetti:
  - 1) Possiedono una propria identità ed autonomia
  - 2) Possiedono informazioni e dati specifici...
  - 3) ...ma sembrano chiaramente rispondere anche ad altri tratti comuni a tutti gli oggetti della stessa tipologia
- Serve una sorta di *fabbrica specializzata* che definisca come tutti gli oggetti di un certo *tipo* siano fatti lasciando la libertà di variare alcuni aspetti
- Si pensi ad un fabbrica di *Orologi* tutti uguali
  - 1) Ciascun esemplare è distinto dagli altri
  - 2) Ciascun esemplare ha un numero di serie ed un colore diverso...
  - 3) ...ma tutti offrono la possibilità di leggere l'ora
- Nella programmazione orientata agli oggetti queste fabbriche sono dette **classi**

# Classe (3)

- Per ogni tipo di oggetto che si vuole rappresentare esiste una *classe*
  - Una per **Giocatore**, una per **Stanza**, ecc...
- Tutti gli oggetti sono costruiti a partire dalla definizione di una classe
- La classe astrae e definisce
  - lo stato di un oggetto di un certo tipo
  - il comportamento degli oggetti di un certo tipo

# Classe, un Esempio

- Tralasciamo per ora le stanze, gli attrezzi e i giocatori... Torneranno più tardi (>>)
- Più semplice utilizzare forme geometriche in un piano cartesiano con coordinate intere per un primo esempio di classe
  - per rappresentare una forma è necessario conoscerne la posizione
  - servono le coordinate di un punto sul piano cartesiano: **x**, **y**
    - si definisce una classe **Punto**



# La Classe Punto (1)

```
public class Punto {
```

```
    private int x;
```

```
    private int y;
```

} Stato

```
    public void setX(int posX) {
```

```
        x = posX;
```

```
    }
```

```
    public void setY(int posY) {
```

```
        y = posY;
```

```
    }
```

```
    public int getX() {
```

```
        return x;
```

```
    }
```

```
    public int getY() {
```

```
        return y;
```

```
    }
```

```
}
```

} Operazioni, per impostare e leggere lo stato

# La Classe Punto (2)

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

Ogni oggetto di tipo **Punto** ha una *x* e una *y*  
*variabili d'istanza*

Inoltre, dispone di una serie di operazioni per leggere e impostare il suo stato

*metodi*

# Creazione di Oggetti

- La creazione di un oggetto a partire da una classe avviene utilizzando l'operatore **new**:  
`Punto origine = new Punto();`
- Restituisce un *riferimento* ad un oggetto appena creato
- Ora è presente un oggetto in memoria ed è possibile chiedergli di svolgere delle operazioni per tramite del riferimento, qui conservato nella variabile locale **origine**
- Ad esempio per impostare la ascissa ed ordinata  
`origine.setX(0);`  
`origine.setY(0);`

# Concetti Correlati ma Ben Distinti!

Questa semplice istruzione

```
Punto origine = new Punto();
```

già nasconde almeno tre concetti  
chiaramente distinti sebbene correlati

*I. Variabile locale*

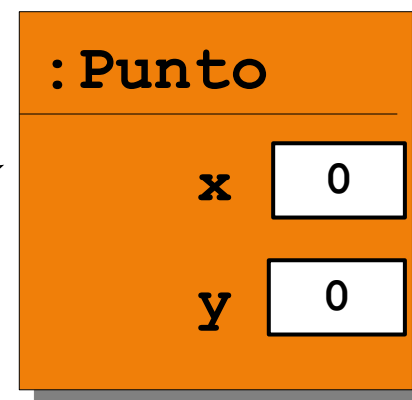
*II. Riferimento*

*III. Oggetto*

*Variabile locale*    `origine`

*riferimento*

*oggetto*



Ci torneremo più volte sopra!

