

Programmazione a Oggetti

Lezione 7

Il linguaggio Java: aspetti generali

Sommario

- Storia e Motivazioni
- Definizione di Classi
- Campi e Metodi
- Istanziamento di oggetti
- Tipi

Obiettivi di JAVA

Portabilità: produrre codice eseguibile su più piattaforme (JVM)

Affidabilità: Evitare il più possibile errori e crash di sistema (no pointer)

Sicurezza: Evitare programmi "maliziosi" (tipi, no pointer)

Semplicità: Accessibile a programmatori medi. Più semplice di C++.

Efficienza: importante, ma secondaria

Obiettivi di JAVA

Semplicità: - quasi tutto è un oggetto;

- no puntatori, no coercions automatiche, no funzioni, no ereditarietà multipla, no goto.

Portabilità: - codice mobile nella rete;

- bytecode interpreter.

Affidabilità e Sicurezza

- Sistema dei tipi più forte rispetto a C e C++
- controlli run time di tipo e accessi a memoria
- no puntatori espliciti (garbage collection)

JAVA: caratteristiche

Sintassi: simile a C/C++.

Oggetti: - formati da campi e metodi;

- allocati sull'heap, no run-time stack;

- **accessibili via puntatori** (assegnamento di puntatori)

Dynamic Lookup:

- comportamento simile a Smalltalk

- **Tipaggio statico** □ più efficiente di Smalltalk

- **Linkaggio dinamico**, interfacce □ meno efficiente di C++

JAVA: definizione di classi

Definire una classe implica:

- definire una collezione di campi con relativo tipo (instance variables)
- definire una collezione di metodi (instance methods)

I campi **descrivono lo stato** di ciascun oggetto della classe

I metodi possono **modificare lo stato e/o ritornare un valore**

Esempio: definizione di classe

Modificatori di visibilità

```
class Punto
{ private int xcoord; Definizione di campi
  protected void setX (int val)
    {xcoord=val;} Definizione di metodi
  public int getX() {return xcoord;}
  public void move(int dx)
    {xcoord=xcoord+dx;}
                                     costruttore
  Punto(int val) {xcoord=val;}
}
```

private

Una classe genera uno **scope** (ossia una porzione di programma che può accedere ad un set di nomi)

Un campo (o un metodo) dichiarato privato è accessibile **solo all'interno della classe**.

L'unità di programma che stabilisce lo scope è la classe, quindi **può accedere ai campi/metodi privati di qualsiasi oggetto di quella classe** (non solo di **this**).

public

Campi e metodi pubblici sono **accessibili** da **qualsunque contesto** che abbia un riferimento ad un oggetto di una classe.

Campi e metodi pubblici stabiliscono **l'interfaccia degli oggetti**, ossia i servizi che forniscono.

Buona abitudine: **campi SEMPRE privati**, un sottoinsieme piccoli di metodi pubblici (metodi ausiliari andrebbero dichiarati private o protected)

Protected & Package

Le classi vengono organizzate in **packages** (vedremo meglio)

Campi e metodi protected sono **accessibili nella classe e nelle sue sottoclassi (segreti di famiglia)**. E dalle classi dello stesso package

Visibilità **package** (default): una campo/metodo è accessibile da tutte le classi dello stesso package, ma non da sottoclassi fuori dal package.

JAVA: istanziazione di Oggetti

```
Punto myPoint = new Punto(3);
```

Dichiara una variabile punto e richiede esplicitamente la creazione una **nuova istanza** con coordinata 3.

Alla creazione viene eseguito il **codice del "metodo" costruttore.**

Java: tipi

Una definizione di classe introduce un **nuovo tipo** (tipo definito dall'utente).

Una volta definita una classe, il suo **nome** può essere utilizzato come un **normale tipo** per definire variabili:

```
Punto myPoint;
```

Un tipo è identificato unicamente dal nome: classi **strutturalmente uguali ma con nomi diversi** generano **tipi diversi!**

Equivalenza di tipi

Nome: Due tipi sono equivalenti se hanno lo stesso nome

- più astrazione
- usando nomi diversi per la stessa struttura dati si indicano dati logicamente distinti
- controllo contro usi impropri

Struttura: Due tipi sono equivalenti se hanno la stessa struttura

- più flessibilità

Esempio: classe Point

```
class Point
{ private int xcoord;
  protected void setX (int val)
    {xcoord=val;}
  public int getX(){return xcoord;}
  public void move(int dx)
    {xcoord=xcoord+dx;}

  Point(int val)
    {xcoord=val;}
}
```

**Identica alla definizione
di Punto, ma:**

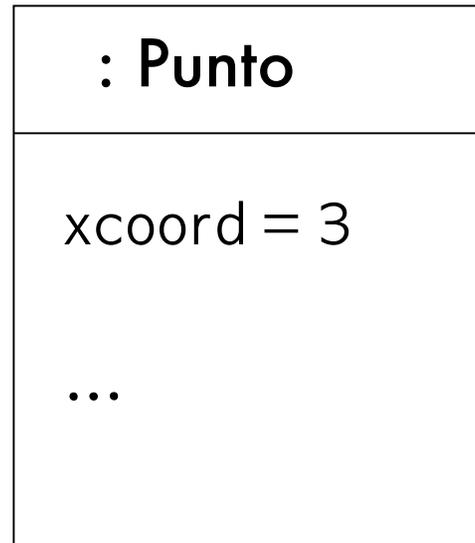
```
Punto mioPunto;
Point myPoint;
myPoint = mioPunto;
```

Genera un errore di tipo!

