Università degli Studi di Roma “La Sapienza”
Facoltà di Ingegneria – Corso di Laurea in Ingegneria Gestionale

Corso di Progettazione del Software

Proff. Toni Mancini e Monica Scannapieco
Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”

S.JOO.1 – Java: package e visibilità

versione del February 2, 2008

Struttura di un programma Java

- Una classe è un aggregato di campi, che possono essere **dati**, **funzioni**, **classi**.
- La definizione di una classe è contenuta in un file, e un file contiene una o più definizioni di classi, una sola delle quali può essere `public`.
- Un package è una collezione di classi.
- Un file (con tutte le classi in esso contenute) appartiene ad uno ed un solo package.
- Un programma è una collezione di una o più classi, appartenenti anche a diversi package. Una di queste classi deve contenere la funzione che è il punto di accesso per l'esecuzione del programma (`main()`).

Package

- Esistono nella libreria standard Java molti package (ad esempio `java.io`)
- Un **nuovo** package `mio_pack` viene dichiarato scrivendo all'inizio di un file `F.java` la dichiarazione:

  ```
  package mio_pack;
  ```

- La stessa dichiarazione in un altro file `H.java`, dichiara che anche quest'ultimo appartiene allo stesso package.
- Se un file non contiene una dichiarazione di package, allora alle classi di tale file viene associato automaticamente un package di default, il cosiddetto package senza nome.

Uso dei package

Se, in un file `G.java`, vogliamo usare una classe `C` definita nel package `mio_pack`, possiamo usare due metodi:

1. riferirci ad essa mediante `mio_pack.C` (oppure semplicemente `C`, se essa è definita nel package senza nome);
2. scrivere all'inizio del file `G.java` una delle seguenti dichiarazioni:

```
import mio_pack.C; // semplifica il riferimento alla classe C
                   // del package mio_pack
import mio_pack.*; // semplifica il riferimento a tutte le classi
                   // del package mio_pack
```

   A questo punto, possiamo riferirci alla classe mediante `C` (senza specificare esplicitamente che appartiene a `mio_pack`).

Struttura dei package e dei direttori

Tutti i file relativi al package `mio_pack` devono risiedere in un **direttorio** dal nome `mio_pack`.

È possibile definire altri package con un nome del tipo `mio_pack.mio_subpack`.

In tal caso, tutti i file relativi al package `mio_pack.mio_subpack` devono risiedere in un **sottodirettorio** di `mio_pack` dal nome `mio_subpack`.

La dichiarazione `import mio_pack.*;` **non significa** che stiamo importando anche da `mio_pack.mio_subpack`.

Se desideriamo fare ciò, dobbiamo dichiararlo **esplicitamente** mediante la dichiarazione `import mio_pack.mio_subpack.*;`

Esempio uso package

```
// File Codice/J2/mio_package/C.java
package mio_package;

public class C {
  public void F_C() {
    System.out.println("Sono F_C()");
  }
}
// File Codice/J2/Esempio1.java
// uso package
import mio_package.*; // importo mio_package.*
public class Esempio1 {
  public static void main(String[] args) {
    C c = new C();
    c.F_C();
  }
}
// Nota: posso usare C definita in mio_package, usando il "nome corto"
```

Esempio uso package

```
// File Codice/J2/mio_package/C.java
package mio_package;

public class C {
  public void F_C() {
    System.out.println("Sono F_C()");
  }
}
// File Codice/J2/Esempio2.java
// uso package
//Nota: non importo mio_package.*
public class Esempio2 {
  public static void main(String[] args) {
    mio_package.C c = new mio_package.C();
    c.F_C();
  }
}
// Nota: posso usare C definita in mio_package, ma devo usare il "nome lun
```

Esempio uso package

```
// File Codice/J2/mio_package/C.java
package mio_package;

public class C {
  public void F_C() {
    System.out.println("Sono F_C()");
  }
}


// File Codice/J2/mio_package/mio_subpackage/D.java
package mio_package.mio_subpackage;
public class D {
  public void F_D() {
    System.out.println("Sono F_D()");
  }
}
```

Esempio uso package

```
// File Esempio3.java
// uso package
import mio_package.*;
// Nota: non importo mio_package.mio_subpackage.*
public class Esempio3 {
  public static void main(String[] args) {
    C c = new C();
    c.F_C();
    mio_package.mio_subpackage.D d = new mio_package.mio_subpackage.D();
    d.F_D();
  }
}
```