# La Notazione Puntata

- Per richiedere un'operazione ad un oggetto si usa la cosiddetta *notazione puntata*  
`<referimento-oggetto>.<metodo>(<parametri-attuali>);`
- Ad esempio:
  - `origine.setX(0);`
- Anche solo questa notazione mostra come i dati siano stati *avvicinati* alle operazioni sugli stessi  
*dati.operazione(parametri)*

# Classi e Oggetti

- A partire da una classe possiamo creare molte *istanze*, ognuna dotata di uno stato autonomo

```
Punto unoZero = new Punto();
```

```
unoZero.setX(1);
```

```
unoZero.setY(0);
```

Un nuovo oggetto viene creato in memoria; è possibile chiedergli di svolgere operazioni attraverso il riferimento conservato nella variabile locale **unoZero**

- L'oggetto creato in precedenza (ovvero quello il cui riferimento è conservato dentro **origine**) risulta invariato dopo tali operazioni e non viene modificato dall'invocazione di metodi tramite il riferimento conservato dentro **unoZero** perché le operazioni sono svolte su un oggetto distinto

# Terminologia

- Nella programmazione orientata agli oggetti si utilizza la seguente terminologia:
  - **Oggetti o istanze**: istanze di una certa classe, presenti in memoria, mantengono lo stato di un oggetto
  - **Metodi**: specificano le operazioni che determinano il comportamento degli oggetti
    - Es.: `setX()`, `getY()`, ...
  - **Variabili di Istanza**: consentono di memorizzare le informazioni di ciascun oggetto (lo stato dell'oggetto)
  - **Riferimento** ad oggetto: un riferimento ad un oggetto in memoria (`>>`)

# Classi e File

- In Java, una classe di nome **xyz** *DEVE* essere contenuta all'interno di un file di nome **xyz.java**
  - esistono alcune eccezioni a questa regola che tralasciamo per il momento  
*classi nidificate (>>)*
- Da qualsiasi altra classe del nostro programma sarà possibile far riferimento ad una classe specificandone il suo nome
  - In Java *non* è necessario importare un file esplicitamente per poter usare il codice in esso contenuto
  - Ma bisogna mettere il compilatore (e la JVM durante l'esecuzione) in condizione di trovarlo a partire dal suo nome (>>)



# Esercizio: Eclipse (0)

- JDK (Standard Edition)
  - Scaricare ed installare dal sito della ORACLE l'ultima versione disponibile di Java 8  
[https://www.java.com/download/java8\\_update.jsp](https://www.java.com/download/java8_update.jsp)
  - Recommended Version 8 Update 321
  - Oppure Java 11
- Documentazione JDK
  - Scaricare ed installare dal sito della ORACLE
- Strumenti di sviluppo
  - Un IDE professionale: Eclipse (4.X)
- Scaricare **Eclipse IDE for Java Developers** da  
<https://www.eclipse.org/downloads/eclipse-packages/>  
(è la versione usata anche in sede d'esame)
- Configurare Eclipse per compilare **Java 7**

# Esercizio: Eclipse (1)

- Realizzare la classe **Punto** usando l'IDE eclipse
  - *File > New > Java Project*
    - *Project Name: Forme > click su 'Finish'*
  - Selezionare il progetto appena creato
  - *File > New > Class*
    - *Name: Punto > click su 'Finish'*
- Oltre al codice mostrato prima, aggiungere il metodo `public void trasla(int dx, int dy)` che sposta il punto di `dx` sull'asse x e `dy` sull'asse y

