Uso di classi e oggetti

Prof. Francesco Acarino
IIS Altiero Spinelli
Via Leopardi 132 Sesto San Giovanni

Tipi di dati

- Tipi primitivi: interi
- Java fornisce otto tipi primitivi indipendenti dall'implementazione e dalla piattaforma
- Interi
  - Tipo **byte** **(8 bit)**
    - Interi con segno tra -128 e 127, valore di default 0
  - Tipo **short** **(16 bit)**
    - Interi con segno tra -32768 e 32767, valore di default 0,
  - Tipo **int** **(32 bit)**
    - Interi con segno tra -2^{31} e 2^{31} -1, valore di default 0
  - Tipo **long** (64 bit)
    - Interi con segno tra -2^{63} e 2^{63} -1, valore di default 0

Tipi primitivi: numeri con virgola

- Seguono standard IEEE 754
- Tipo **float** (32 bit)
  - numeri in virgola mobile con 7 cifre significative
  - compresi tra 1.4E-45 e 3.4028235E+38
  - valore di default 0.0
  - le costanti vanno terminate con **F** o **f** **(**es. float a=3.456F;)
- Tipo **double** (64 bit)
  - numeri in virgola mobile in doppia precisione (15 cifre significative)
  - compresi tra 4.9E-324 e 1.7976931348623157E+308 valore di default 0.0
  - le costanti con virgola sono di tipo double per default possono essere terminate con **D** o **d** ma non è necessario

Tipi Primitivi: caratteri

- Seguono la codifica Unicode che estende ASCII su 16 bit
- Tipo **char** (16 bit)
  - Si possono usare i caratteri o i relativi codici numerici preceduti da **\u**, sempre tra apici. Es.
    - char a='A';
    - char a='\u0041' (in esadecimale su 4 cifre);
    - char a=65;
  - Caratteri con codici tra 0 e 65535
  - valore di default '\u0000' ('\0' del C)
  - Possibilità di usare \ per caratteri particolari ('\n', '\t', '\"', '\b', '\0', …)

Operatori per i Tipi Primitivi (1)

- Java ha gli stessi operatori del C, con qualche leggera differenza
- Aritmetici (**+**, **-**, *****, **/**, **%**, **++**, **--**, **+=**, **-=**, ***=**, **/=**, **%=**)
  - Non sono applicabili a variabili di tipo boolean
- Relazionali (**<**, **>**, **<=**, **>=**, **==**, **!=**)
  - Producono risultati di tipo boolean (true, false)
  - **<**, **>**, **<=**, **>=** non sono applicabili a variabili di tipo boolean

Variabili e costanti

- In Java le variabili possono essere dichiarate ovunque nel codice
  - int a=20;
  - int n=a*10;
- Una dichiarazione consiste in uno specificatore di accesso, una serie modificatori, un tipo e un nome
- Variabili **final**
  - Il loro valore non può essere modificato (costante)
  - Possono essere dichiarate in
    - un metodo:
      - **final** nomeTipo nomeVar = espressione;
    - una classe:

specificatoreDiAccesso **static final** nomeTipo nomeVar = espressione;

- Si usano in genere nomi con caratteri maiuscoli
- **Nota:** static denota una variabile della classe, quindi non ne viene creata una copia per ogni oggetto istanziato ma tutti gli oggetti fanno riferimento alla stessa variabile

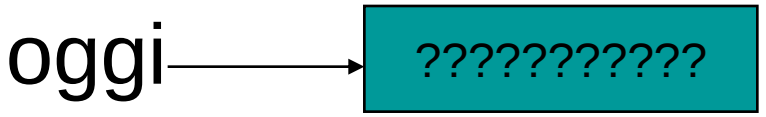
Tipi Reference

- Class (utilizzato per descrivere un classe)
- String (in java le stringhe sono degli oggetti)
- Array (anche gli array sono oggetti)

Quando viene dichiarata una variabile di tipo reference non viene creata la memoria per contenere l'oggetto ma solo per contenere il reference

