Approfondimento delle classi

Programmazione
Corso di laurea in Informatica

I costruttori

- Un costruttore è uno speciale metodo per creare nuovi oggetto 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 delle variabili di istanza
- Non è sempre necessario definire un costruttore di una classe

Costruttori di default

- Il costruttore di default non ha argomenti e crea l'oggetto inizializzando i membri di 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`

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

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 `this` genera il riferimento all'oggetto corrente
- Spesso può essere sottointesa

Esempi d'uso di this

- La parola `this` è necessaria in alcuni casi

```
Class Fiori {
   int petali;
   Fiori cresci() {
        this.petali++;
        return this;
   }
}
```

- `this` restituisce il riferimento all'oggetto corrente

```
Fiori rosa = new Fiori();
rosa.cresci().cresci().stampa();
```

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`

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`

Il modificatore static

- I metodi ***statici*** o anche ***metodi di classe*** possono sono 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

Significato di static

- Nei metodi dichiarati con il modificatore `static` non è possibile riferirsi all'oggetto corrente con `this`
  - Perché non c'é 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

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 è risolto in modo statico durante la compilazione

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

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`

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*

Classi annidate

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

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 variabili e metodi d'istanza come sempre
- Una classe annidata e non statica si chiama anche ***classe interna***

Inizializzazione con il costruttore

- 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

Ordine d'inizializzazione

- L'ordine d'inizializzazione dei campi è determinato dall'ordine in cui sono definiti nella classe.
- I campi sono inizializzati **prima** che venga eseguito alcun metodo – anche prima di eseguire i costruttori.
- `OrdineInizializzazione.java`
  - I campi d'istanza della classe **Carta** sono dichiarati in ordine sparso, ma saranno inizializzati prima dell'esecuzione del costruttore dell'istanza, all'atto della sua invocazione.

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
  - In quest'ultimo caso il compilatore deve caricare la classe e così facendo inizializza i campi statici
- `InizializzazioneStatica.java`

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 vengo inizializzati.
  - L'inizializzazione statica avviene una volta sola.

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;