# Esercizio: Eclipse (2)

Realizzare la classe **MainForme** usando l'IDE eclipse

- Creare una classe di nome **MainForme**
- Aggiungere il metodo **main()**

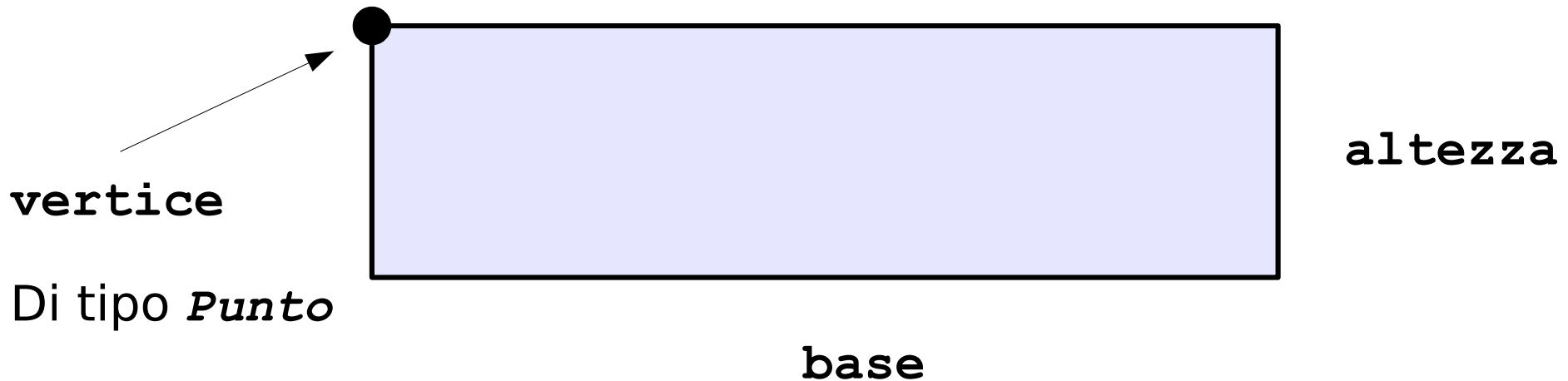
```
public class MainForme {  
    public static void main(String[] args) {  
        Punto origine = new Punto();  
        origine.setX(0);  
        origine.setY(0);  
        System.out.println(origine.getX());  
        System.out.println(origine.getY());  
        origine.trasla(1, 1);  
        System.out.println(origine.getX());  
    }  
}
```

# Gli Oggetti in *Rete* (1)

- La *definizione* di un programma “orientato agli oggetti” consiste nella definizione di diverse *classi* di oggetti
- L'*esecuzione* di un programma orientato agli oggetti avviene orchestrando lo scambio di messaggi tra un pluralità di oggetti istanza delle classi definite nel programma
  - gli oggetti devono “conoscersi”
    - un oggetto può possedere *riferimenti* verso gli altri oggetti
    - gli oggetti possono inviare messaggi ad altri oggetti dei quali possiedono un *riferimento*
- Si vuole realizzare la classe **Rettangolo**
  - Stato composito:
    - base
    - altezza
    - posizione del vertice in alto a sx

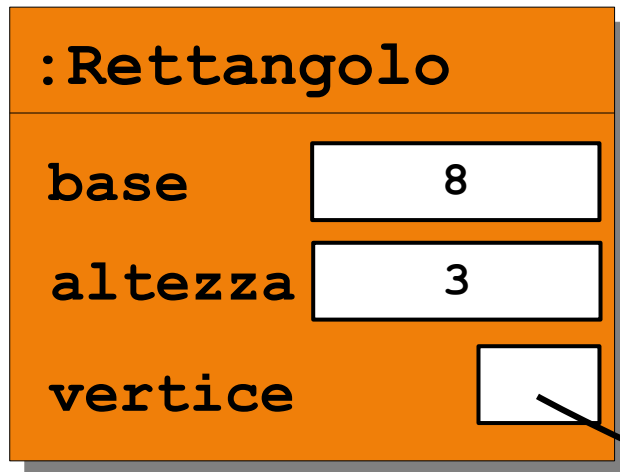
# Gli Oggetti in *Rete* (2)

- Il vertice in alto a sinistra è un oggetto istanza della classe **Punto**
  - di coordinate (x, y)
- Ogni oggetto della classe **Rettangolo** deve conoscere un oggetto della classe **Punto** che rappresenta il suo vertice in alto a sinistra



