

Metodi statici

I metodi statici sono metodi della classe e non delle sue istanze

Per i metodi statici non c'è il concetto di oggetto implicito
- non c'è il riferimento this

In un metodo statico possiamo usare

- Campi statici
- altri metodi statici
- oggetti, anche della stessa classe, e potranno possedere un riferimento a tali oggetti

Un metodo statico può restituire oggetti di cui ha il riferimento o che esso stesso crea

Per chiamare un metodo statico:

NomeDellaClasse. nomeDelMetodo()

↑
forma più generale

Se da dentro una classe voglio chiamare un metodo static, posso farlo senza dover specificare il nome della classe stessa

Metodi static incontrati nelle lezioni precedenti:

Meth. sqrt(...)
 ↑ double

public static void main(...)
 ↑

class Studente {

String nome;
double media;
int matricola;
static int contatore;

Studente(String n) {

 nome = n;
 matricola = ++contatore;

}

```

    }
    double getMedia() {
        return media;
    }
}

static int quantiStudenti() {
    return countore;
}
:

public static void main(String[] args) {
    Studente s1 = new Studente("Marco");
    Studente s2 = new Studente("Luigi");
    int quanti = Studente.quantiStudenti();
    // oppure semplicemente
    // int quanti = quantiStudenti();
    // perche' sono dentro la stessa classe
    :
}

}

```

static perché
 voglio chiamarla
 e prescindere dall'
 esistenza di oggetti

I package

Un package è un insieme di classi tra loro correlate

I package ci aiutano a

- raggruppare classi tra loro correlate
- evitare conflitti nei nomi delle classi
- controllare in modo fine l'eccesso delle classi e ai loro membri

Per mettere una classe dentro un package :

```
package miopackage;
class A {
    :
}
```

prima istruzione
del file sorgente
A.java

Il nome completo di una classe è
nomedelpackage . NomeDellaClasse

(detto anche nome "completamente qualificato")

Nel caso precedente: miopackage . A

Posso mettere nel solo package anche se uso più file sorgenti

```
package miopackage;  
class B {  
    :  
}
```

B.java

A e B sono entrambe in miopackage

Il nome di un package può essere composto da più parti separate da ":".

```
package abc.def;
```

```
class X {  
    :  
}
```

Allora il nome completamente qualificato di X è abc.def.X

Se fatto di avere dei nomi completamente qualificati ci aiuta a evitare conflitti

`mioPackage.A` è diverso da
`pack1.A`

```
package email;  
class Message {  
};
```

```
package cs;  
class Message {  
};
```

Per far funzionare bene tutto (essenza di conflitti) è necessario che i nomi dei package siano diversi (possibilmente univoci)

Convenzione: quando un'organizzazione scrive del codice usa come parte iniziale dei nomi dei package il proprio nome di dominio al contrario

Esempi:

`com.ibm.db...`

`it.unipi.email...`

`it.unipi.dii.db...`

Le classi che non vengono messe esplicitamente in un package vengono a finire nel package di default (il package senza nome).

Regola: mettere sempre le classi dentro package

Le classi che sono nel package senza nome non possono essere usate da altro codice contenuto in package veri e propri.

Tutte le librerie Java sono strutturate in package

- sia le classi di sistema
- sia le classi prodotte da terzi

Alcuni package di sistema:

java.lang Contiene le classi fondamentali del linguaggio per esempio String, Object

java.io Contiene classi dedicate a ingresso/uscita

java.net Contiene classi utili a

java.net
contiene classi utili a
accedere alla rete

com.ibm.sql....

com.oracle...

Una classe può usare tutte le altre
classi del suo stesso package
e le classi di altri package che sono
dichiarate public.

