



Programmazione orientata agli oggetti

Classi, package e file system

Classi e file

- Java impone delle regole molto precise di corrispondenza tra classi e file
- In generale , le regole sono:
 - **Ogni classe deve stare in un file separato**
 - **Il nome del file deve essere esattamente uguale al nome della classe con l'estensione .java**
- 💣 **Attenzione:** le lettere maiuscole e minuscole sono considerate diverse fra loro: **Counter ≠ counter**
- Nell'esempio dell'orologio, che comprendeva 3 classi (Counter, Orologio ed EsempioOrologio) avremo quindi 3 file e ogni file conterrà la classe omonima: **Counter.java, Orologio.java ed EsempioOrologio.java**
- Oppure, unico file e **una sola classe pubblica**

Counter

```
public class Counter
{
    private int val;
    public Counter() { val = 0; }
    public Counter(int n) { val = n; }
    public void reset() { val = 0; }
    public void inc() { val++; }
    public void inc(int k) { val=val+k; }
    public int getValue() { return val; }
    public String toString() {
        return "Counter di valore " + val; }
}
```

Esempio: implementazione di Orologio

```
public class Orologio
{
    private Counter ore, min;
    public Orologio()
    {ore = new Counter(); min = new Counter()}
    public void reset()
    { ore.reset(); min.reset(); }
    public void tic()
    {
        min.inc();
        if (min.getValue() == 60)
        {
            min.reset();
            ore.inc();
        }
        if (ore.getValue() == 24)
            ore.reset();
    }
    public int getOre(){return ore.getValue();}
    public int getMinuti(){return min.getValue();}
}
```

Esempio: implementazione di EsempioOrologio

```
public class EsempioOrologio
{
    public static void main(String args[])
    {
        Orologio o;
        o = new Orologio();
        o.tic();
        o.tic();
        System.out.println(o.getOre());
        System.out.println(o.getMinuti());
        o.reset();
    }
}
```

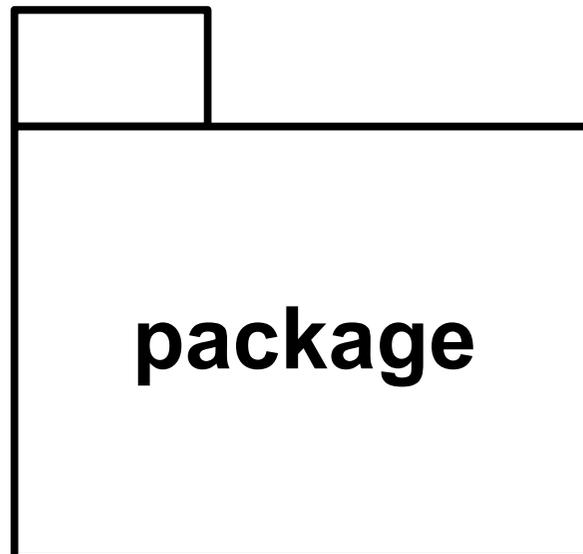
I package

- Un'applicazione è spesso composta da molte classi
- Ci possono essere gruppi di classi correlate fra loro, che in qualche modo costituiscono un'unità logica
- Java mette a disposizione il concetto di **package** per raggruppare classi che sono logicamente correlate
- **Un package è un gruppo di classi che costituiscono una unità logica**
- Un package può comprendere parecchie classi, contenute in file separati.
- Se vogliamo indicare l'appartenenza di una classe ad un package dobbiamo mettere all'inizio del file una dichiarazione di questo tipo

package <nomepackage>;

Package: rappresentazione

- Nel diagramma delle classi UML i package sono rappresentati come cartelle
- Per convenzione i nomi dei package sono tutti in minuscolo



Esempio

- Se vogliamo raggruppare in un package le tre classi dell'esempio dell'orologio dovremo inserire in ogni file la dichiarazione del package:

```
--- File Counter.java ---
```

```
package orologi;  
public class Counter  
{...}
```

```
--- File Orologio.java ---
```