# Gli Oggetti in *Rete* (3)

*oggetto*



*riferimento  
ad oggetto*

*oggetto*



# *(Con Eclipse)* La Classe Rettangolo

- Nello stesso progetto della classe Punto

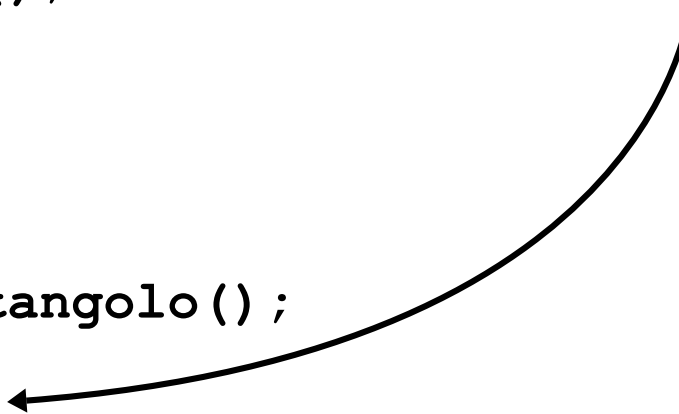
```
public class Rettangolo {  
    private int base;  
    private int altezza;  
    private Punto vertice;  
  
    public void setBase(int b) { base = b; }  
    public void setAltezza(int a) { altezza = a; }  
    public void setVertice(Punto v) { vertice = v; }  
  
    public int getBase() { return base; }  
    public int getAltezza() { return altezza; }  
    public Punto getVertice() { return vertice; }  
}
```

# La Classe Rettangolo (2)

- Il `main()` può essere modificato come segue

```
public class MainForme {  
    public static void main(String[] args) {  
        Punto origine = new Punto();  
        origine.setX(0);  
        origine.setY(0);  
  
        Rettangolo rect = new Rettangolo();  
        rect.setVertice(origine);  
        rect.setBase(8);  
        rect.setAltezza(3);  
        // ...  
        // Seguono stampe per verificarne il funzionamento  
    }  
}
```

Dopo questa istruzione l'oggetto istanza di **Rettangolo** conosce l'oggetto istanza di **Punto**





# Messaggi tra Oggetti (1)

- Si vuole implementare il metodo  
`sposta(int deltaX, int deltaY)`  
nella classe `Rettangolo`
  - Per traslare il suo vertice il rettangolo può chiedere al suo stesso vertice di spostarsi: scambio di messaggi tra oggetti

```
public class Rettangolo {  
    // ... Come prima ...  
    public void sposta(int deltaX, int deltaY) {  
        vertice.trasla(deltaX, deltaY);  
    }  
}
```

# Messaggi tra Oggetti (2)

- E se la classe `Punto` non disponesse del metodo `trasla()`?

# Messaggi tra Oggetti (3)

```
public class Rettangolo {  
    // ... come prima ...  
    public void sposta(int deltaX, int deltaY) {  
        int xVertice = vertice.getX();  
        int yVertice = vertice.getY();  
        vertice.setX(xVertice + deltaX);  
        vertice.setY(yVertice + deltaY);  
    }  
}
```

- Il comportamento desiderato è comunque ottenibile utilizzando i metodi `setX()` e `setY()` ma il codice risulta “meno pulito” rispetto alla soluzione basata sulla disponibilità del metodo `trasla()` già all’interno della classe `Punto`

# Variabili di Istanza e Metodi

- Una classe definisce sia delle variabili di istanza sia dei metodi; rappresentano, rispettivamente:
  - *lo stato* degli oggetti istanza di quella classe
  - *il comportamento* di tali oggetti
- La definizione di una classe segue questa sintassi di fatti:

```
public class <NomeClasse> {  
    <definizione di variabili di istanza>  
    <definizione di metodi>  
}
```

# Variabili di Istanza (1)

- Le variabili di istanza memorizzano informazioni che rappresentano lo stato di un oggetto

```
public class Rettangolo {  
    private int base;  
    private int altezza;  
    private Punto vertice;  
    ...  
}
```

# Variabili di Istanza (2)

- Nella definizione di una variabile di istanza:
  - Modificatore di visibilità (>>)
  - Tipo
  - Nome della variabile

**private int base;**



Modificatore di visibilità

Tipo

Nome della variabile

- Il modificatore di visibilità specifica se una certa variabile è visibile dall'esterno
  - Per il momento si usa **private**: la variabile è visibile solo all'interno della classe in cui è dichiarata
  - Per accedervi dall'esterno si usano i metodi *getter* e *setter*

