Università degli Studi della Campania *Luigi Vanvitelli*

Università di degli Studi della Campania Luigi Vanvitelli Dipartimento di Ingegneria

Programmazione ad Oggetti

a.a. 2020-2021

Information Hiding

Docente: Prof. Massimo Ficco

E-mail: *massimo.ficco@unicampania.it*

1

Information Hiding

L'information hiding è il meccanismo della Programmazione ad Oggetti che consente di:

Nascondere l'implementazione di una classe

Ciò si fa regolando l'accesso agli attributi di una classe da parte di oggetti di altri classi

Specificatori/modificatori di accesso

Quando usati gli specificatori di accesso in Java sono:
public e **private**

Devono essere utilizzati per ogni membro di una classe:

- Ogni definizione di attributo
- Ogni definizione di metodo

Lo specificatore di default

Se per un metodo o un attributo non specifica il tipo di accesso si parla di:

- *package access* (oppure "friendly").

Significa che:

- Tutte le classi del package corrente possono avere accesso a quel membro.
- Tutte le classi al di fuori di quel package non possono avere accesso a quel membro

*Esempio

```
package accesso;
class Prova {
  public static void main(String args)
  {
     Default d=new Default();
     System.out.println(d.valore);
  }
}
class Default{
   int valore
}
```

Stesso file*

Una unità di compilazione (in File) può appartenere ad un solo package.

Quindi tutte le classi contenute in un file possono accedere l'un l'altra se non si specifica per gli attributi nessun livello di accesso.

Garantire l'accesso

L'unico modo per garantire l'accesso ad un membro di una classe è:

- Rendere il membro **public:** *Tutti, ovunque possono accedere ad esso*
- Non specificare il tipo di accesso ed inserire tutte le classi nello stesso package.
- sviluppare metodi per accedere o modificare gli attributi privati (tali metodi sono conosciuti come "get/set" methods

L'ultimo punto è particolarmente utilizzato e consigliato in un corretto approccio alla programmazione ad oggetti.

Specificatore public

La keyword **public** specifica che l'attributo/metodo seguente è disponibile a tutti, in particolare al programma client che la utilizza

Supponiamo di definire un package **dessert** contenente la seguente unità di compilazione:

```
package c05.dessert;
public class Cookie {
 public Cookie() {
    System.out.println("Cookie constructor");
 }
 void bite() { System.out.println("bite"); }
} ///:~
```

Osservazioni

Si ricordi che **Cookie.class** deve trovarsi in una subdirectory chiamata **dessert**, in una directory **c05** che deve essere una delle directories del CLASSPATH.

Si ricordi che se non c'è il '.' tra i percorsi del CLASSPATH, Java non cercherà nella directory corrente.

Esempio

Ora creiamo un programma che usa **Cookie**:

```
import c05.dessert.*;
public class Dinner {
  public Dinner() {
        System.out.println("Dinner constructor");
  }

  public static void main(String[] args) {
        Cookie x = new Cookie();
        /! x.bite(); // Can't access
}
} ///:~
```

Private

La keyword **private** specifica che nessuno può accedere a quel membro eccetto la classe stessa a cui quel membro appartiene.

Solo i metodi di quella classe possono accedere agli attributi ed ai metodi privati.

Quando si utilizza private

Classi nello stesso package non possono accadere a membri privati

Questo è comodo quando più persone lavorano collaborando insieme ed indipendentemente ad un package.

L'utilizzo di private consente di cambiare quei membri senza che tale cambiamento venga avvertito dalle altre classi dello stesso package.

Quando si utilizza private V:

```
class Sundae {
 int i;
 private Sundae() {}
 public Sundae(int);
 static Sundae makeASundae() {return new Sundae();}
}

public class IceCream {
     public static void main(String[] args) {
          //! Sundae x = new Sundae();
          Sundae x = new Sundae(5);
          Sundae x = Sundae.makeASundae();
}
} ///:~
```

Potremmo volere controllare il modo in cui un oggetto viene creato. Nell'esempio siamo in grado di proibire l'accesso ad un costruttore (o a tutti).

Metodi di utilità

Metodi di utilità per il programmatore possono essere resi **private,** in modo che coloro che utilizzano la classe da codice esterno non possano accedervi

Ciò consente anche al programmatore di cambiarli o rimuoverli quando vuole

Rendere un metodo **private** permette a chi ha realizzato la classe di riservarsi questa opzione.

Esercizio

Realizzare una classe lista

Definire metodi pubblici per inserimento ed eliminazione e ricerca di elementi della lista

Definire metodi privati che facilitano tali operazioni

A cura del

Prof. Massimo Ficco

e del

Prof. Salvatore Venticinque