Programmazione a Oggetti

Definizioni di Classi II

Sommario

- ***Costruzioni di oggetti***
- ***Campi e metodi di classe***
- ***Overloading***
- ***Istanziazione di oggetti***

Costruzione di un oggetto

Processo complesso che comprende varie fasi:

1. ***Allocare spazio*** *per ogni campo in ordine testuale* ***inizializza al valore di default*** *del tipo*
2. *Esegue inizializzatori per ogni campo*
3. *Esegue il* ***corpo del costruttore***
4. *Restituisce il* ***riferimento*** *alla memoria allocata*

Costruttori

Ogni classe ha sempre un costruttore associato

Gli oggetti vengono ***creati con una chiamata al costruttore*** *(eseguita con la* **`new`***)*

Default constructor*: se non viene definito nessun metodo costruttore, il compilatore ne genera uno.*

Il default constructor ***inizializza tutti i campi al loro valore di default****:*

`0` *(interi), "" (stringhe) ,* ***`null`*** *(oggetti)*

Costruttori: utilità

***Buona abitudine**: usare i costruttori permette di inizializzare i correttamente e garantire degli **invarianti di classe**.*

```
Class Date
{private int gg, mm, aa;
 public Date(int g,int m, int a)
 if (g<1 || g>31)
 {System.out.println "giorno illegale"; g=1;}
 if (m<1 || m>12)
  {System.out.println "mese illegale"; m=1;}
 aa=a; mm=m; gg=g;
}
```

Alternativa

***Inizializzatori**: inizializzare le variabili come in C, usando comandi del tipo:*

```
tipo variabile = espressione;
```

A patto che l'espressione abbia tipo corretto e non generi eccezioni:

```
Class Paperopoli{
  private int abitanti = 0;
  public Papero zioPaperone = new Papero();
  /* . . . */
}
```

La classe `Papero` *può avere riferimenti alla classe* `Paperopoli`

Attenzione all'ordine!

```
Class esempioPatologico{
 private int a=1;
 private int b = init_b();Inizializza b a 1
 private int c = init_c();Inizializza c a 0
 private int d=1;

 private int init_b()
   {return a;}

 private int init_c()
   {return d;}
  . . .
}
```

Attenzione ai loop!

```
Class C
{int i=1;
 D d = new D();
 int k=3
 /* . . . */
}
```

```
Class D
{ int a=0;
  C c = new C();
  int h=3;
  /* . . . */
 }
```

Il compilatore non da problemi, ma chiaramente questo genera un loop.

Definire più costruttori

*E' possibile definire **più costruttori** per una stessa classe (**overloading**).*

*Sarà eseguito il costruttore coerente (**tipi!**) con la chiamata;*

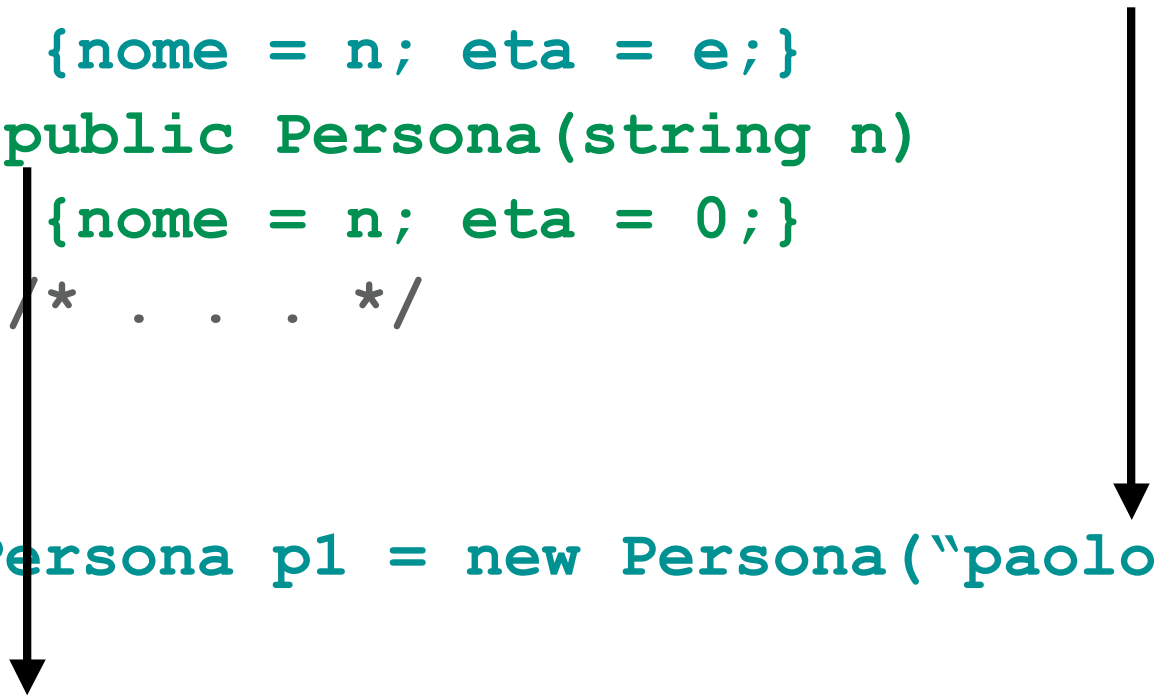
*Ciò dipenderà dal **numero e tipo dei parametri**.*