# Variabili Istanza: Inizializzazione

- Ci sono diversi modi per inizializzare lo stato di un oggetto
- Le variabili di qualsiasi genere (ovvero di istanza, locali ed altro >>) vengono *sempre* inizializzate, esplicitamente od implicitamente
- In Java non esiste il problema delle variabili accidentalmente rimaste non inizializzate tipico del linguaggio C
- Per le variabili di istanza: se non viene specificato alcun valore iniziale, assumono un valore di default:
  - per le variabili di un tipo numerico (`int`, `float`...) è 0

```
Rettangolo rect = new Rettangolo();  
System.out.println(rect.getBase()); // Stampa 0
```


- Stampa (sempre e prevedibilmente) 0 nonostante non sia stato invocato il metodo `setBase(0)` ;

# Variabili di Istanza: Campo d'Azione (o *Scope*)

- Le variabili di istanza sono visibili all'interno della sola classe in cui sono dichiarate
  - Ogni metodo può referenziarle semplicemente per nome

```
public class Rettangolo {  
    private int base;  
    // ...  
    public int getBase() {  
        return base;  
    }  
    // ...  
}
```

Qui 'base' fa  
riferimento alla  
variabile di  
istanza **base**





# Variabili di Istanza

- Una variabile di istanza ha un valore come parte dello stato di uno specifico oggetto istanza della sua classe
- Si usa dire che una variabile di istanza *“appartiene ad un oggetto”* anche se è *definita* nella sua classe
- Un oggetto, tramite le proprie variabili di istanza, possiede un proprio stato *autonomamente* rispetto a tutti gli altri oggetti istanza della sua stessa classe

```
Rettangolo rect1 = new Rettangolo();  
rect1.setBase(10);
```

```
rect1.base; // NON COMPILA
```

```
Rettangolo rect2 = new Rettangolo();  
rect2.setBase(20); /* N.B. la base del secondo oggetto cambia;  
                    Quella del primo rimane invariata */
```

# Un Parallelismo con il Linguaggio C (1)

- Pare abbastanza naturale associare una variabile di istanza di una classe Java ad un campo di una **struct** di C

```
typedef struct {  
    int base;  
    ...  
} Rettangolo;
```

```
Rettangolo *r = malloc(sizeof(Rettangolo)) ;  
r->base = 15;  
Rettangolo *r2 = malloc(sizeof(Rettangolo)) ;  
r2->base = 30;  
free(r1) ;  
free(r2) ;
```

# Un Parallelismo con il Linguaggio C (2)

- I metodi definiscono le operazioni che si possono svolgere su un oggetto di una certa classe
- Viene naturale associare un metodo di una classe Java ad una funzione C che opera su una **struct**

```
typedef struct {  
    int base;  
  
    ...  
} Rettangolo;
```

```
void setBase(Rettangolo *this, int base) {  
    this->base = base;  
}
```

# Invocazione dei Metodi

- L'invocazione dei metodi è alla base della programmazione orientata agli oggetti come meccanismo per lo scambio di messaggi tra oggetti
- I metodi mettono in comunicazione diretta l'oggetto che invoca il metodo con quello su cui il metodo viene invocato
- Ad esempio:
  - Per impostare od ottenere la base di un rettangolo abbiamo invocato dei metodi della classe **Rettangolo**
  - Per spostare un oggetto istanza della classe **Rettangolo** il suo metodo **sposta()** ha invocato dei metodi della classe **Punto** una cui istanza ne rappresenta il vertice

# Metodo `main()`

- L'esecuzione di un programma inizia sempre con l'invocazione di un particolare e specifico metodo
- Per convenzione (eredità dal linguaggio C) tale metodo si chiama `main()`
  - Questo metodo “scatena” l'esecuzione invocando a sua volta altri metodi
- Tranne che per il metodo `main()` da cui comincia l'esecuzione, per ogni invocazione di metodo esiste sempre un metodo *invocante* ed un metodo *invocato*

# Metodo Invocante e Invocato

*Invocazione di metodo*

*Metodo  
invocante*

```
public class Main {  
    public static void main(String args[]) {  
        Rettangolo rect = new Rettangolo();  
  
        rect.setBase(22);  
    }  
}
```

