

## Modificatori

Sono Keywords che danno info al compilatore sulle nature del codice.  
Prono essere d'accesso o di "qualificazione".

Modificatori di accesso → Public, Protected, Private (se nessuno dei tre è presente, Default è Protected)  
→ usati per classi, attributi e metodi

Generalità → Modificatore d'accesso per :

- Classi, regola dove è possibile usare il suo nome;
- Attributi, regola quali espressioni del tipo exp. x sono valide
- Metodi, regole quali invocazioni possono usare quel metodo

Modificatore public → Permessi, accesso non limitato

Modificatore private → Restrizivo, applicabile ad attributi, metodi, costruttori, classi interne (non a classi Top)

Se appartengono ad A, sono accessibili dalle espressioni di A oppure se il rifer. è di tipo A

```

public class Complesso {
    private double reale, immag;

    public Complesso(double r, double i) {
        reale=r; immag=i;
    }
    public Complesso add(Complesso c) {
        return new Complesso(reale + c.reale,
                              immag + c.immag);
    }
}

public class Cliente {
    void usali() {
        Complesso c1 = new Complesso(1, 2);
        Complesso c2 = new Complesso(3, 4);
        Complesso c3 = c1.add(c2);
        double d = c3.reale; // Err. di comp.
    }
}

```

Attributo privato di Complesso

Gli attributi privati sono nascosti anche alle sottoclassi

```

class Complesso {
    private double reale, immag;
    ...
}

class SubComplesso extends Complesso {
    SubComplesso(double r, double i) {
        reale = r;                                // Err. di comp.
    }
}

```

La classe SubComplesso eredita gli attributi della superclasse,  
MA quegli attributi possono essere usati solo dal codice della  
classe Complesso

In questo caso interviene il **scope**  
privato

```

class A {
    private int n;
    public void foo(B b) {
        System.out.println(b.n); // Err. di comp.
    }
}

class B extends A {
    private int n; Masconde quella d'A
}

```

La visibilità dipende anche dal tipo dichiarato del riferimento  
usato per accedere

Qui, nessuno dei due metodi f è visibile:

```

class A {
    private void f() { }
    public void g(B b) {
        b.f(); // Errore di compilazione
    }
}

class B extends A {
    private void f() { }
}

```

discorso analogo a prima, con  
la differenza che adesso è il  
metodo ad essere privato

## Modificatore protected

Accesso permesso al codice dello stesso pacchetto e a tutte le sottoclassi in pacchetti solo per ereditarietà.

Una sottoclasse accede ad un **protected** solo per riferimento ad un oggetto del suo tipo o di un sottotipo

```
package p1;  
  
public class A {  
    protected int x = 7;  
}
```

```
package p2;  
import p1.A;  
  
public class B extends A {  
    public void test() {  
        System.out.println(x); // Ok: x è ereditato  
        // sinonimo di this.x  
        A a = new A();  
        System.out.println(a.x); // Errore di comp.  
        // a non è di tipo B, né di sottotipo di B  
        B b = new B();  
        System.out.println(b.x); // Ok: accesso attraverso  
        // riferimento di tipo B  
    }  
}
```

## Altri modificatori

Final → applicabile a **classi**, **metodi** e **variabili**

↳ varia in base al contesto (in generale, un elemento **Final** non può essere modificato)

Classe → non può avere sotto classi;

Varabile → può essere solo mutabili

Metodo → non permette overriding in una sottoclasse

Un riferimento Final non può essere riassegnato, ma può modificare il contenuto dell'oggetto a cui fa riferimento

```
final Impiegato luca = new Impiegato("Luca", 1500);
luca = new Impiegato("Luca", 1600); // Err. di comp.
luca.setSalario(1600);           // OK
```

Stesso discorso per l'array final

```
final int[] numeri = new int[10];
numeri = new int[20]; // Err. di comp.
numeri[0] = 77;     // OK
```

Modificatore abstract → Applicabile a classi e a metodi

↳ non hanno un corpo

La classe abstract è tale se :

- Contiene almeno un metodo abstract, oppure;
- Eredita " " " " per il quale non ha una realizzazione, oppure
- Declara ma non fornisce metodi per un interface

Ogni classe può essere abstract, anche senza metodi, ma il compilatore non le istruisce  
↳ opposto di final, perché final non può avere sottoclassi mentre queste nasce per questo rego

Attributi static → Applicabili ad attributi, metodi e a blocchi esterni ai metodi

Attributi vengono inizializzati quando la classe viene caricata in memoria.