Esempio uso package

```
// File Codice/J2/mio_package/C.java
package mio_package;

public class C {
  public void F_C() {
    System.out.println("Sono F_C()");
  }
}


// File Codice/J2/mio_package/mio_subpackage/D.java
package mio_package.mio_subpackage;
public class D {
  public void F_D() {
    System.out.println("Sono F_D()");
  }
}
```

Esempio uso package

```
// File Esempio4.java
// uso package
import mio_package.mio_subpackage.*;
// Nota: non importo mio_package.*
public class Esempio4 {
  public static void main(String[] args) {
    mio_package.C c = new mio_package.C();
    c.F_C();
    D d = new D();
   d.F_D();
  }
}
```

Esercizio 1: package

```
// File Codice/J2/Esempio5.java
import java.io.*;

public class Esempio5 {
  public static void main(String[] args) throws IOException {
    // stampa su schermo il file passato tramite linea di comando
    FileInputStream istream = new FileInputStream(args[0]);
    BufferedReader in = new BufferedReader(new InputStreamReader(istream));
    String linea = in.readLine();
    while(linea != null) {
      System.out.println(linea);
      linea = in.readLine();
    }
    in.close();
  }
}
```

Riscrivere il programma eliminando la dichiarazione `import java.io.*;`.

Livelli di accesso di una classe

Una classe può essere specificata con **due** livelli di accesso:

1. `public`
2. non qualificato (è il default)

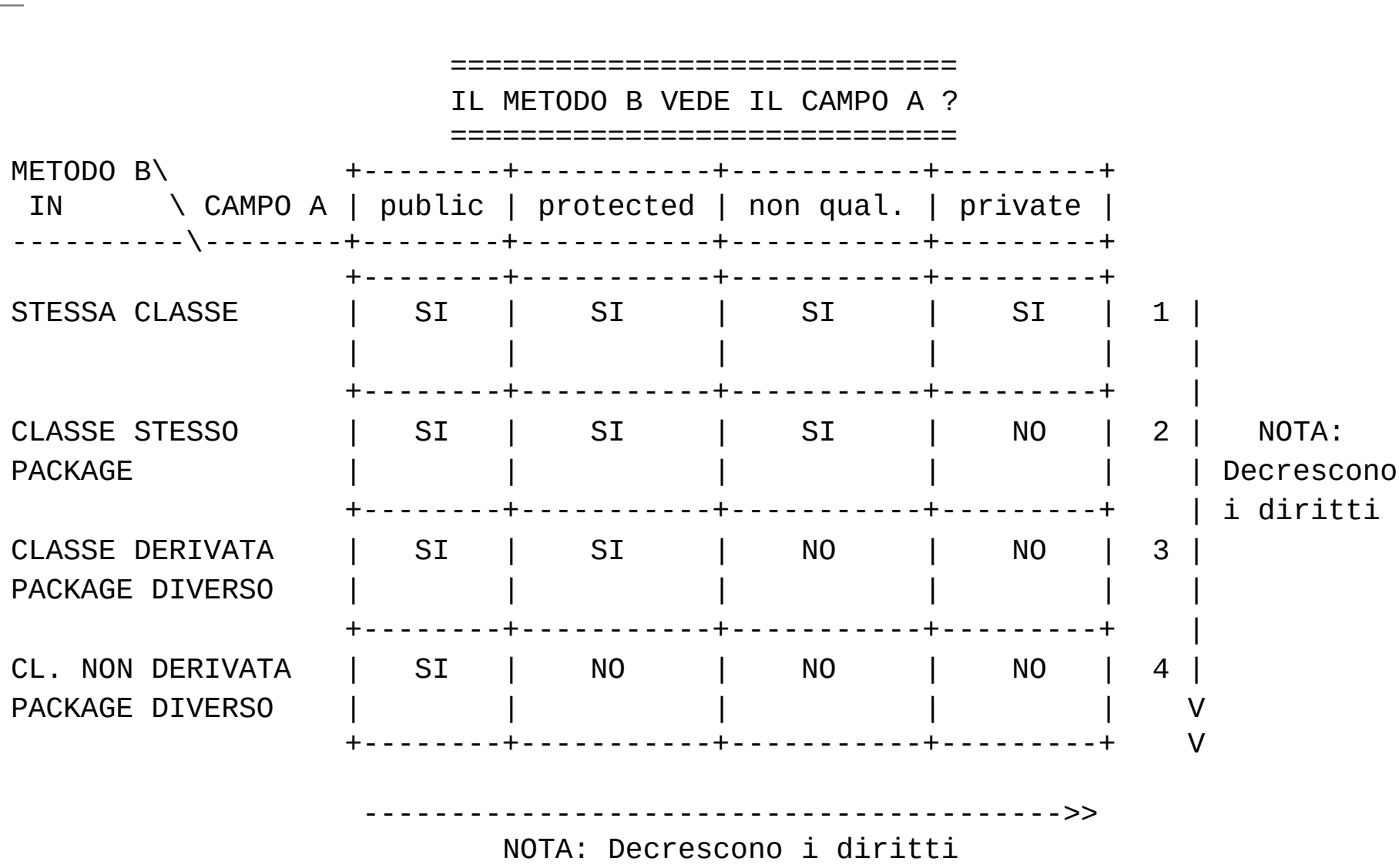
Se una classe è dichiarata `public` allora è accessibile fuori dal package al quale appartiene, altrimenti è accessibile solo all'interno del package al quale appartiene.

