Java: Costruzione di oggetti

Marco Faella

Dip. Ing. Elettrica e Tecnologie dell'Informazione Università di Napoli "Federico II"

Corso di Linguaggi di Programmazione I



Elementi sintattici:

- I costruttori (metodi col nome della classe e senza tipo di ritorno)
- Invocazioni da un costruttore a un altro (this e super)
- I blocchi di inizializzazione (blocchi di codice anonimi nello scope di classe)
- Gli inizializzatori

```
public class A {
    private int n = <exp>;

    public A(int a) {
        n = a;
    }

    Blocco di inizializzazione

{
        n++;
    }
}
```



Invocazioni **esplicite** ad un altro costruttore:

- Un costruttore può chiamarne un altro della stessa classe usando la parola chiave this, oppure un costruttore della sua superclasse diretta usando la parola chiave super
- this e super, usati in questa accezione, devono comparire alla prima riga del costruttore, pena un errore di compilazione

- Attenzione: ricordate che this e super hanno anche altri significati:
 - this rappresenta il riferimento all'oggetto corrente
 - super si usa per riferirsi a un elemento (metodo o attributo) mascherato appartenente a una superclasse
 - · Ad esempio, per invocare la versione originale di un metodo di cui stiamo facendo l'overriding



Esempio 1:

L'utente della classe (client) sceglie se fornire un valore o usare quello di default

```
class A {
    private int size;
    public A() {
        this(1000); // invoca l'altro costruttore
    }
    public A(int n) {
        size = n;
    }
}
```



Esempio 2:

```
class A {
    public A() { this("Costruttore senza argomenti"); }
    public A(String msg) { System.out.println(msg); }
class B extends A {
    public B() { }
    public B(int n) {
         super("Valore: " + n);
```



Concatenazione dei costruttori

Chiamate implicite ad un altro costruttore:

- Se un costruttore **non inizia** con una chiamata ad un altro costruttore (*this* o *super*), il compilatore inserisce **automaticamente** una chiamata al costruttore senza argomenti della superclasse
 - Ovvero, inserisce l'istruzione "super()"
- In tal caso, se la superclasse non ha un costruttore senza argomenti, si verifica un errore di compilazione

Il meccanismo tramite il quale i costruttori possono invocarsi a vicenda prende il nome di concatenazione dei costruttori (constructor chaining)



Sequenza di inizializzazione di un oggetto:

- 1a) Se il costruttore inizia con super(...), passare al costruttore della superclasse (con risoluzione overloading)
- 1b) Se il costruttore inizia con this(...), passare al costruttore indicato (con risoluzione overloading)
- 1c) Se il costruttore non inizia né con super né con this, passare al costruttore senza argomenti della superclasse (a meno di non essere in Object)
- 2) Eseguire i blocchi di inizializzazione e gli inizializzatori degli attributi, nell'ordine in cui compaiono nel codice
- 3) Eseguire il resto del costruttore
- 4) Tornare al chiamante

Nota 1: un attributo privo di inizializzatore assume il valore di default del suo tipo

Nota 2: un blocco di inizializzazione o un inizializzatore non può fare riferimento a un attributo la cui dichiarazione segue lessicalmente quell'elemento

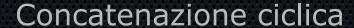


- Il compilatore controlla che la concatenazione **non sia ciclica**
- Infatti, se dei costruttori si chiamano a vicenda, siccome tali chiamate si trovano alla prima riga del rispettivo costruttore, e pertanto avvengono incondizionatamente, ci si trova in presenza di una mutua ricorsione non ben fondata, ovvero infinita
- Ad esempio, tentando di compilare la seguente classe:

```
class A {
  public A() { this(3); }
  public A(int i) { this(); }
}
```

si ottiene il seguente errore di compilazione:

```
A.java:3: recursive constructor invocation public A(int i) { this(); }
```





Per analizzare la concatenazione dei costruttori di una data gerarchia di classi, ed in particolare controllare che essa non sia ciclica, è possibile realizzare il seguente diagramma:

- 1) Creare un **grafo** con un **nodo** per ogni costruttore, compresi quelli impliciti
- 2) Aggiungere un **arco** da un nodo x ad un nodo y se il costruttore x chiama, esplicitamente o meno, il costruttore y
- 3) Il grafo ottenuto non deve presentare cicli



Determinare l'output del seguente programma

```
public class A {
    public A() {
       System.out.println("A()");
}
public class B extends A {
    public B() {
       this(0);
       System.out.println("B()");
    }
    public B(int n) {
       System.out.println("B(int)");
    }
    public static void main(String[] args) {
       new B();
    }
}
```



Determinare l'output del seguente programma

```
public class A {
   public A() {
      System.out.println("A()");
   }
   {
      System.out.println("Blocco 1");
   }
}
```

```
public class B extends A {
   public B() {
      this(0);
      System.out.println("B()");
      System.out.println("Blocco 2");
   }
   public B(int n) {
      System.out.println("B(int)");
   public static void main(String[] args)
      new B();
```



Errori di compilazione specifici della costruzione di oggetti:

- Concatenazione ciclica di costruttori
- Fare riferimento, in un inizializzatore o in un blocco di inizializzazione, a un attributo che segue lessicalmente (forward reference)
- Invocare implicitamente un costruttore della superclasse inesistente



• Dall'archivio di Linguaggi II: Esercizio d'esame 27/3/08, #4