Lezione 3

JAVA

Basi e Fondamenti

R-DeV
Revolution Development
learning@r-dev.it

Argomenti della lezione 3

- Array
- Stringhe
- Modificatori di accesso
- Costruzione di classi
- Getter e setter

R-DeV
Revolution Development
learning@r-dev.it

Array

Col termine *Array* (anche detto *vettore*) viene indicato, nella programmazione, un gruppo di elementi che viene dichiarato inizialmente specificandone la dimensione. Un Array può assegnare e restituire il valore di ciascun elemento accedendovi mediante un indice creato automaticamente.

Un Array si dichiara con il nome delle variabile seguito dalle parentesi quadre [] ed è a tutti gli effetti un oggetto, quindi deve essere inizializzato con la parola chiave `new`. Nella dichiarazione dobbiamo assegnare il numero massimo di argomenti da contenere, utilizzando il tipo di dato seguito dalla dimensione tra le parentesi quadre, come nell'esempio:

```
int x[] = new int[10];
```

In questo caso "`x[]`" potrà contenere al massimo 10 valori, possiamo assegnare e visualizzare i valori corrispondenti utilizzando la forma sintattica seguente:

```
x[0] = 10;
x[1] = 17;
```

Un array può contenere sia tipi primitivi che tipi reference.

java.util.Arrays

Questa classe contiene vari metodi per manipolare gli array (come l'ordinamento e la ricerca).
Possiamo, ad esempio, effettuare la ricerca binaria di un elemento in un array di interi invocando il seguente metodo:

```
static int binarySearch(int[] a, int key)
```

ci restituirà l'indice dell'elemento cercato se presente, -1 altrimenti.

Possiamo anche cercare in una porzione dell'array invocando un'altra versione dello stesso metodo:

```
public static int binarySearch(int[] a, int fromIndex, int toIndex, int
key)
```

Grazie all'overloading abbiamo varie versioni dello stesso metodo che si adattano al tipo di dato contenuto nel nostro array.
La classe Arrays mette a disposizioni molteplici metodi di utility: per il riempimento, la ricerca, l'ordinamento e la trasformazione in lista.
Per un elenco completo visita la documentazione ufficiale:
Arrays_java_doc

Array multidimensionali

E' anche possibile utilizzare Array multidimensionali in Java, ovvero con più di un indice, per esempio possiamo emulare una tabella grazie ad un Array Bidimensionale, impostando le colonne e le righe.

Esempio:

```
int x[][] = new int[2][2];
```

assegnamo i valori a `x[i][j]`:

```
for(int i = 0; i < 2; i++) {
     for(int j = 0; j < 2; j++) {
         x[i][j] = 7;
     }
}
```

Stringhe

Le stringhe sono oggetti che possono contenere un testo composto da caratteri alfanumerici e speciali, e possono essere creati come segue:

```
String s = new String (" abcdef ");
```

oppure ancora così:

```
String s = " abcdef ";
```

Vedremo quali sono le (sottili) differenze tra questi modi.
La cosa importante da capire è che le stringhe sono oggetti **immutabili**.

Stringhe

Per motivi di efficienza, poiché gli oggetti String occupano molta memoria, la JVM riserva un'area speciale di memoria ad essi: la **String constant pool**.
Quando il compilatore incontra un oggetto String inizializzato con un letterale, esso controlla che non sia già presente nel pool. Se è presente, allora il riferimento al nuovo letterale è diretto alla stringa esistente (la stringa esistente ha semplicemente un ulteriore riferimento), altrimenti lo aggiunge al pool.

Qual è quindi la (sottile) differenza tra i due enunciati?

```
String s = "abcdef";
String s = new String ("abcdef");
```

Metodi di String

Ecco alcuni metodi che offre la classe String:

- `char charAt(int index)` → tratta la stringa come un array di caratteri e restituisce il carattere in posizione `index`
- `int compareTo(String other)` → restituisce un valore < 0 se `this` precede alfabeticamente `other`, `0` se `this.equals(other) = true`, un valore > 0 se `other` precede alfabeticamente `this`.
- `String concat(String other)` → restituisce una nuova stringa concatenando `this` e `other`
- `boolean contains(CharSequence s)` → Restituisce `true` se e solo se `this` contiene `s`
- `int length()` → ritorna la lunghezza della stringa
- `String replace(char oldChar, char newChar)` → restituisce una stringa con le occorrenze di `oldChar` sostituite con `newChar`
- `String trim()` → elimina gli spazi iniziali e finali
- molti altri sulla documentazione ufficiale di String

NB: Ricordiamo che le stringhe sono immutabili, infatti e si può notare che tutti i metodi che "modificano" la stringa non sono void

Costruzione di Classi

Una classe si dichiara utilizzando la seguente struttura:

[class_modifier] **class** class_identifier [type_parameter_list]
[**extends** class_base] [**implements** interface]
{
	class_body;
}

I modificatori associati ad una classe possono essere i seguenti: **public**, **protected**, **private**, **abstract**, **static**, **final** e **strictfp**.

