OOP in JAVA

Programmazione orientata agli Oggetti

Giulio Angiani
I.I.S. "Blaise Pascal" - Reggio Emilia

OOP
Object Oriented Programming

Modificatori final e static

Cosa si intende per **final** e **static**

Sono due modificatori che assegnano alcune proprietà ad attributi e metodi delle classi

final viene utilizzato per implementare sostanzialmente il concetto di valore costante

static per implementare il concetto di variabile di classe o di metodo di classe

Modificatore final

Può essere applicato ad attributi, a metodi, a classi

Se applicato ad attributi

- permette la modifica di un attributo solo nei metodi **costruttore**
- impedisce che il valore di un attributo possa essere modificato dopo l'instanziazione di un oggetto
- se un attributo definito final non ha un valore di default **obbliga** la valorizzazione nel costruttore

con questo codice...

JAVA

```java
public class Poligono {
    final int numerolati;

    public Poligono(int n) {

    }
}
```

il compilatore mi segnala l'errore...

OUTPUT

```
error: variable numerolati might not have been initialized
```

Modificatore final

Posso quindi inizializzare a livello di classe

JAVA
```java
final int numerolati = 5;
```

oppure nel costruttore

JAVA
```java
public class Poligono {
    final int numerolati;

    public Poligono(int n) {
        this.numerolati = n;
    }
}
```

Questo valore **non** sarà più modificabile da alcun metodo

JAVA
```java
public void setNumerolati(int n) {
    this.numerolati = n;
}
```

OUTPUT
```
error: cannot assign a value to final variable numerolati
```

Modificatore final

Se applicato ad metodi

- impedisce che un metodo possa essere sovrascritto da classi derivate (vedremo nel capitolo ereditarietà)

Se applicato a classi

- impedisce che una classe possa essere estesa da un'altra (vedremo nel capitolo ereditarietà)

Il concetto di **final** è appunto di descrivere un attributo, classe o metodo come finale, completo e quindi correttamente **non modificabile** da alcun altro elemento.

Modificatore static

Può essere applicato ad attributi, a metodi, a classi

Cosa significa **static** ? Il senso cambia in funzione del contesto:

Se si parla di un attributo

- indica un attributo di classe e non di oggetto
- è **condiviso** fra tutte le istanze di una certa classe
- può essere invocato o acceduto con la notazione puntata da un oggetto
- il valore è condiviso dagli altri oggetti

Modificatore static

Esempio:

se ho definito un attributo così...

JAVA

```java
public static int numeropoligoni = 0;
```

posso invocarlo da un oggetto

JAVA

```java
Poligono p1 = new Poligono(5);
Poligono p2 = new Poligono(5);
System.out.println("p1.numeropoligoni: " + p1.numeropoligoni);
System.out.println("p2.numeropoligoni: " + p2.numeropoligoni);
// questa operazione modifica il valore di un attributo statico
// e viene recepita da tutti gli oggetti di classe
p1.numeropoligoni++;
System.out.println("p1.numeropoligoni: " + p1.numeropoligoni);
System.out.println("p2.numeropoligoni: " + p2.numeropoligoni);
```

OUTPUT

```
p1.numeropoligoni: 0
p2.numeropoligoni: 0
p1.numeropoligoni: 1
p2.numeropoligoni: 1
```

Modificatore static

Un attributo o metodo **static** non può essere acceduto con la notazione **this.** all'interno della classe.

Va utilizzata la notazione **Nomeclasse.**

esempio:

JAVA

```
public static void incPoligoni() {
    // this.numeropoligoni++;      // ERRORE: non-static variable this cannot be referenced from a static context
    Poligono.numeropoligoni++;  // corretta
}
```

Modificatore static

Il metodo statico può invece essere invocato anche da un oggetto fuori della classe, anche se si preferisce la notazione con nomeclasse

JAVA

```java
System.out.println("Poligono.numeropoligoni: " + Poligono.numeropoligoni);
Poligono p1 = new Poligono(5);
Poligono p2 = new Poligono(5);
System.out.println("p1.numeropoligoni: " + p1.numeropoligoni);
System.out.println("p2.numeropoligoni: " + p2.numeropoligoni);
p1.incPoligoni();
System.out.println("p1.numeropoligoni: " + p1.numeropoligoni);
System.out.println("p2.numeropoligoni: " + p2.numeropoligoni);
Poligono.incPoligoni();
System.out.println("p1.numeropoligoni: " + p1.numeropoligoni);
System.out.println("p2.numeropoligoni: " + p2.numeropoligoni);
System.out.println("Poligono.numeropoligoni: " + Poligono.numeropoligoni);
```

OUTPUT

```
Poligono.numeropoligoni: 0
p1.numeropoligoni: 0
p2.numeropoligoni: 0
p1.numeropoligoni: 1
p2.numeropoligoni: 1
p1.numeropoligoni: 2
p2.numeropoligoni: 2
Poligono.numeropoligoni: 2
```

Modificatore static - Esercizio

Progettare la classe Utente in modo che il metodo statico **getNumeroUtenti()** restituisca il numero di oggetti instanziati in un certo momento

es:

JAVA

```
System.out.println("Utenti istanziati: " + Utente.getNumeroUtenti());
```

OUTPUT

```
Utenti istanziati: 5
```

(10 minuti di tempo...)

Modificatore static - Soluzione

JAVA (CLASSE)

```java
public class Utente {
    private static int numeroutenti = 0;

    public Utente() {
        Utente.numeroutenti++;
    }

    public static int getNumeroUtenti() {
        return Utente.numeroutenti;
    }
}
```

JAVA (MAIN)

```java
Utente u1 = new Utente();
Utente u2 = new Utente();
System.out.println("Utenti instanziati: "+Utente.getNumeroUtenti());
Utente u3 = new Utente();
System.out.println("Utenti instanziati: "+Utente.getNumeroUtenti());
```

OUTPUT

```
Utenti instanziati: 2
Utenti instanziati: 3
```

Giulio Angiani
I.I.S. "Blaise Pascal" - Reggio Emilia