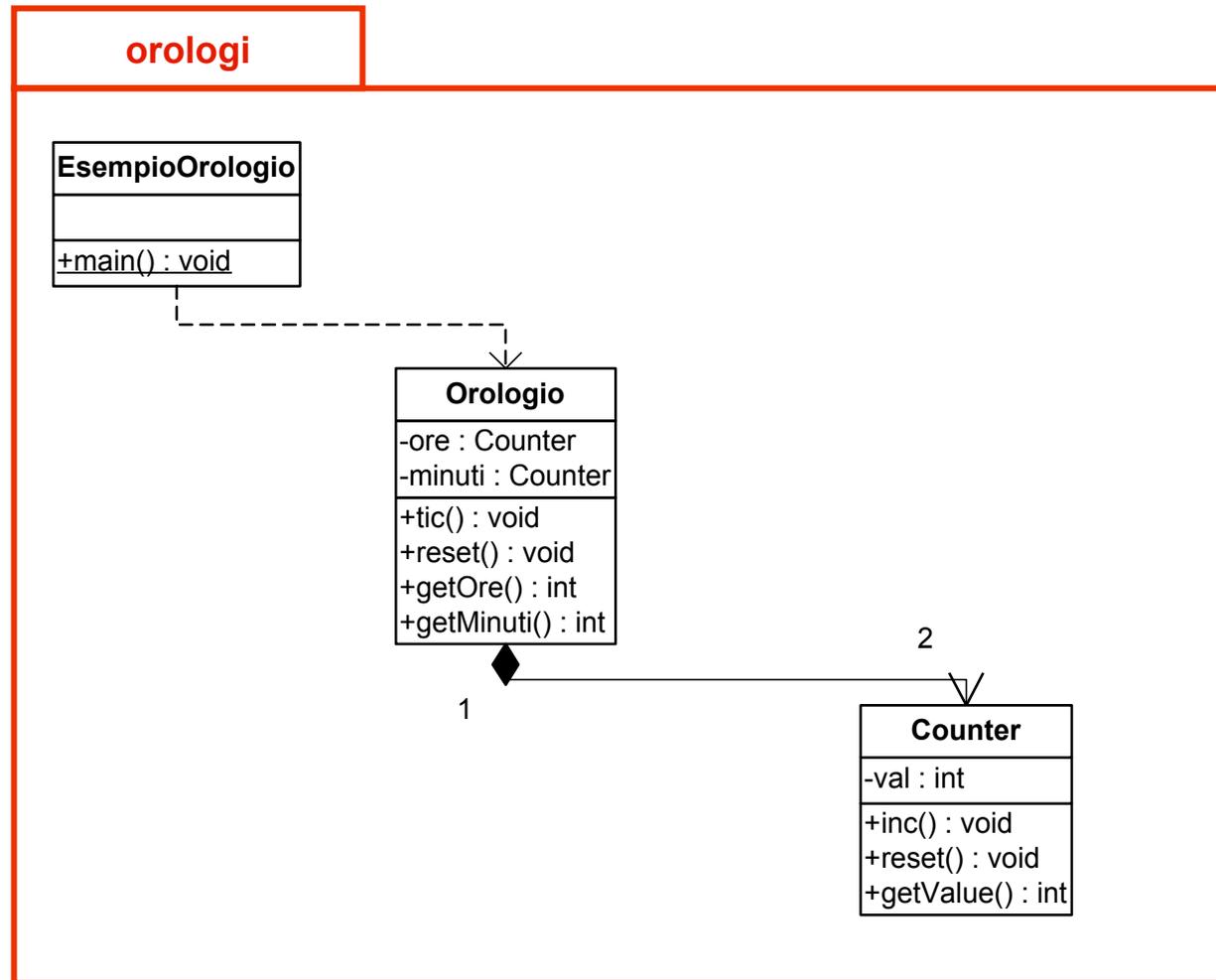
```
package orologi;  
public class Orologio  
{...}
```

```
--- File EsempioOrologio.java ---
```

```
package orologi;  
public class EsempioOrologio  
{...}
```