JAVA: variabili e Oggetti

Una variabile con un tipo classe **contiene un riferimento** ad un oggetto istanziato da quella classe.

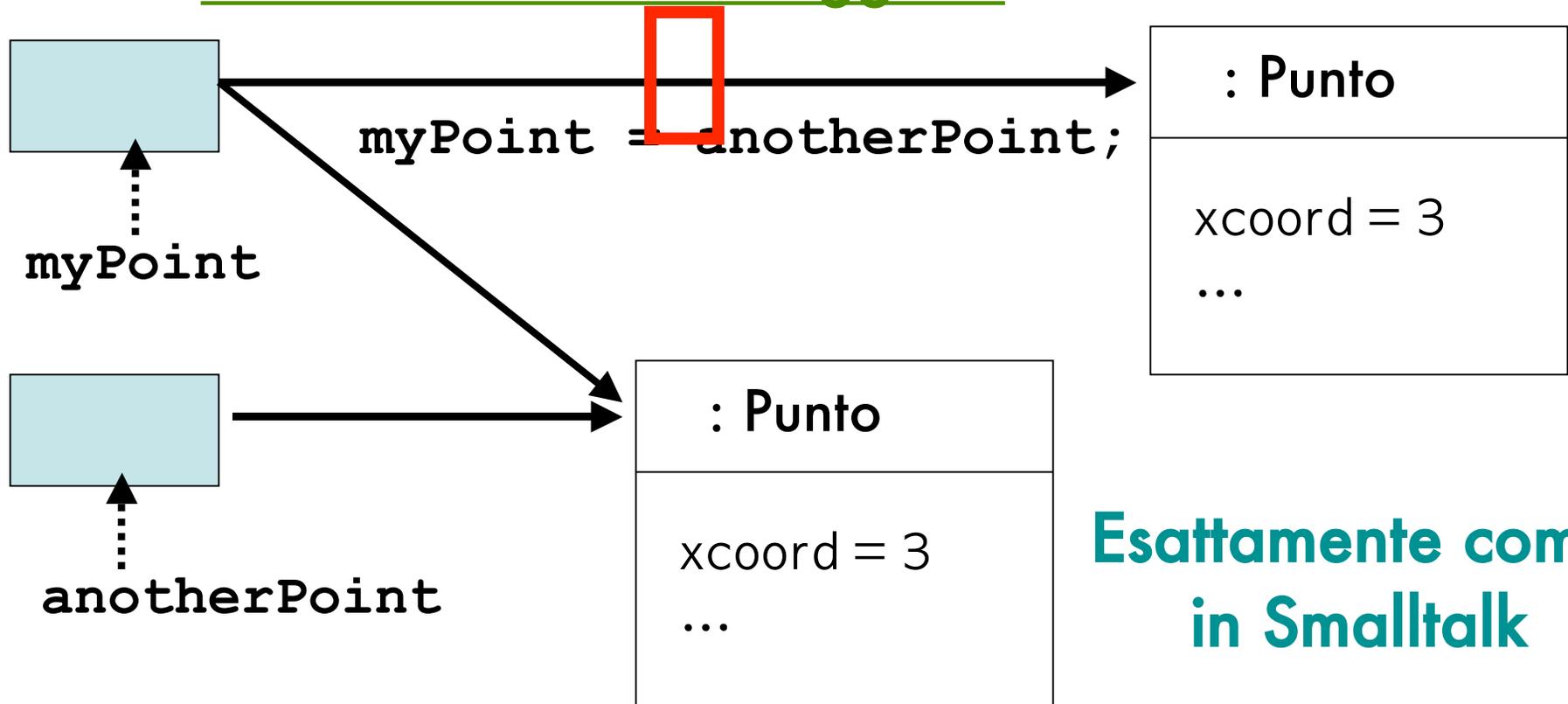
```
Punto myPoint = new Punto(3)
```



**Esattamente come
in Smalltalk**

Assegnamento

Assegnare una variabile significa assegnare un riferimento a un oggetto.



**Esattamente come
in Smalltalk**

Invocazione di metodi

Avendo un riferimento a un oggetto, è possibile invocare un metodo (o accedere a un campo) via **dot notation**:

Purché:

- il riferimento **non sia nullo**
- il metodo invocato **sia visibile** nel contesto in cui viene effettuata la chiamata.

```
Point myPoint = new Point(3);  
myPoint.move(2);  
myPoint.xcoord += 2; /*illegale */
```

Passaggio di Parametri

Sempre per VALORE (come in C).

Non ci sono modi espliciti di **(de)referenziare** le variabili (operatori * e &)

TUTTAVIA, passando il riferimento ad un oggetto, **il valore che state passando è un puntatore** (modifiche a quell'oggetto saranno visibili dal chiamante)

Sono oggetti (vedremo) anche gli array e altri tipi predefiniti (non i numeri).

Uguaglianza, identità e pointer

null: è il valore di default dato ad una variabile di tipo classe

Eventuali invocazioni di metodi su un oggetto null, genera una eccezione (`NullPointerException`)

==: confronta due riferimenti (`identità dell'oggetto`)

Per avere l'uguaglianza occorre (ri)definire un metodo `equals()`.