*Metodo  
invocato*

# Definizione di Metodo

- I metodi sono dichiarati all'interno della definizione di una classe e definiscono il comportamento di tutti gli oggetti appartenenti a quella classe
- La dichiarazione di un metodo comprende due parti:
  - Intestazione
    - Modificatore di accesso/visibilità
    - Tipo valore restituito
    - Nome del metodo
    - Lista dei parametri formali
  - Corpo
    - Definizioni di variabili locali
    - Istruzioni

*Intestazione* *Corpo*

```
public void setX(int x) { ... }
```

# Metodi: Valore Restituito

- I metodi possono comunicare verso l'esterno restituendo un valore
  - Il metodo *invocato* comunica con il metodo *invocante*
  - esattamente come per le funzioni in C
- Se un metodo non ritorna nessun valore al momento della dichiarazione del tipo di ritorno si utilizza la parola chiave **void**



# Metodi e Aggiornamenti di Stato

- Conviene, per diversi motivi (>>), distinguere sempre i metodi che
  - **interrogano** (solamente) lo stato dell'oggetto su cui sono invocati
    - Solo *lettura* dello stato
  - **aggiornano** lo stato dell'oggetto su cui sono invocati
    - Anche *scrittura* dello stato
- Ad esempio (nella classe **Punto**)

```
public int getX() {  
    return x;           // interroga lo stato  
}  
  
public void trasla(int dx, int dy) {  
    x += dx;            // aggiorna lo stato  
    y += dy;            // aggiorna lo stato  
}
```

# *(Esercizio con Eclipse)*

## Variabili di Istanza e Metodi

- Realizzare la classe **Attrezzo**
  - Con le variabili di istanza
    - **nome** di tipo **String**
    - **peso** di tipo **int**
  - aggiungere i relativi metodi *getter & setter*
- Realizzare la classe **Stanza**
  - Con le variabili di istanza
    - **nome** di tipo **String**
    - **stanzaAdiacente** di tipo **Stanza**
    - **attrezzoContenuto** di tipo **Attrezzo**
  - aggiungere i relativi metodi *getter & setter*

# Modificare lo Stato di un Oggetto (1)

- Lo stato di un oggetto può essere cambiato

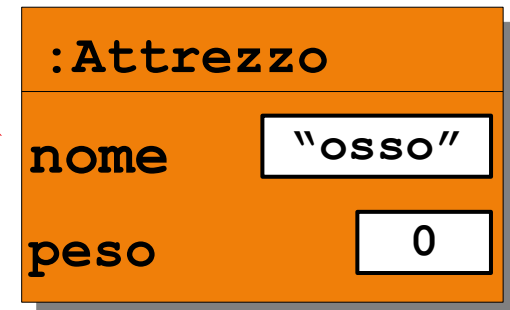
```
public class MainStanzeAttrezzi {  
    public static void main(String[] args) {  
        Attrezzo spada = new Attrezzo();  
        spada.setNome("spada");  
        spada.setPeso(7);  
  
        Attrezzo osso = new Attrezzo();  
        Osso.setNome("osso");  
        Osso.setPeso(1);  
  
        Stanza n11 = new Stanza();  
        n11.setNome("N11");  
  
        n11.setAttrezzo(spada);  
  
        n11.setAttrezzo(osso);  
    }  
}
```




# Modificare lo Stato di un Oggetto (2)

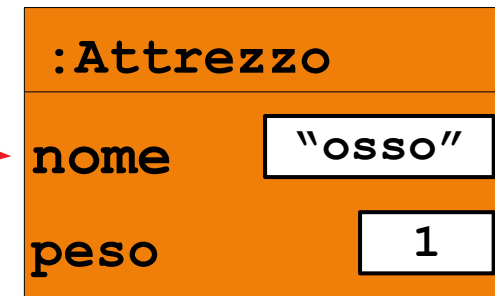
- Lo stato di un oggetto può essere cambiato

```
public class MainStanzeAttrezzi {  
    public static void main(String[] args) {  
        Attrezzo spada = new Attrezzo();  
        spada.setNome("spada");  
        spada.setPeso(7);  
  
        Attrezzo osso = new Attrezzo();  
        osso.setNome("osso");  
        osso.setPeso(1);  
  
        Stanza n11 = new Stanza();  
        n11.setNome("N11");  
  
        n11.setAttrezzo(spada);  
  
        n11.setAttrezzo(osso);  
    }  
}
```