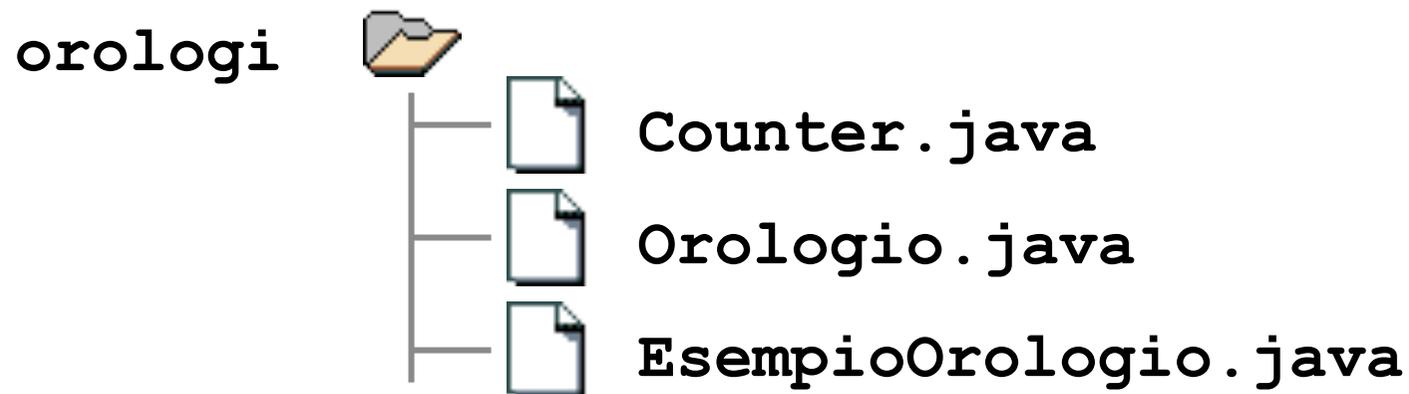
Package: diagramma

- Il diagramma delle classi sarà quindi:



Package e file system

- Esiste una **corrispondenza biunivoca** fra il nome del package e posizione nel file system delle classi del package
- Un package di nome **orologi** richiede che tutte le sue classi si trovino in una cartella (directory) di nome **orologi**



Package di default

- Se una classe non dichiara di appartenere ad alcun package, è automaticamente assegnata al **package di default**
- Per convenzione, questo package fa riferimento alla cartella (directory) corrente
- E' l'approccio usato in tutti i precedenti esempi

Altre funzioni dei package

- Abbiamo visto fino a questo momento una delle funzioni svolte dai package:
 - **Definizione di un gruppo di classi correlate**
- I package svolgono però altri due ruoli molto importanti:
 - **Definizione di uno spazio di nomi (namespace)**
 - **Definizione di un ambito di visibilità**
- Vediamo il significato di questi due ultimi concetti

Package come namespace

- Quando si opera con insiemi di classi molto complessi è abbastanza probabile che si creino omonimie
- Si usano *framework* di classi, spesso acquistati da fonti diverse ed è impensabile che non possano esistere in assoluto due classi con lo stesso nome
- Java infatti richiede solo che il **nome della classe sia unico all'interno del package di appartenenza**
- Un package è quindi un **namespace** (spazio dei nomi)

Il sistema dei nomi dei package

- Il sistema dei nomi dei package è strutturato
- Perciò, sono possibili **nomi di package strutturati**, come:

```
java.awt.print  
esempi.fondb.orologi
```

- Conseguentemente, le classi di tali package hanno un nome assoluto strutturato:

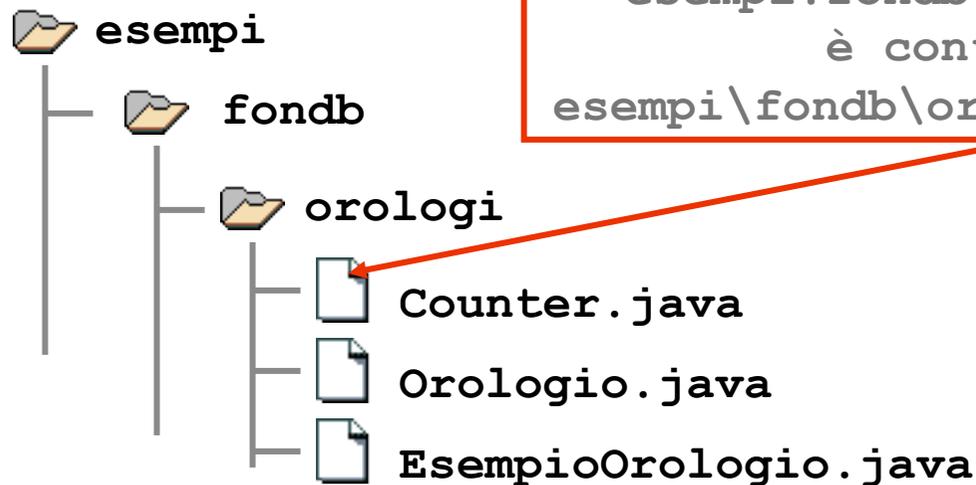
```
java.awt.print.Book
```

- Nel nostro esempio:

```
esempi.fondb.orologi.Counter  
esempi.fondb.orologi.Orologio  
esempi.fondb.orologi.EsempioOrologio
```

Sistema dei nomi strutturato e file system

- Il sistema dei nomi strutturati è simile al meccanismo dei nomi in un file system
- Si possono avere file con lo stesso nome purché in cartelle diverse
- Il file ha un nome assoluto composto dal nome del file stesso preceduto dall'intera gerarchia di cartelle in cui è contenuto
- Non a caso c'è una corrispondenza diretta tra la disposizione dei file (classi) e dei package (cartelle) nel filesystem e il nome strutturato:



`esempi.fondb.orologi.Counter`
è contenuto in
`esempi\fondb\orologi\Counter.java`

Nomi assoluti e importazione

- Ogni volta che si usa una classe, Java richiede che venga denotata con il suo nome assoluto:

```
java.awt.print.Book b;  
b = new java.awt.print.Book();
```

- Questo è chiaramente scomodo se il nome è lungo e la classe è usata frequentemente e per tale motivo si introduce il concetto di importazione di nome
- Se all'inizio del file scriviamo:

```
import java.awt.print.Book;
```

- Potremo usare semplicemente **Book** invece del nome completo:

```
Book b;  
b = new Book();
```

- Per importare in un colpo solo **tutti i nomi pubblici** di un package, si scrive

```
import java.awt.print.*;
```

Il package java.lang

- Il nucleo centrale del linguaggio Java è definito nel package `java.lang`
- È sempre importato automaticamente, anche se non scriviamo esplicitamente:
`import java.lang.*`
- Il sistema inserisce automaticamente l'importazione di questo package
- Definisce i tipi primitivi e una bella fetta della classi di sistema
- Molte altre classi standard sono definite altrove: ci sono più di cinquanta package
- `java.awt`, `java.util`, `java.io`, `java.text`,...
- `javax.swing`, ...

Package e visibilità

- I package definiscono anche un **ambito di visibilità**
- Oltre a public e private, in Java esiste un terzo livello di visibilità intermedio fra i due: la **visibilità package**
- È il default per classi e metodi: se non specifichiamo un livello la visibilità è automaticamente package
- Significa che **dati e metodi (pubblici) sono accessibili solo per le altre classi dello stesso package in qualunque file siano definite**
- Le classi definite in altri package, non possono accedere a dati e metodi di questo package qualificati a “visibilità package”, esattamente come se fossero privati