Esempio

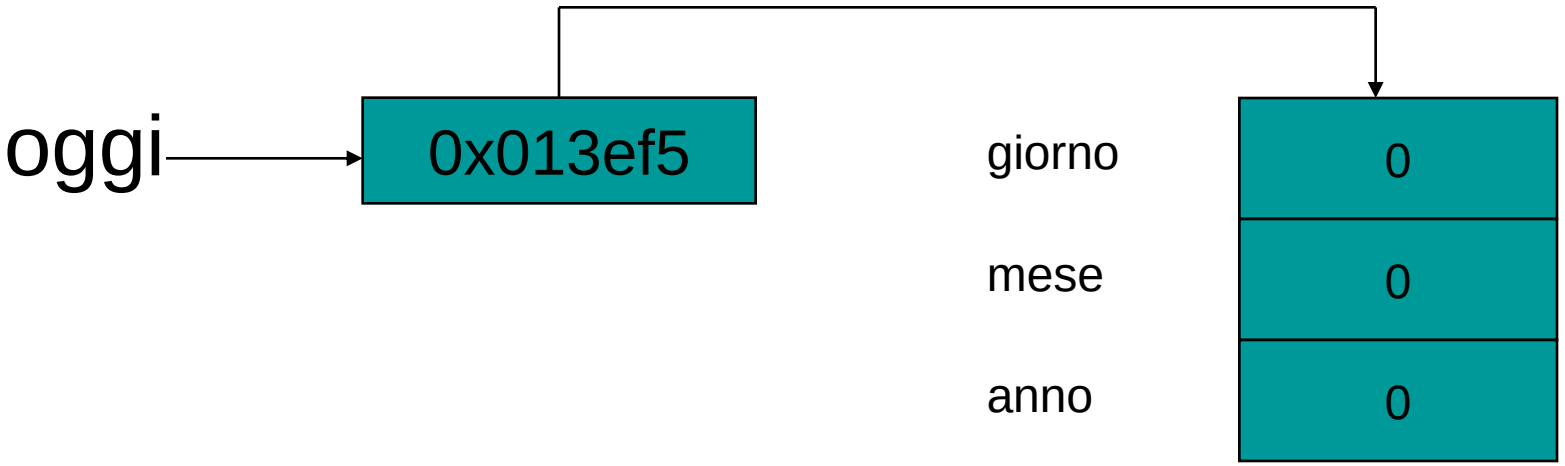
```
public class Data{
Int anno;
Int mese;
Int giorno;
}
Data oggi;  alloca memoria solo per il reference
```



Ossia oggi è la cella di memoria che conterrà l'indirizzo di memoria (reference) all'oggetto quando sarà creato

Esempio

oggi = new Data();



In memoria viene creato l'oggetto vengono inizializzate le variabili di istanza e alla variabile di riferimento viene assegnato l'indirizzo di memoria dove è contenuto l'istanza dell'oggetto

Modificatori di visibilità

Nell'esempio precedente abbiamo utilizzato la parola chiave public. Essa fa parte dei cosiddetti modificatori di visibilità. Possono essere usati per le classi per gli attributi e per i metodi.

Sono in tutto tre:

- Public: specifica una visibilità da parte di tutti
- Private: i dati e i metodi di questo tipo sono visti solo dalla classe che li contiene
- Protect i dati e i metodi di questo tipo sono visti all'esterno solo dalle classi che fanno parte della stessa libreria e dalle classi derivate dalla classe che li ha dichiarati.

Il costruttore di una classe

- Una **classe** può implementare un **metodo** particolare, detto **costruttore,** che serve a inizializzare i nuovi oggetti
- Quando esiste un **costruttore** deve chiamarsi come la **classe**
- Per creare un oggetto di una classe deve essere invocato un costruttore della classe in combinazione con l'operatore **new**
- Se una variabile di istanza non è inizializzata dal costruttore allora è inizializzata automaticamente a 0 se si tratta di un tipo numerico o a null se si tratta di un riferimento

```
oggi = new Data();
```

IL metodo Data() è il costruttore della classe dell'esempio predente

```
public class Data{
Int anno;
Int mese;
Int giorno;
}
```

Quando una classe come in questo esempio non specifica un costruttore gliene viene assegnato uno di default che inizializza a 0 tutte le variabili di istanza