Se applichiamo il modificatore **public** alla dichiarazione di una classe, il file di codice sorgente che la contiene deve avere il suo stesso nome. Un file .java può contenere al massimo una classe **public** ma può contenere anche altre classi a patto che non siano **public.**

In java tutte le classi estendono implicitamente la classe Object.

Modificatori di accesso

L'accesso ad attributi e metodi della classe dall'esterno può essere gestito tramite i seguenti modificatori d'accesso:

Modificatore	Effetto
`public`	visibile da qualsiasi parte del programma
`private`	visibile solo dall'interno della classe stessa
`protected`	visibile solo dalle classi dello stesso package e dalle sottoclassi
default	visibile dallo stesso package e dalle sottoclassi presenti nello stesso package. È la visibilità assegnata di default se non viene specificato nulla.

Costruttore

Un costruttore di una classe viene definito con lo stesso nome della classe e una caratteristica che lo differenzia da un normale metodo è il fatto di non avere un tipo di ritorno.

Il costruttore viene chiamato, solo ed esclusivamente, quando viene istanziata la classe, quindi si accede al costruttore una sola volta per ogni istanza della classe.

Ogni classe può avere 0 o più costruttori che si devono differenziare per i parametri in input (overloading).

Se non viene specificato alcun costruttore il compilatore assegna il costruttore standard, senza argomenti, inizializzando gli attributi ai valori di default (0 per i valori numerici, false per i booleani, null per i riferimenti e valori di default dei tipi dichiarati per gli array). Se invece viene specificato un costruttore il costruttore standard non viene creato.

Il modificatore `static` non può essere usato per un costruttore.

Costruttore

Se all'interno del corpo di un costruttore si vuole invocare esplicitamente un altro costruttore esistente, bisogna sempre farlo alla prima riga, altrimenti si presenterà un errore di compilazione.

`super()` è l'invocazione al costruttore senza argomenti di input della classe madre; è inserita implicitamente come prima riga dal compilatore all'interno di ogni costruttore. Se lo si vuole inserire esplicitamente bisognerà farlo sempre alla prima riga, altrimenti si presenterà un errore di compilazione.

Il seguente codice è corretto ?

```
public class A {
    private int b;
    public A(int b) {
        this.b = b;
    }
}
```

```
public class B extends A {

}
```

Finalizzatore di Istanza

Un *finalizzatore* è un metodo di una classe che fornisce determinate operazioni da compiere prima che un'istanza di una classe, divenuta eleggibile per la sua distruzione in memoria dal parte del garbage collector, sia eliminata.

Un'operazione da implementare in un finalizzatore può essere quella di rilasciare una risorsa, come un file, precedentemente utilizzata.

La sintassi del metodo finalizzatore è la seguente:

```
protected void finalize() {
        body
}
```

Questo metodo viene ereditato dalla classe Object nella quale ha un'implementazione vuota ma è importante rispettarne la firma per effettuarne correttamente l'override.

La keyword this

Ogni oggetto creato ha una sorta di variabile, identificata dalla keyword this, che ha come valore un riferimento all'oggetto stesso.

Quando utilizziamo un metodo oppure una variabile membri di una classe il compilatore antepone implicitamente la keyword this.

Se all'interno di un metodo si deve utilizzare una variabile di istanza che ha lo stesso nome di una variabile locale dobbiamo utilizzare la sintassi this.varibile_istanza.

La classe Object

In Java ogni classe è una classe derivata, in modo diretto o indiretto, dalla classe **Object**. La classe appartiene al package `java.util`.
Ciascuna classe eredita automaticamente i metodi di Object:

`public final Class<?> getClass()`
restituisce la classe a runtime dell'istanza sulla quale è invocato

`protected void finalize() throws Throwable`
chiamato dal Garbage Collector su un oggetto quando determina che non ci sono più riferimenti a quell'oggetto

`public final void notify()`
se qualche thread è in attesa su questo oggetto, uno di essi viene scelto per essere risvegliato. Non è ridefinibile

`public final void notifyAll()`
risveglia tutti i thread che sono in attesa di quell'oggetto

La classe Object

`public final void wait() throws InterruptedException`
`public final void wait(long timeout) throws InterruptedException`
`public final void wait(long timeout, int nanos) throws InterruptedException`
fa in modo che il thread corrente attenda fino a quando un altro thread invoca il metodo `notify()` o il metodo `notifyAll()` per questo oggetto, oppure è trascorso il periodo di tempo specificato in input. Se invocato il primo metodo, senza parametri di ingresso, si assume che l'input sia 0.