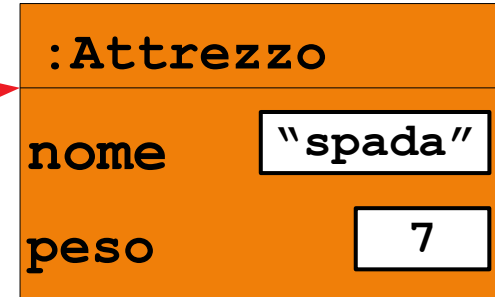
# Modificare lo Stato di un Oggetto (3)

```
public class MainStanzeAttrezzi {  
    public static void main(String[] args) {  
        Attrezzo spada = new Attrezzo();  
        spada.setNome("spada");  
        spada.setPeso(7);  
  
        Attrezzo osso = new Attrezzo();  
        osso.setNome("osso");  
        osso.setPeso(1);  
          
        Stanza n11 = new Stanza();  
        n11.setNome("N11");  
  
        n11.setAttrezzo(spada);  
  
        n11.setAttrezzo(osso);  
    }  
}
```

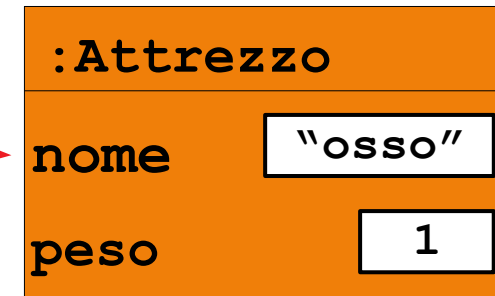


# Modificare lo Stato di un Oggetto (4)

```
public class MainStanzeAttrezzi {  
    public static void main(String[] args) {  
        Attrezzo spada = new Attrezzo();  
        spada.setNome("spada");  
        spada.setPeso(7);
```



```
        Attrezzo osso = new Attrezzo();  
        Osso.setNome("osso");  
        Osso.setPeso(1);
```



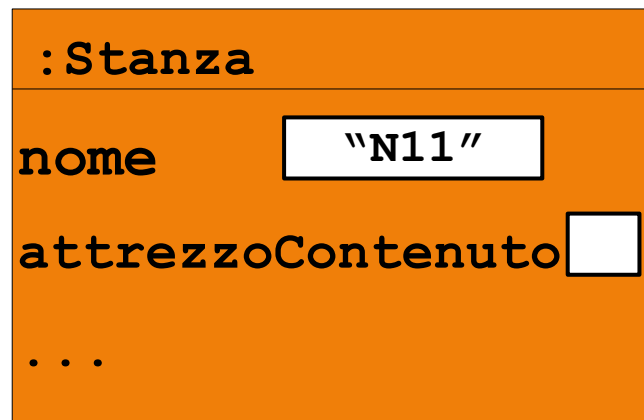
```
        Stanza n11 = new Stanza();  
        n11.setNome("N11");
```

```
        n11.setAttrezzo(spada);
```

```
        n11.setAttrezzo(osso);
```

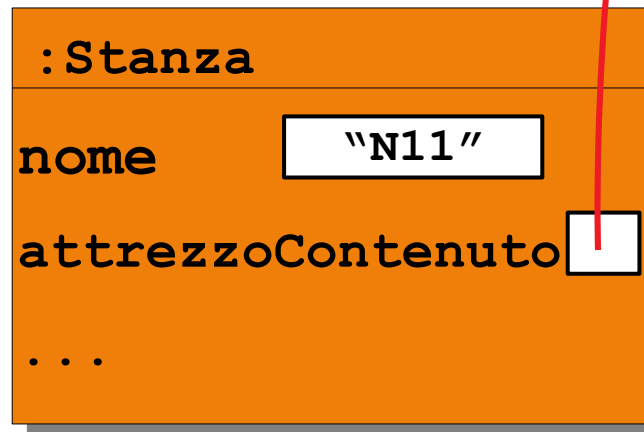
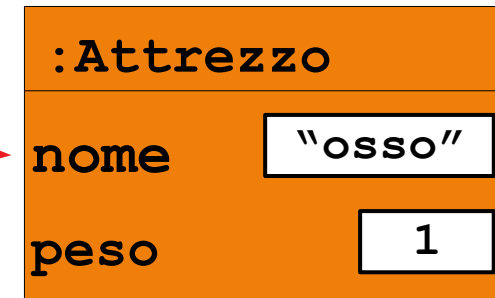
```
    }
```

```
}
```



