Approfondimento delle classi

Programmazione
Corso di laurea in Informatica

Riferimenti

- Il *riferimento* a un oggetto contiene l'indirizzo di memoria dell'oggetto
- Descriviamo l'indirizzo come un *puntatore* all'oggetto

```
PezzoScacchi alfiere = new PezzoScacchi();
```

Assegnamento

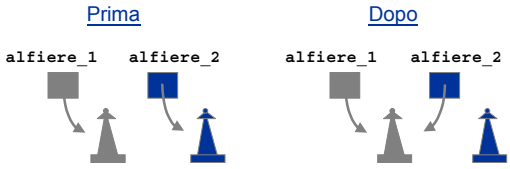
- L'istruzione di *assegnamento* effettua una copia di un valore e o archivia in una variabile
- Per i tipi primitivi:

```
num2 = num1;
```

Assegnamento

- Per variabili riferimenti a oggetti, l'istruzione di assegnamento copia l'indirizzo di memoria:

```
alfiere_2 = alfiere_1;
```



Alias

- Due o più riferimenti allo stesso oggetto si chiamano ***alias***
- Un oggetto e i suoi dati possono essere visti mediante variabili diverse, che lo referenziano
- Gli alias sono utili, ma devono essere usati con cautela
- Cambiare lo stato di un oggetto (le sue variabili) tramite un riferimento, causa il cambiamento anche per tutti gli altri ***alias***

Garbage Collection

- Quando un oggetto non ha più un riferimento non può più essere referenziato da alcun programma
- L'oggetto diventa inaccessibile e quindi inutile
  - si chiama *garbage* (spazzatura)
- Java effettua una raccolta automatica e periodica degli oggetti inutili (*garbage collection*)
  - Per rendere nuovamente utilizzabile per usi futuri lo spazio di memoria che l'oggetto occupava
- In altri linguaggi è responsabilità del programmatore effettuare il rilascio della memoria occupata da oggetti non referenziati e quindi inaccessibili

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 senza inizializzare i membri di d'istanza

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

- Il costruttore di default può essere invocato anche se non è stato esplicitamente definito

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 `this()`
- `this()` con argomenti indica la chiamata al costruttore determinata dagli 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 essere invocati tramite il nome della classe piuttosto che il nome di un oggetto
- Il modificatore `static` può anche essere applicato alle variabili
  - Associa le variabili o i metodi alla classe piuttosto che agli oggetti
- I metodi statici possono fare riferimento a variabili dichiarate `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;
}
```

Un metodo è `static`, viene invocato tramite il nome della classe

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

Può anche essere invocato tramite il riferimento a un oggetto

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

Il riferimento al metodo è risolto in modo statico durante la compilazione

Variabili statiche

- Le variabili dichiarate con il modificatore `static` si chiamano anche ***variabili di classe***
- In generale ogni oggetto reclama lo spazio per le proprie variabili d'istanza
- Mentre esiste solo una copia di una variabile `static`
- Lo spazio di memoria necessario per le variabili statiche è allocato appena viene caricata la classe in cui è dichiarata

Variabili statiche

- Gli oggetti di una stessa classe condividono l'accesso alle variabili statiche della classe
- Cambiare il valore di una variabile statica in un oggetto implica il cambiamento per tutti gli oggetti della classe
- Metodi e variabili statiche spesso lavorano congiuntamente

- `CountInstances.java`
- `MyClass.java`

Classi annidata

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

Classi annidata

- Una classe annidata ha accesso alle variabili e ai metodi della classe esterna, anche se sono dichiarate `private`
- La classe esterna e la classe 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
- Una classe annidata e non statica si chiama anche ***classe interna***

Inizializzazione con il costruttore

- I membri 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` e poi a `999` all'atto della creazione di un oggetto

Ordine d'inizializzazione

- L'ordine d'inizializzazione dei membri è determinato dall'ordine in cui sono definiti nella classe.
- I membri sono inizializzati prima che venga invocato alcun metodo – anche i costruttori
- `OrdineInizializzazione.java`
  - Alcuni oggetti sono dichiarati in ordine sparso, ma saranno instanziati tutti prima dell'invocazione el costruttore

Inizializzazione dati statici

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