`public String toString()`
restituisce una rappresentazione umanamente leggibile dell'istanza dell'oggetto su cui è invocato. L'implementazione nella classe Object è la seguente:

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

di conseguenza è fortemente raccomandato ridefinire il metodo `toString()` negli oggetti creati.

La classe Object - hashCode()

`public int hashCode()`
restituisce un codice hash per l'istanza su cui è invocato. E' un metodo di supporto alle tabelle hash come quelle fornite da HashMap.

Il contratto generale di hashCode è:

- Ogni volta che viene richiamato sullo stesso oggetto più di una volta durante l'esecuzione di un'applicazione Java, il metodo `hashCode` deve restituire costantemente lo stesso numero intero, a condizione che non venga modificata alcuna informazione utilizzata nel metodo.
- Se due oggetti sono uguali in base al metodo `equals`, la chiamata del metodo `hashCode` su ciascuno dei due oggetti deve produrre lo stesso risultato, ma non è necessario il contrario, tuttavia bisogna essere consapevoli che la produzione di risultati distinti per oggetti diversi può migliorare le prestazioni delle tabelle hash.

Se viene ridefinito il metodo `hashcode` è buona norma ridefinire anche `equals` in modo da rispettare il contratto di coerenza tra i due metodi.

La classe Object - equals()

`public boolean equals()`
restituisce true se due oggetti sono uguali, false altrimenti.
L'implementazione del metodo deve rispettare le seguenti proprietà:

- RIFLESSIVA: per ogni oggetto non nullo `x.equals(x)` deve restituire `true`
- SIMMETRIA: per ogni coppia di oggetti non nulli `x.equals(y)` restituisce `true` se e solo se `y.equals(x) = true`
- TRANSITIVA: dati tre oggetti non nulli se `x.equals(y) = true` e `y.equals(z) = true` allora `z.equals(x)` deve restituire `true`
- CONSISTENZA: per ogni coppia di oggetti non nulli le invocazioni ripetitive di `x.equals(y)` o `y.equals(x)` devono restituire il medesimo risultato a patto che durante l'esecuzione non cambino le informazioni degli oggetti
- per ogni riferimento x non nullo `x.equals(null)` deve restituire `false`

Se il metodo non viene ridefinito per comportamento standard confronterà i riferimenti, quindi restituirà true se e solo se le variabili punteranno allo stesso oggetto `(x == y)`.
Ridefinire `equals` comporta ridefinire anche `hashcode` in modo da non violare il contratto e mantenere la coerenza tra i due metodi.

La classe Object - clone()

`protected Object clone()  throws CloneNotSupportedException`
Crea e restituisce una copia dell'oggetto su cui è invocato. Il significato preciso di "copia" dipende dall'implementazione. L'intento generale è che, per qualsiasi oggetto x, l'espressione:
`x.clone() != x` deve essere true, ovvero che l'oggetto creato deve avere un riferimento diverso dall'oggetto copiato.

Per convenzione, l'oggetto restituito da questo metodo dovrebbe essere completamente indipendente dall'oggetto copiato. Per ottenere questa indipendenza, potrebbe essere necessario modificare uno o più campi dell'oggetto restituito da `super.clone` prima di restituirlo effettuando una copia profonda dei campi, ovvero invocando il metodo clone su qualsiasi attributo di tipo non primitivo presente tra gli attributi. Se una classe contiene solo campi primitivi o riferimenti a oggetti immutabili si può evitare di ridefinire il metodo `clone`.

Getter e Setter

Getter : metodo che restituisce il valore di un attributo di una classe

Setter : metodo che valorizza un attributo di una classe

Esempio:

```
public class Esempio {

     private String variabile;

     public String getVariabile() {
          return this.variabile;
     }

     public void setVariabile(String variabile) {
          this.variabile = variabile;
     }
}
```

Getter e Setter

La progettazione di metodi get/set consente di soddisfare il principio dell'incapsulamento perché l'integrità dei dati di un oggetto è preservata da accessi arbitrari e potenzialmente pericolosi da parte di client esterni, i quali potranno accedere a tali dati esclusivamente impiegando metodi publici definiti ad-hoc.

Da questo punto di vista, quindi, una classe così progettata può essere vista come una sorta di black box che incapsula, protegge e nasconde i dati e i suoi dettagli implementativi.