# Modificare lo Stato di un Oggetto (5)

```
public class MainStanzeAttrezzi {  
    public static void main(String[] args) {  
        Attrezzo spada = new Attrezzo();  
        spada.setNome("spada");  
        spada.setPeso(7);  
  
        Attrezzo osso = new Attrezzo();  
        Osso.setNome("osso");  
        Osso.setPeso(1);  
  
        Stanza n11 = new Stanza();  
        n11.setNome("N11");  
  
        n11.setAttrezzo(spada);  
  
        n11.setAttrezzo(osso);  
    }  
}
```



# Modificare lo Stato di un Oggetto (6)

```
public class MainStanzeAttrezzi {  
    public static void main(String[] args) {  
        Attrezzo spada = new Attrezzo();  
        spada.setNome("spada");  
        spada.setPeso(7);
```

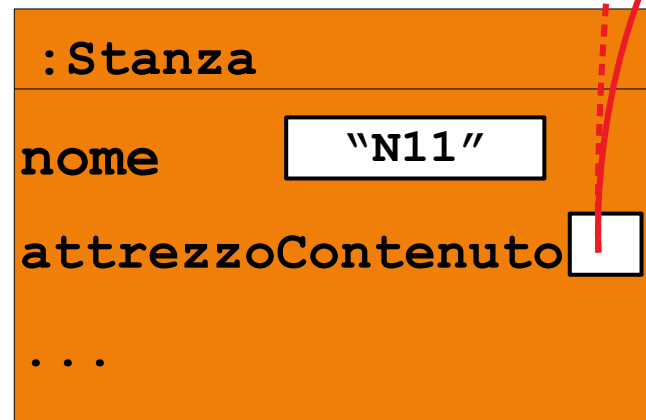
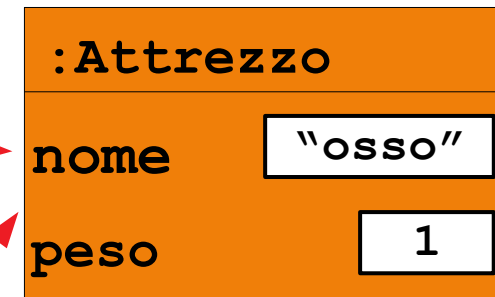
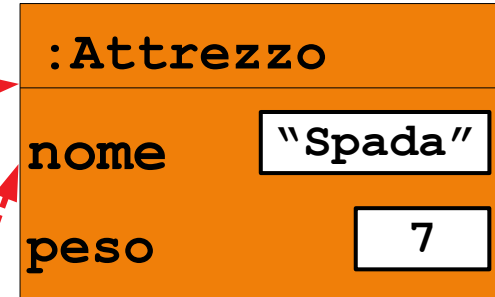
```
        Attrezzo osso = new Attrezzo();  
        Osso.setNome("osso");  
        Osso.setPeso(1);
```

```
        Stanza n11 = new Stanza();  
        n11.setNome("N11");
```

```
        n11.setAttrezzo(spada);
```

```
        n11.setAttrezzo(osso);
```

```
    }  
}
```





# Riferimenti ad Oggetti (continua...)

- La creazione di un nuovo oggetto in memoria avviene tramite l'operatore **new**
- L'operatore **new** restituisce un *riferimento ad un oggetto* appena creato
- Ad esempio: `Stanza n11 = new Stanza();`
- La variabile locale **n11** **NON** contiene l'oggetto creato, ma bensì un *riferimento* ad esso (>>)