Livelli di accesso dei campi di una classe

Un campo di una classe (dato, funzione o classe) può essere specificato con uno fra **quattro** livelli di accesso:

1. `public`,
2. `protected`,
3. `private`,
4. non qualificato (è il default, intermedio fra protetto e privato).

Regole di visibilità tra campi



Commento sulle regole di visibilità

Le regole di visibilità vengono sfruttate per aumentare l'information hiding.

Ricordiamo che uno dei princìpi di buona modularizzazione è che l'information hiding deve essere **alto**.

In base a questo principio, i campi dati **non sono mai pubblici**, ma **privati** o **protetti**.

In tal modo diamo al cliente un **accesso controllato** ai campi dati, mediante le funzioni pubbliche.

Visibilità: esempio

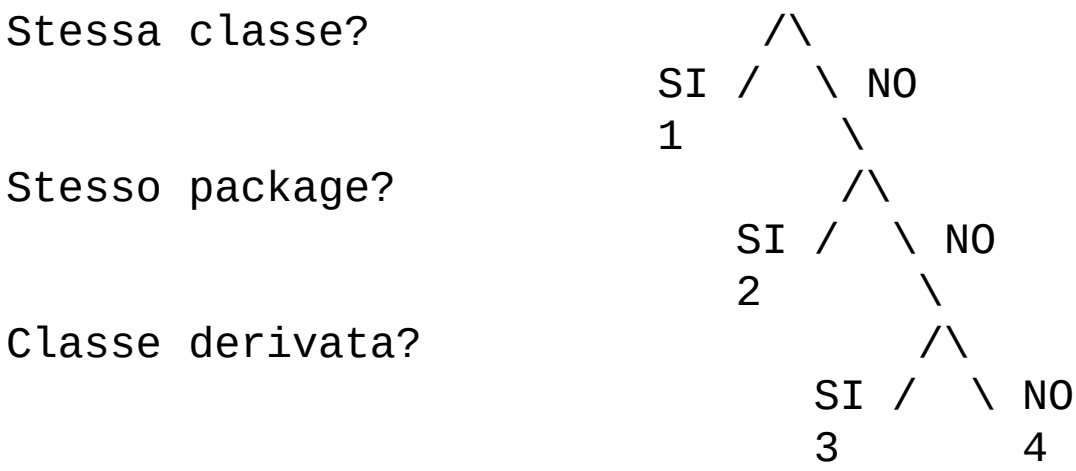
```
// File Codice/J2/Esempio6.java

class C {
  private int x, y;
  public C(int a, int b) { x = a; y = b; }
  public void stampa() { System.out.println("x: " + x + ", y: " + y); }
}

class Esempio6 {
  public static void main(String[] args) {
    C c = new C(7,12);   // OK: il costruttore di C e' pubblico
    c.stampa();          // OK: la funzione stampa() di C e' pubblica
  //    int val = c.x;   // NO: il campo x e' privato in C
  //                ^
  //Variable x in class C not accessible from class Esempio6.
  }
}
```

Esercizio 2: visibilità

Verificare, tramite opportuni frammenti di codice, la veridicità delle regole di visibilità della tabella vista in precedenza.

Regole di visibilità (cont.)

Il seguente “albero delle decisioni” fa notare che essere nello stesso package **dà più diritti** di essere una classe derivata.



Va inoltre ricordato che ogni package è “aperto”, ovvero possiamo sempre dichiarare di fare parte di un package qualunque.