... accaduti con la dot-notation

- partendo da un riferimento ad un'istanza di quelle classe;
- " del nome stesso della classe

```
System.out.println(Ecstatic.x);
Ecstatic e = new Ecstatic();
e.x = 100;
Ecstatic.x = 100;
```

## Metodi static

- Appartengono alla classe e non alle singole istanze
- Si possono invocare a partire dal nome della classe:

```
class Test {
    public static void f() { ... }
}
...
Test.f();
```

- Non posseggono il riferimento this
- Esempio: il metodo main
- Hanno binding statico e non possono essere sovrascritti (overridden)

## Blocchi di inizializzazione static

- Una classe può contenere blocchi di codice marcati static
- Tali blocchi sono eseguiti una sola volta, nell'ordine in cui compaiono, quando la classe viene caricata in memoria

```
public class EsempioStatic {
    private static double d=1.23;

    static {
        System.out.println("Primo blocco static: d=" + d++);
    }

    public static void main(String[] args) {
        System.out.println("main: d=" + d++);
    }

    static {
        System.out.println("Secondo blocco static: d=" + d++);
    }
}
```

# Questionario

1

Quale dei seguenti frammenti viene correttamente compilato e stampa "Uguale" in esecuzione?

A. 

```
Integer x = new Integer(100);
Integer y = new Integer(100);
if (x == y) {
    System.out.println("Uguale");
}
```

B. 

```
Integer x = Integer.valueOf(100);
Integer y = new Integer(100);
if (x == y) {
    System.out.println("Uguale");
}
```

C. 

```
int x=100; float y=100.0F;
if (x == y) {
    System.out.println("Uguale");
}
```

D. 

```
String x = new String("100");
String y = new String("100");
if (x == y) {
    System.out.println("Uguale");
}
```

E. 

```
String x = "100";
String y = "100";
if (x == y) {
    System.out.println("Uguale");
}
```

C → viene fatto il confronto delle parti intere

E → String constant pool che interviene

2

Quali delle seguenti dichiarazioni sono *illegali* in una classe A?

- A. `private A() { }`
- B. `public final A() { }`
- C. `final abstract int f();`
- D. `final int g() { return 0; }`
- E. `abstract double d;`
- F. `abstract static double getValue();`

B → Un costruttore non ha senso che sia final

C → abstract e final non stanno bene insieme

E → un abstract non può essere abstract

F → abstract e static non stanno bene insieme  
non può essere overrided

3

Quale delle seguenti affermazioni è vera?

- A. Una classe abstract non può avere metodi final.
- B.** Una classe final non può avere metodi abstract.

final non può essere sperimentato e quindi non avrà abstract

4

Qual è la minima modifica che rende il seguente codice compilabile?

```
1. final class Aaa {  
2.     int xxx;  
3.     void yyy() { xxx=1; }  
4. }  
5.  
6. class Bbb extends Aaa {  
7.     final Aaa fref = new Aaa();  
8.     final void yyy() {  
9.         System.out.println(  
10.            "In yyy()");  
11.         fref.xxx = 12345;  
12.     }  
13. }
```

- A.** Alla linea 1, rimuovere final.
- B. Alla linea 7, rimuovere final.
- C. Rimuovere la linea 11.
- D. Alle linee 1 e 7, rimuovere final.
- E. Nessuna modifica è necessaria.

final non può essere sperimentato

5

Riguardo al codice seguente, quale affermazione è vera?

```
1. class Roba {  
2.     static int x = 10;  
3.     static { x += 5; }  
4.  
5.     public static void main(String[] args) {  
6.         System.out.println("x=" + x);  
7.     }  
8.  
9.     static { x /= 5; }  
10. }
```

- A. Le linee 3 e 9 non sono compilate, poiché mancano i nomi di metodi e i tipi di ritorno.
- B. La linea 9 non è compilata, poiché si può avere solo un blocco top-level static.
- C. Il codice viene compilato e l'esecuzione produce x=10.
- D. Il codice viene compilato e l'esecuzione produce x=15.
- E.** Il codice viene compilato e l'esecuzione produce x=3.

Per come funzionano i blocchi statici, il risultato è 3