Una top-level class può essere
public
o
non public (assenza di modificatore)

A

(Nota: per una top-level class le
possibilità sono solo due

mentre

package esempio;
class A {

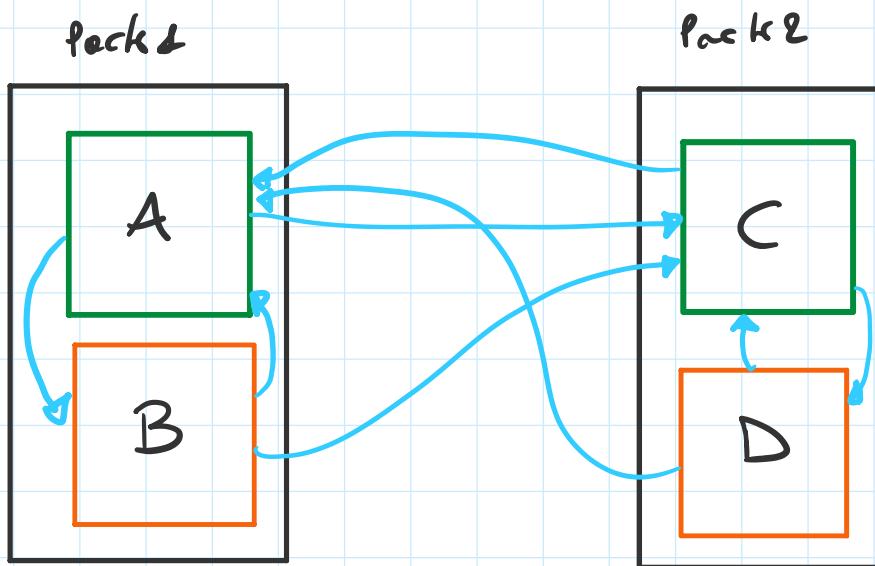
package esempio;
public class B {

M&P

```
package esempio;  
class A {  
    ;  
}  
;
```

```
package esempio;  
public class B {  
    ;  
}  
;
```

Esempio:



- classe **public**
- classe **non public**
- può usare

Files

- Main.java
- A.java
- B.java
- C.java
- D.java

```

A.java x
1 package pack1;
2
3 public class A {
4     // b e' il riferimento
5     // pack1.B e' il tipo, usando il nome
6     // completamente qualificato
7     pack1.B b = new pack1.B();
8     pack2.C c = new pack2.C();
9     pack2.D d = new pack2.D();
10 }

```

Console Shell

```

1.3.jar:/run_dir/json-simple-1.1.1.jar -d . A.java B.java C.java D.java
Main.java
A.java:9: error: D is not public in pack2; cannot be accessed from outside package
    pack2.D d = new pack2.D();
                           ^
B.java:6: error: D is not public in pack2; cannot be accessed from outside package
    pack2.D d = new pack2.D();
                           ^
C.java:5: error: B is not public in pack1; cannot be accessed from outside package
    pack1.B b = new pack1.B();
                           ^
D.java:5: error: B is not public in pack1; cannot be accessed from outside package
    pack1.B b = new pack1.B();
                           ^
A.java:9: error: D is not public in pack2; cannot be accessed from outside package
    pack2.D d = new pack2.D();
                           ^
B.java:6: error: D is not public in pack2; cannot be accessed from outside package
    pack2.D d = new pack2.D();
                           ^
C.java:5: error: B is not public in pack1; cannot be accessed from outside package
    pack1.B b = new pack1.B();
                           ^
D.java:5: error: B is not public in pack1; cannot be accessed from outside package
    pack1.B b = new pack1.B();
                           ^
8 errors
exit status 1
> []

