

Approfondimento delle classi

Fondamenti di architettura e programmazione
Corso di laurea in Comunicazione digitale

I costruttori

- Un costruttore è uno speciale metodo per creare nuovi oggetti di una data classe
- Nello scrivere un costruttore, ricordare che:
 - Deve avere lo stesso nome della classe
 - Non riporta alcun valore
 - Di conseguenza non ha alcun tipo di rientro, neanche **void**
 - Spesso inizializza i valori dei campi degli oggetti
- Non è sempre necessario definire un costruttore di una classe

AA2008/09
© M.A. Alberti

2

Fondamenti programmazione e
architettura
Classi 2

Costruttori di default

- Il costruttore di default non ha argomenti e crea l'oggetto inizializzando i campi d'istanza con valori di default

```
Class Roccia {  
    int i;  
}  
Roccia granito = new Roccia();
```

- Il costruttore di default può essere invocato anche se non è stato esplicitamente definito
 - Il membro d'istanza **i** di tipo **int** è inizializzato a 0

AA2008/09
© M.A. Alberti

3

Fondamenti programmazione e
architettura
Classi 2

Costruttori di default

- Se vengono definiti esplicitamente dei costruttori, con o senza parametri, allora **non** è possibile usare quello di default

```
Class Fiore {  
    Fiore (int i) {...};  
    Fiore (String s) {...};  
}  
Fiore rosa = new Fiore();
```

- Causa un errore in compilazione

AA2008/09
© M.A. Alberti

4

Fondamenti programmazione e
architettura
Classi 2

La parola chiave this

- Quando si invoca un metodo su un oggetto, il riferimento all'oggetto è un parametro implicito del metodo
- All'interno di un metodo per indicare questo parametro implicito si usa **this**
- La parola riservata **this** genera il riferimento all'oggetto corrente
- Spesso può essere sottintesa

AA2008/09
© M.A. Alberti

5

Fondamenti programmazione e
architettura
Classi 2

Esempi d'uso di this

- La parola **this** è necessaria in alcuni casi

```
Class Fiori {  
    int petali;  
    Fiori cresci() {  
        this.petali++; //non necessaria  
        return this; //necessaria  
    }  
}  
Fiori rosa = new Fiori();  
rosa.cresci().cresci().stampa();
```

AA2008/09
© M.A. Alberti

6

Fondamenti programmazione e
architettura
Classi 2

Chiamare costruttori da costruttori

- Può essere utile definire più di un costruttore e richiamarne uno dentro all'altro
 - Si usa allora `this()`
- `this(...)` con argomenti indica la chiamata al costruttore determinata da quegli argomenti
- Dopo aver usato `this` per costruire l'oggetto corrente, sempre `this` consente di riferirlo
- [Fiori.java](#) e [TestFiori.java](#)

AA2008/09
© M.A. Alberti

7

Fondamenti programmazione e
architettura
Classi 2

Metodi statici

- I metodi statici sono dichiarati mediante il modificatore `static`
- Il metodo `main` è `static` ed è invocato dal sistema senza creare un oggetto
- L'ordine dei modificatori può essere cambiato, ma per convenzione i modificatori di visibilità vengono prima
 - `public static void`

AA2008/09
© M.A. Alberti

8

Fondamenti programmazione e
architettura
Classi 2

Il modificatore static

- I metodi `statici` o anche `metodi di classe` possono essere invocati tramite il nome della classe ma possono essere invocati tramite il riferimento di un oggetto della classe
- Il modificatore `static` può anche essere applicato ai campi
 - Associa i campi e i metodi alla classe piuttosto che agli oggetti
- I metodi statici **possono** fare riferimento solo a campi dichiarati `static` e a variabili locali

AA2008/09
© M.A. Alberti

9

Fondamenti programmazione e
architettura
Classi 2

Significato di static

- Nei metodi dichiarati con il modificatore `static` non è possibile riferirsi all'oggetto corrente con `this`
 - Perché non c'è nessun oggetto corrente
 - **Non** si possono chiamare metodi non statici all'interno di metodi statici
 - Non ci si può riferire a membri d'istanza
- I metodi statici non si riferiscono a oggetti
- Hanno la semantica di una funzione globale

AA2008/09
© M.A. Alberti

10

Fondamenti programmazione e
architettura
Classi 2

Metodi statici

```
class Helper
{
    public static int triple (int num)
    {
        int result;
        result = num * 3;
        return result;
    }
}
```

Il metodo è `static` e viene invocato tramite il nome della classe

```
int valore = Helper.triple (5);
```

Può anche essere invocato tramite il riferimento a un oggetto (meglio evitare)

```
Helper x = new Helper();
int valore = x.triple (5);
```