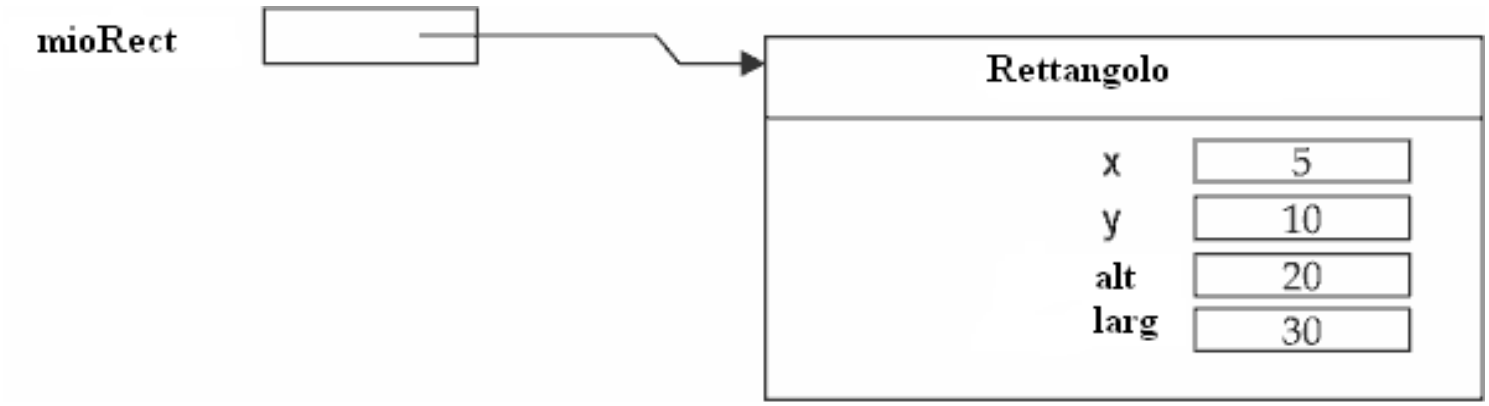
Esempio: codice Java classe Rectangle

File Rettangolo.java

```
public class Rettangolo{
    // variabili di istanza
    private int x,y,larg,alt;
    //costruttore
    public Rettangolo(int x_init,int y_init,int larg_init, int alt_init) {
            x=x_init;
            y=y_init;
            larg=larg_init;
            alt=alt_init;
    }
    //metodo che sposta il rettangolo
    public void translate(int x_new,int y_new){
            x=x+x_new;
            y=y+y_new;
    return;
    }
… //altri metodi
    }
```

Variabili oggetto

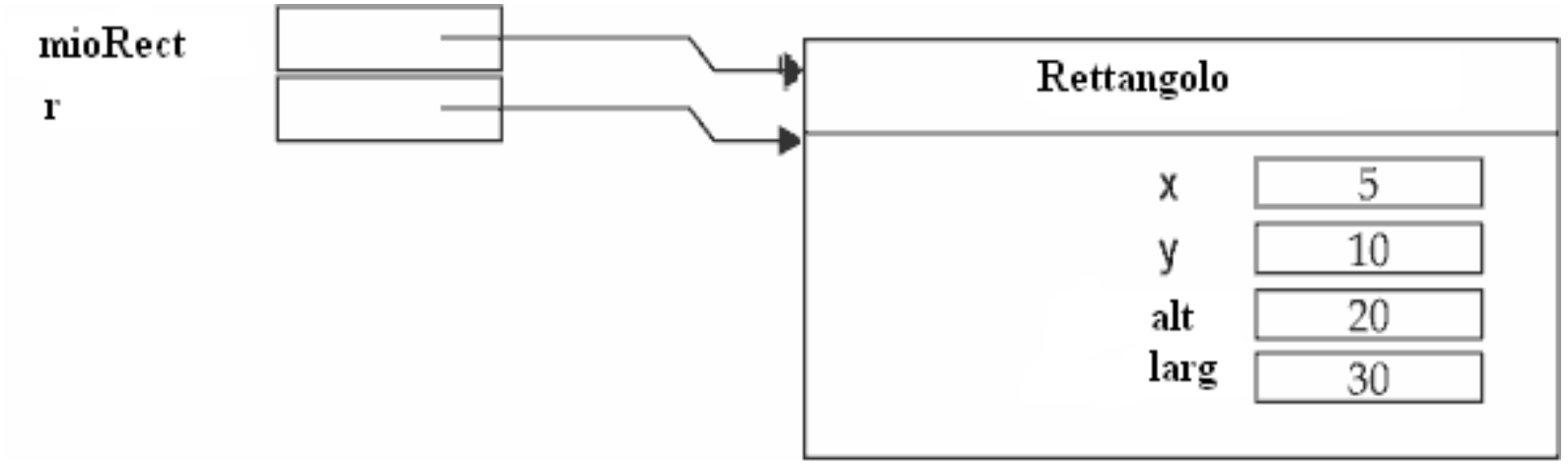
Rettangolo mioRect= new Rettangolo(5,10,20,30);