Costruttori: esempio

```
Class Persona
{private string nome;
 private int eta;
 public Persona(string n, int e)
   {nome = n; eta = e;}
 public Persona(string n)
   {nome = n; eta = 0;}
 /* . . . */
}

Persona p1 = new Persona("paolo rossi", 37);

Persona p2 = new Persona("mario bianchi")
```



Overloading

*Permette di definire **metodi diversi con lo stesso nome**.*

*La selezione (**overloading resolution**) avviene **attraverso il tipo*** *(parametri e tipo tornato)*

*Tutte le versioni di un metodo **devono differire** nel numero dei parametri o nel tipo o nell'ordine degli argomenti*

***a meno di subtyping!**

Overloading: esempio

```
Class testOverloading{
void prt(String s)
     {System.out.println(s);}
 void f1(char x){prt("f1:char");}
 void f1(float x){prt("f1:float");}
 void f2(int x){prt("f2:int")}
 void f2(float x){prt("f1:char")}
}
```

Quando vengono invocati i diversi corpi di metodo?

Overloading: esempio

```
testOverloading t = new
  testOverloading();
```

Chiamata	*Output*
`t.f1('c');`	`f1:char`
`t.f1(5);`	`f1:float`
`t.f2('c');`	`f2:int`
`t.f2(5);`	`f2:int`
`t.f2(5.1)`	`f2:float`

Bisogna ricordare: **`char ≤ int ≤ float`**

parleremo più in dettaglio del ***subtyping***

Costruttori: chiamate esplicite

Deve *essere* ***la prima istruzione*** *nel corpo di un costruttore.*

Solo all'interno di un costruttore*.*

Gli argomenti ***non*** *possono riferire a campi o metodi dell'oggetto*

```
Class Persona{
 private string nome;
 private int eta;
 public Persona(string n)
   {nome = n; eta = 0;}
 public Persona(string n, int e)
   {this(n); eta = e;}
/* . . . */
}
```

Costruttori: ossevazioni

I costruttori ***appartengono alla classe*** *e non agli oggetti.*

I costruttori ***non sono metodi*** *(non possono essere invocati sugli oggetti della classe).*

Posso definire ***più di un costruttore per una classe****.*

Oggetti e Riferimenti

Attenzione! *Definire una variabile* ***non crea un nuovo oggetto.***

```
Class C {public void m(){}}
```

Fuori della classe:

```
C[] ref;
```
Ok. Definizione di una variabile

```
ref.m();
```
Type error

```
ref[0].m();
```
NullPointerException

```
ref = new C[100]
```
Ok. Crea l'array di pointer

```
ref[0]=new C();
```
Ok. Crea un oggetto e mette il riferimento in pos. 0

```
ref[0].m();
```
Ok

Campi e metodi di classe

Si definiscono con la parola chiave `static`.

Esiste ***una sola copia*** *dei campi static nel descrittore della classe.*

Utilizzo*:*

Descrivere proprietà ***di tutta la classe*** *(esempio: il campo salario è uguale per tutte le istanze della classe Segretario) o* ***dell'insieme delle istanze*** *(esempio: contare il numero delle istanze create)*

Campi di classe: istanziazione

Possono essere inizializzati:

con costanti o riferimenti ad oggetti;

invocando ***metodi di classe****.*

Inizializzazione*:* ***al caricamento della classe****.*

Caricamento di una classe*:*

quando viene riferito uno static member

quando viene ***creata un'istanza***

L'inizializzazione dei campi **`static`** ***avviene una sola volta***

Esempio istanziazione

```
Class C{
 private static int a = init();
 public static int init()
  { System.out.println("init done");
    return 1;
  }

C c = new C();
C = new C();
```

Carica la classe e inizializza il Campo privato a. Stampa init done

Non genera nessun output, perché L'inizializzazione è già avvenuta

Metodi di classe

Si possono invocare su un oggetto, ***ma più tipicamente su una classe****.*

Non possono riferire a campi o metodi non static dell'oggetto ricevente (`this` ***è indefinito****).*

Sono, sostanzialmente ***funzioni*** *in senso classico.*

Esempi*:*

- *Sono metodi static, le tipiche* ***funzioni matematiche****:*

`Math.sin()`, `Math.exp()` *(*`Math` *è una classe!)*

- ***E' static il metodo*** `main()` ***(primo metodo invocato nell'esecuzione)***

Metodi di Classe: esempio

Errore: un metodo static non può riferire un metodo non static sullo stesso oggetto

```
Class C
{private int x = 1;
 public int m() {return x;}
 public static int s() {return m();}
 public static int s(C c)
  {return c.m()}
```

OK. Invoco un metodo non static di un altro oggetto