Il riferimento al metodo statico è risolto durante la compilazione

AA2008/09
© M.A. Alberti

11

Fondamenti programmazione e
architettura
Classi 2

Campi statici

- I campi dichiarati con il modificatore `static` si chiamano anche `campi di classe`
- In generale ogni oggetto reclama lo spazio per i propri campi d'istanza
- Mentre esiste solo una copia di un campo `static`
- Lo spazio di memoria necessario per i campi statici è allocato nella `memoria statica` appena viene caricata la classe in cui è dichiarata

AA2008/09
© M.A. Alberti

12

Fondamenti programmazione e
architettura
Classi 2

Campi statici

- Gli oggetti di una stessa classe condividono l'accesso ai campi statici della classe
- Cambiare il valore di un campo statico in un oggetto implica il cambiamento per tutti gli oggetti della classe
- Metodi e campi statici spesso lavorano congiuntamente
- [CountInstances.java](#)
- [MyClass.java](#)

AA2008/09
© M.A. Alberti

13

Fondamenti programmazione e
architettura
Classi 2

Classi annidate

- Una classe oltre a contenere campi e metodi può contenere altre classi
- Una classe dichiarata all'interno di un'altra si chiama *classe annidata*



AA2008/09
© M.A. Alberti

14

Fondamenti programmazione e
architettura
Classi 2

Classi annidate

- Una classe annidata ha accesso ai membri (campi e metodi) della classe esterna, anche se sono dichiarate **private**
- Le due classi, esterna e annidata, quindi condividono le informazioni
- La classe annidata è protetta dalla classe in cui è definita dall'uso esterno

AA2008/09
© M.A. Alberti

15

Fondamenti programmazione e
architettura
Classi 2

Classi annidate

- Una classe annidata produce un file bytecode separato in compilazione
 - Es. La compilazione di una classe annidata dichiarata con il nome **Interna** in una classe di nome **Esterna** produce i file bytecode
Esterna.class
Esterna\$Interna.class
- Le classi annidate possono essere dichiarate statiche
 - Allora non possono fare riferimento a campi e metodi d'istanza come sempre
- Una classe annidata e non statica si chiama anche **classe interna**

AA2008/09
© M.A. Alberti

16

Fondamenti programmazione e
architettura
Classi 2

Inizializzazione dei campi

- I campi d'istanza sono spesso inizializzati dai costruttori
- Anche se il compilatore li inizializza con valori di default in ogni caso

```
public class Contatore {
    int i;
    Contatore() {
        i = 999;
    }
}
```
- Il membro **i** viene inizializzato a **0** all'atto della creazione di un oggetto alla chiamata del costruttore e poi a **999** alla esecuzione delle istruzioni del costruttore

AA2008/09
© M.A. Alberti

17

Fondamenti programmazione e
architettura
Classi 2

Ordine d'inizializzazione

- L'ordine d'inizializzazione dei campi è determinato dall'ordine in cui sono definiti nella classe.
- I campi sono inizializzati all'atto della chiamata del costruttore, **prima** che venga eseguito alcun metodo – anche prima di eseguire il codice del costruttore
- [OrdineInizializzazione.java](#)
 - I campi d'istanza della classe **Busta** sono dichiarati in ordine sparso, ma saranno inizializzati prima dell'esecuzione del costruttore dell'istanza, all'atto della sua invocazione.

AA2008/09
© M.A. Alberti

18

Fondamenti programmazione e
architettura
Classi 2

Inizializzazione campi statici

- I campi statici sono archiviati in un unico segmento di memoria
- Vengono inizializzati secondo l'ordine di definizione al momento della creazione del primo oggetto della classe o quando avviene il primo accesso statico
 - Al primo accesso alla classe il compilatore deve caricarla e così facendo inizializza i campi statici
- [InizializzazioneStatica.java](#)

AA2008/09
© M.A. Alberti

19

Fondamenti programmazione e
architettura
Classi 2

In sintesi - 1

- La JM alloca l'eseguibile di una classe, cercandolo nel path quando
 - si istanzia un oggetto della classe, o
 - si invoca per la prima volta un accesso statico alla classe (invocando un membro statico);
- Quando l'eseguibile viene allocato, prima dell'esecuzione, tutti i suoi campi statici vengono inizializzati.
 - L'inizializzazione statica avviene una volta sola.

AA2008/09
© M.A. Alberti

20

Fondamenti programmazione e
architettura
Classi 2

In sintesi - 2

- Quando si istanzia un nuovo oggetto della classe il costruttore cerca lo spazio sufficiente per contenerlo sullo heap.
 - I campi d'istanza vengono inizializzati ai valori neutri per i diversi tipi (**0** per i numeri, *blank* per i car, stringa vuota, **null** per riferimenti etc);
- Successivamente si eseguono le inizializzazioni indicate nelle dichiarazioni dei campi;
- Il costruttore viene quindi eseguito;

AA2008/09
© M.A. Alberti

21

Fondamenti programmazione e
architettura
Classi 2