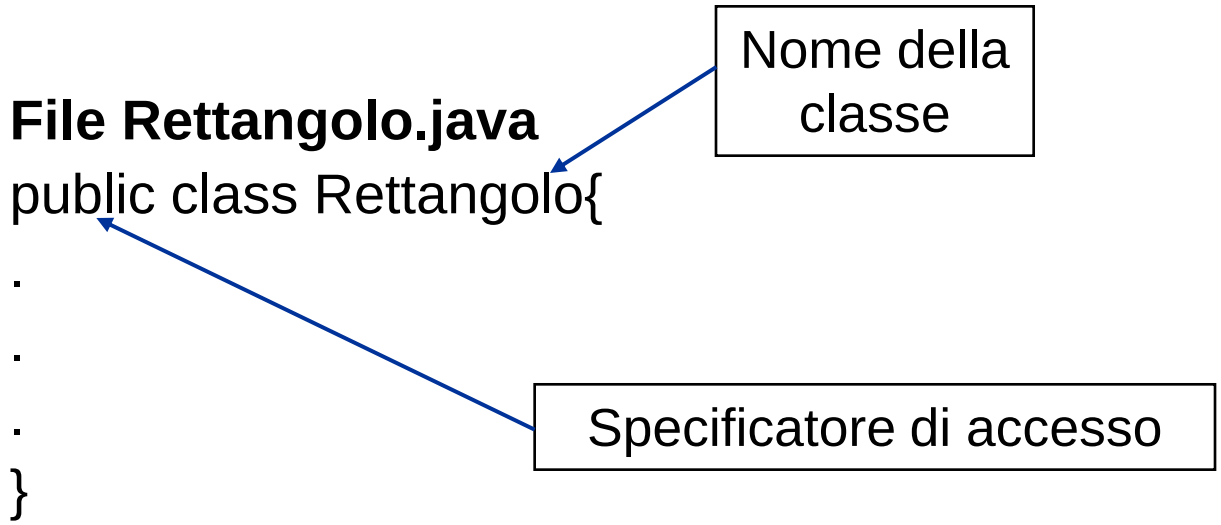
La variabile mioRect contiene un riferimento (indirizzo) ad un oggetto di tipo Rettangolo.

Variabili oggetto 2

- Rectangolo mioRect = new Rettangolo(5, 10, 20, 30);
- Rettangolo r = mioRect; (assegnamento a variabile)
  - Si ottengono due variabili oggetto che si riferiscono allo stesso oggetto

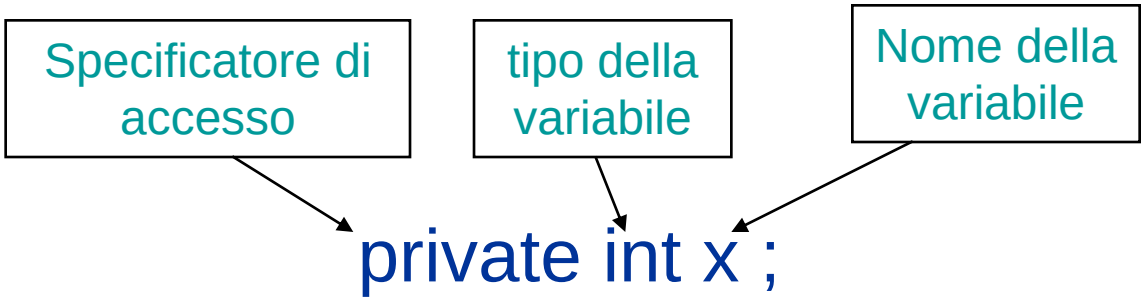


Definizione di una classe



- *specificatore di accesso* **public** indica che la classe Rettangolo è utilizzabile anche al di fuori del *package* di cui fa parte la classe
- una classe pubblica deve essere contenuta in un file avente il suo stesso nome
  - Es.: la classe Rettangolo è memorizzata nel file Rettangolo.java