```

```

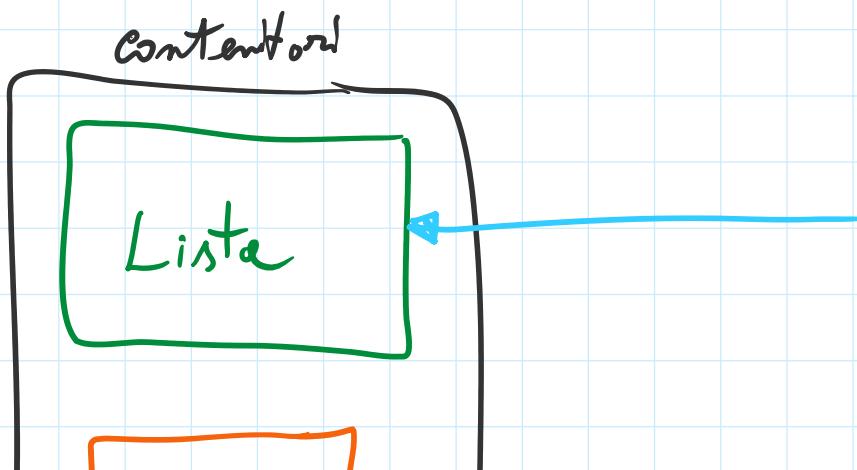
package pack1;

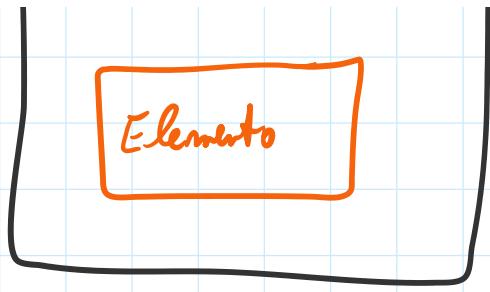
public class A {
    pack1.B b = new pack1.B(); // OK: B è nello stesso package NA
    pack2.C c = new pack2.C(); // OK: C è in altro package, C è public
    pack2.D d = new pack2.D(); // ERRORE: D è in altro package
}                                // D è non public

```

Lo stesso vale per le altre classi

Esempio





È possibile usare il nome semplice
(e non il nome completamente qualificato)
per riferirsi a classi che sono nello
stesso package.

È possibile "importare" le classi di
altri package e a questo punto possiamo
usare il nome semplice.

Per "importare" una classe dobbiamo scrivere

import nome del package. NomeDellaClasse;

↑
dopo package ...

e prima della definizione della classe

Lo stesso esempio di prima :

Lo stesso esempio di prima:

```
package pack1;  
  
import pack2.C;  
// D non può essere importata  
// non è public  
  
public class A {  
  
    B b = new B();  
    C c = new C();  
}
```

A.java

```
package pack1;  
import pack2.C;  
  
class B {  
  
    A a = new A();  
    C c = new C();  
}
```

```
package pack2;  
import pack1.A;  
  
public class C {  
  
    A a = new A();  
}
```

A $a = \text{new } A();$

D $d = \text{new } D();$

}

D è fatto in modo simile

Potrò avere tanti import

import java.net.Socket;
import java.util.Date;
:

E' possibile importare tutte le classi
di un package con *

Esempio :

import java.net.*;

→ importa tutte le classi del package
java.net

Quando faccio un import con *
non vengono importate ; package
"annidati"

package abc;

package abc.def;

Se scrivo

import abc.*;

↑ importa le classi del package abc ma non le classi del package abc.def

import abc.def.*; OK, deve essere esplicito

Il package java.lang viene

importato automaticamente

per questo posso scrivere semplicemente

String al posto di java.lang.String
anche senza fare import java.lang.*;

Se class con stesso nome in package
dovrei

```
import java.util.*;  
import java.sql.*;
```

:

Date x = new Date(); ←

quale Date ? *java.sql.Date* o
java.util.Date ?

java.util.Date x = new *java.util.Date*();

Modificatori di accesso

E' possibile applicare dei modificatori di
accesso ai campi e ai metodi di una classe

livelli possibili

private

il campo o il metodo può essere
accessato solo dalla stessa classe
in cui è definito

< nessun modificatore>
livello package
o friendly
detto anche ↗

il campo o il metodo può essere acceduto dalla stessa classe in cui è definito e da tutte le classi dello stesso package

protected

il campo o il metodo può essere acceduto dalla classe in cui è definito, dalle classi nello stesso package e dalle sottoclassi in altri package

public

il campo o il metodo può essere sempre acceduto

Note: Per i campi e i metodi i livelli sono 4, le top level class invece possono solo essere public / non public (2 livelli)



gli stessi modificatori possono essere
applicati anche ai costruttori