Membri statici

Una classe può avere, oltre che dei *membri di istanza,* anche dei membri definiti come *membri statici*, che esistono indipendentemente dall'oggetto che ne è istanza.

In pratica possiamo affermare che mentre un membro di istanza è un membro appartenente a un oggetto (cioè a un'istanza di un tipo classe) un membro statico è un membro appartenente a un tipo classe. Questo vuol dire che mentre ogni oggetto di una determinata classe avrà una sua copia privata dei membri di istanza, i membri statici saranno invece condivisi da tutto gli oggetti istanze della classe.

Per dichiarare un membro statico si utilizza il modificatore **static**

L'accesso ad un membro statico **s** di una classe **C** può essere effettuato direttamente utilizzando il nome del tipo es. C.s oppure tramite una sua istanza (considerando o istanza di C, o.s) ma è preferibile accedere tramite il nome della classe.

Membri statici

Un metodo statico non può invocare un metodo di istanza o un campo di istanza, cioè non può accedere a membri che non siano statici in quanto i membri di istanza esistono solo se esiste il loro oggetto. Un metodo di istanza può accedere senza problemi ai membri statici.

Naturalmente non è possibile utilizzare all'interno dei metodi statici le keywords this e super.

Classi interne

Una classe si può dichiarare all'interno di un'altra classe, in questo caso viene definita classe interna o inner class.

Le inner class devono rispettare le seguenti regole:

- non è possibile dichiarare inizializzatori statici;
- non è possibile dichiarare membri statici (si possono dichiarare solo se costanti);
- non si può utilizzare this per fare riferimento ai membri di istanza della classe che la contiene (ma si possono referenziare in via diretta indipendentemente dalla visibilità);
- non può far riferimento, se dichiarata in un metodo statico, ai membri di istanza della classe che la contiene.

Le inner class possono essere dichiara come membro o all'interno di un qualsiasi blocco di codice (come un metodo) in questo caso vengono definite anche classi annidate (nested class). Nel caso in cui vengono dichiarate come membri statici non sono definite classi interne.

Classi interne

```
public class TopLevelClass {
        private String attr1;

        public class Inner {
                public void print() {
                        System.out.println(attr1); // può accedere a tutti gli attributi della classe che la
                                                   // contiene indipendentemente dalla visibilità
                }
        }
}
```

Se public il client potrà istanziarla solo da un'istanza della classe che la contiene. Sarà sempre possibile instanziarla internamente alla classe che la dichiara.

```
TopLevelClass tlc = new TopLevelClass();
var i = tlc.new Inner();
```

Classi interne

```
public class TopLevelClass {
	private String attr1;

	public static class Nested {
		public void print() {
			// non può accedere a membri non statici della classe che la contiene
		}
	}
}
```

Se public il client potrà istanziarla indipendentemente dall"istanza della classe che la contiene. Sarà sempre possibile istanziarla all'interno della classe che la dichiara.

```
var n = new TopLevelClass.Nested();
```

Classi interne

```
public class TopLevelClass {
        private String attr1;

        public void print() {
                class Inner {
                        public void display() {
                                System.out.println(attr1);
                        }
                }
                Inner n = new Inner();
                m.display();
        }
}
```

Le classi locali potranno essere utilizzate solo localmente al blocco di codice in cui sono definite

Enumerazioni

Un tipo enumerato è un costrutto sintattico messo a disposizione da Java che consente di creare un nuovo tipo di dato composto (è un tipo speciale del tipo classe).

[enum_modifier] **enum** identifier [implements interface] {
    body
}

Il corpo di un'enumerazione contiene la dichiarazione di apposite costanti che ne rappresentano i suoi membri fondamentali.

Essendo un tipo speciale di classe è possibile dichiarare, oltre alle constanti di enumerazione, anche membri proprio di una classe, inizializzatori e costruttori di istanza.

Enumerazioni

Le enum sono implicitamente final quindi non possono essere estese;

Sono implicitamente static se annidate come membro di un altro tipo;

non si può creare un'istanza di un'enum con l'operatore new;

una variabile di un determinato tipo di enumerazione può aver assegnato come valore solo una delle costanti dichiarate nel suo corpo;

le costanti di enumerazione devono sempre essere scritte come prima istruzione e l'ultima costante deve terminare con punto e virgola se sono presenti altri membri

Enumerazioni

```
enum SO {
        WINDOWS(“10”),
        LINUX(“Ubuntu”),
        MAC(“Lion”);

        private final String name;

        SO(String n) { this.name = n; }

        public String getName() { return this.name; }
}
```

Fine lezione 3

R-DeV
Revolution Development
learning@r-dev.it