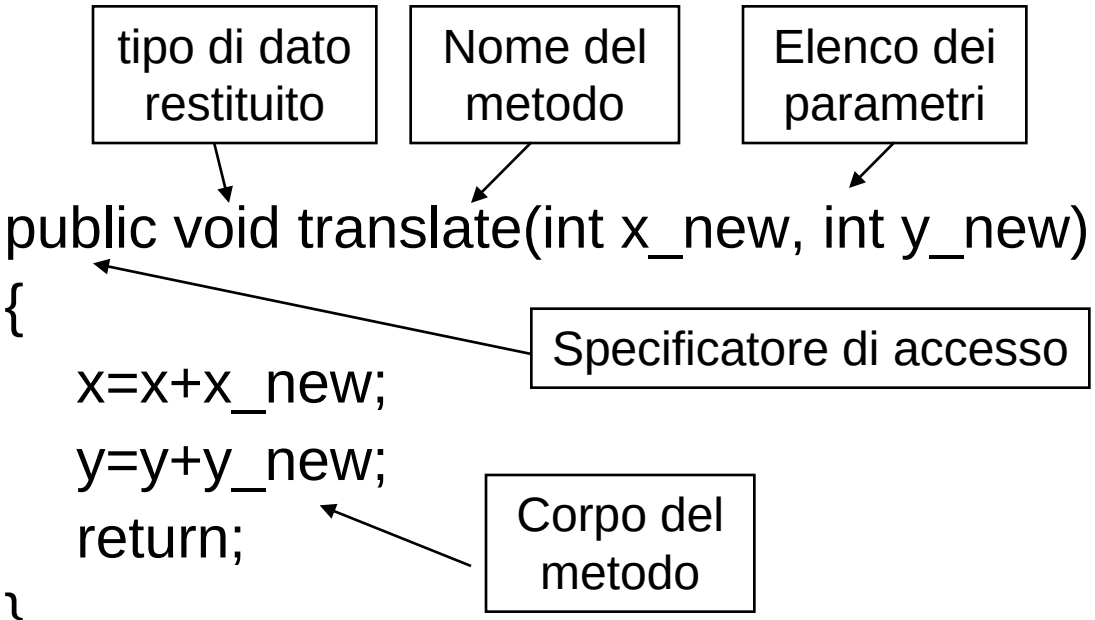
Definizione di una variabile istanza



- Lo *specificatore di accesso* indica la visibilità *(scope)* della variabile
  - **private** indica che la variabile istanza può essere letta e modificata solo dai metodi della classe
    - dall'esterno è possibile accedere alle variabili istanza **private** solo attraverso i metodi pubblici della classe
    - Solo raramente le variabili istanza sono dichiarate **public**
- Il tipo delle variabili istanza può essere
  - una classe, Es.: String
  - un array
  - un tipo primitivo, Es.: int

} Tipi riferimento

Definizione di un metodo



- Lo *specificatore di accesso* indica la visibilità (scopo) del metodo
  - **public** indica che il metodo può essere invocato anche dai metodiesterni alla classe Rectangle (e anche da quelli esterni al package a cuiappartiene la classe Rectangle)

Invocazione dei metodi

- Per invocare un metodo di un certo **oggetto** bisogna specificare il nome del metodo preceduto dal nome dell'oggetto e da un punto
  - Es.: mioRect.translate(1,34);
    (Eseguiamo il metodo *translate* sull'oggetto **mioRect** passandogli i valori 1 e 34)
- L'oggetto funge da *parametro implicito* nell'invocazione del metodo
  - E' come passare a *translate* come terzo parametro mioRect

Parametri impliciti

- Il nome di una variabile istanza o di un metodo all'interno di un metodo di una classe si riferisce alla variabile istanza o al metodo della stessa classe
- All'interno di un metodo per riferirsi esplicitamente alla classe si può usare la parola chiave this
- Si usa this.nomeMetodo o this.nomeVariabile è frequente nei costruttori quando si usano parametri formali con lo stesso nome delle variabili istanze
  - ad es.,

```
public Rettangolo(int x,int y,int larg, int alt) {
                this.x=x;
                this.y=y;
                this.larg=larg;
                this.alt=alt;
        }
```

Il metodo costruttore

```
public Rettangolo(int x_init,int y_init,int larg_init, int alt_init) {
    x=x_init;
    y=y_init;
    larg=larg_init;
    alt=alt_init;
}
```

- Una **classe** può implementare un **metodo** particolare, detto **costruttore,** che serve a inizializzare i nuovi oggetti
- Quando esiste un **costruttore** deve chiamarsi come la **classe**
- Per creare un oggetto di una classe deve essere invocato un costruttore della classe in combinazione con l'operatore **new**
- Se una variabile di istanza non è inizializzata dal costruttore allora è inizializzata automaticamente a 0 se si tratta di un tipo numerico o a null se si tratta di un riferimento

Overloading (sovraccarico)

- Più metodi con lo stesso nome
- Consentito se i parametri li distinguono, cioè hanno firme diverse
  (firma = nome del metodo + lista tipi dei parametri nell'ordine in cui compaiono)
  - Il tipo restituito non conta
- Frequente con costruttori
  - Devono avere lo stesso nome della classe

    Es.: aggiungiamo a Rettangolo il costruttore

```
public Rettangolo(int x_init,int y_init) {
x=x_init;
y=y_init;
}
```

- Usato anche quando dobbiamo agire diversamente a seconda del tipo passato

Usare classi di altri pacchetti

- Le classi non contenute nel pacchetto java.lang devono essere importate
  - Es.: import java.awt.Rectangle;
- Se interessano tutte le classi di un pacchetto si può usare il costrutto ***import nomePacchetto.****
  - Es.: import java.awt.*
- In alternativa, si può specificare tutto il nome di ogni classe.
  - Es.: java.awt.Rectangle cerealBox =

    new java.awt.Rectangle(5, 10, 20, 30);

Esempio: importazione di Rectangle

File MoveTest.java

```
//importa la classe Rectangle del pacchetto java.awt
import java.awt.Rectangle;
public class MoveTest
    {
    public static void main(String[] args)
            {
            Rectangle mioRect = new Rectangle(5, 10, 20, 30);
            // sposta il rettangolo
            mioRect.translate(15, 25);
            // stampa il rettangolo spostato
            System.out.println(mioRect);
            }
    }
```

Categorie di variabili

- **Variabili istanza**
  - Appartengono all'oggetto
  - Esistono finché l'oggetto esiste
  - Hanno un valore iniziale di default
- **Variabili locali**
  - Appartengono al metodo
  - Vengono create all'attivazione del metodo e cessano di esistere con esso
  - Non hanno valore iniziale se non inizializzate
- **Parametri formali**
  - Appartengono al metodo
  - Vengono create all'attivazione del metodo e cessano di esistere con esso
  - Valore iniziale è il valore del parametro reale al momento dell'invocazione

Progettazione ad oggetti

- Caratterizzazione attraverso le **classi** delle entità (oggetti) coinvolte nel problema da risolvere (individuazione classi)
  - identificazione delle classi
  - identificazione delle responsabilità (operazioni) di ogni classe
  - individuazione delle relazioni tra le classi
    - dipendenza (usa oggetti di altre classi)
    - aggregazione (contiene oggetti di altre classi)
    - ereditarietà (relazione sottoclasse/superclasse)
- Realizzazione delle classi

Realizzazione di una classe

1. individuazione dei metodi dell'interfaccia pubblica:
   - determinazione delle operazioni che si vogliono eseguire su ogni oggetto della classe
2. individuazione delle variabili istanza:
   - determinazione dei dati da mantenere
3. individuazione dei costruttori
4. Codifica dei metodi
5. Collaudo del codice

Incapsulamento dei Dati (1)

- In genere quando si definisce una classe, i metodi dell'interfaccia pubblica sono dichiarati **public**, mentre i metodi di servizio e tutti i campi sono dichiarati **private**
  - Si rende accessibile al mondo esterno solo l'interfaccia dell'oggetto e si nasconde la sua implementazione (astrazione delle caratteristiche della classe)
  - L'interfaccia determina il modo in cui l'oggetto comunica con l'ambiente circostante
- Questo processo di nascondere i dettagli di definizione degli oggetti si chiama incapsulamento dei dati ed è uno dei concetti base della programmazione a oggetti

Incapsulamento dei Dati (2)

- L'incapsulamento dei dati presenta due notevoli
- vantaggi
  - Gli oggetti sono protetti da un utilizzo errato o fraudolento
    - Solo i metodi definiti per la classe possono alterare i campi di un oggetto
  - Le classi sono facilmente riutilizzabili
    - Alcuni programmatori possono utilizzare classi già definite
      - Non hanno bisogno di sapere come è stata implementata la classe
      - Possono creare oggetti appartenenti alla classe e invocarne i metodi dell'interfaccia pubblica
      - Possono creare nuove classi che utilizzano la classe come membro

Programmi Java

- Un programma Java consiste di una o più classi

- Per poter eseguire un programma bisogna definire una classe pubblica che contiene un metodo

```
 public static void main(String[ ] args){
        .......
}
```

Primo Programma Java

```
Public class Saluto
{
Public static void main(String arg[])
    {
       System.out.println(”ciao mondo”);
    }
}
```