



Metodi e variabili istanza

Definizione e chiamata di metodi
Uso delle variabili istanza
Documentazione del codice

Definiamo la prima classe “vera”

- È una classe che si occupa di salutare, per ora solo “World”
- Per gli oggetti di questa classe è definito un **metodo** `sayHello`

```
public class Greeter {  
    public String sayHello() {  
        String message = "Hello, World!";  
        return message;  
    }  
}
```

Definizione di un metodo

- *SpecificatoreDiAccesso TipoRestituito nomeMetodo (lista_parametri) { corpo }*
- Specificatore di Accesso: `public`, `private`, `protected`
- TipoRestituito: tipo base o nome di classe
- nomeMetodo: inizia per minuscola
- Lista_parametri: coppie *Tipo nome* separate da virgola

Il corpo del metodo

- Il corpo del metodo contiene le istruzioni necessarie per l'esecuzione del compito dello stesso
- Un metodo finisce la sua attività restituendo al suo chiamante un valore del Tipo specificato prima del suo nome nella sua definizione
- La parola chiave per terminare è **return** seguita da un'espressione il cui valore è del tipo da restituire

Tipo void

- In alcuni casi un metodo può semplicemente compiere certe operazioni senza per questo dover ritornare un valore
- Per questi metodi si può usare il tipo speciale **void** che significa “nessun tipo”
- Per ritornare, in questo caso, basta inserire solo **return**; oppure niente: il metodo termina alla conclusione delle operazioni del blocco principale (le `{ }` che racchiudono il suo corpo)

Collaudare una classe

- La classe **Greeter** che abbiamo visto non è dotata di un metodo **main**
- È una classe progettata per una certa funzione (salutare) ed esiste tranquillamente senza la presenza di nessun **main**
- Questa è una situazione tipica per un linguaggio orientato agli oggetti come Java
- Per collaudare una classe possiamo decidere di usare BlueJ o di scrivere una classe test

Classe test

- Viene creata esclusivamente per contenere un main che crea/utilizza/testa/collauda i metodi di una o più classi:

```
public class GreeterTest {  
    public static void main(String [] args) {  
        Greeter worldGreeter = new Greeter();  
        System.out.println(worldGreeter.sayHello());  
    }  
}
```

Esercizi

- Provare a collaudare la classe con Bluej
- Provare a vedere cosa succede se i file GreeterTest.java e Greeter.java sono in directory diverse o nella stessa directory
- Provare a compilare dalla linea di comando solo GreeterTest.java
- Si verifichi che il compilatore compila automaticamente anche tutte le classi usate da quella che sta compilando, se le trova

Nota sulla compilazione/esecuzione da console

- Per compilare un file .java occorre trovarsi nella directory in cui si trova il file
- Lanciare C:\Java>javac NomeClasse.java
- Possibile problema:
 - javac non viene trovato
- Possibili soluzioni:
 - inserire nella variabile di ambiente PATH la directory in cui è installata la jdk 1.4.1 con il suffisso \bin
 - Esplicitare il nome completo di path del file javac.exe:
C:\Java> C:\jdk.1.4.1_01\bin\javac NomeClasse.java

Nota sulla compilazione/esecuzione da console

- Possibile problema:
 - La classe che si sta compilando/ eseguendo non viene trovata
- Possibile soluzione:
 - Inserire nella linea di comando la seguente opzione: C:\Java> java -cp . NomeClasseOppure
 - C:\Java> javac -cp . NomeClasse.java

Classi che si usano

- Quando all'interno di una classe A ne viene riferita un'altra, B, in un qualunque contesto (tipo di ritorno, dichiarazione di variabile riferimento, ...) si dice che la classe A **usa** B
- In genere A e B sono definite in file .java separati
- Il compilatore, nel compilare A, deve conoscere B.class (o B.java, da cui ricava B.class)
- Stessa cosa vale per la JVM in fase di esecuzione
- Ovviamente due classi possono usarsi a vicenda
- La relazione "usa" è la freccia tratteggiata in Bluej

Packages

- Le classi dei pacchetti java.lang e java.io e quelle importate con import sono rintracciate dal compilatore e/o dalla JVM tramite il classpath (nella variabile di ambiente o nell'opzione specificata sulla riga di comando [-cp lista_di_cartelle])
- Esiste comunque sempre, implicitamente, un package senza nome che corrisponde a tutte le classi presenti (file .java o .class) nella cartella corrente di lavoro

Packages

- In conclusione: sia in fase di compilazione che in fase di esecuzione le classi che si usano devono essere:
 - Nello stesso package (vero e proprio o quello senza nome corrispondente alla cartella di lavoro)oppure
 - In package diversi, ma che vengono importati (nel file .java della classe che usa deve essere importato il package della classe usata)oppure
 - Nei package importati implicitamente
- Se ci sono più classi che si usano conviene compilarle insieme: C:\Java> javac *.java

29/10/2004

Laboratorio di Programmazione - Luca Tesei

13

Miglioriamo il Greeter

- Gli oggetti della classe **Greeter** che possiamo creare fanno tutti la stessa cosa: stampare, tramite il metodo **sayHello** il saluto "Hello, World!"

```
public class GreeterTest {  
    public static void main(String [] args) {  
        Greeter worldGreeter = new Greeter();  
        Greeter anotherGreeter = new Greeter();  
        System.out.println(worldGreeter.sayHello());  
        System.out.println(worldGreeter.sayHello());  
        // Stampano entrambi la stessa stringa  
    }  
}
```

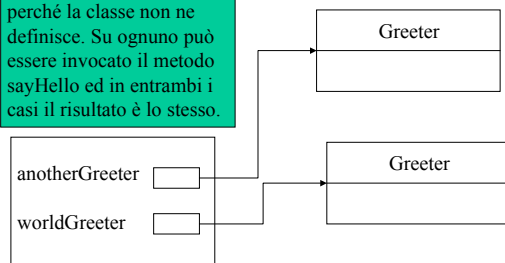
29/10/2004

Laboratorio di Programmazione - Luca Tesei

14

La situazione al momento della stampa

Sono due oggetti diversi, senza variabili istanza perché la classe non ne definisce. Su ognuno può essere invocato il metodo **sayHello** ed in entrambi i casi il risultato è lo stesso.



29/10/2004

Laboratorio di Programmazione - Luca Tesei

15

Variabili istanza

- Miglioriamo la classe **Greeter** prevedendo che ogni oggetto abbia un particolare nome che viene salutato
- Ogni oggetto deve avere quindi la possibilità di memorizzare da qualche parte il suo nome da salutare
- È proprio questo il ruolo delle variabili istanza: dotare ogni oggetto della classe di un suo proprio stato

29/10/2004

Laboratorio di Programmazione - Luca Tesei

16

Dichiarare variabili istanza

- All'interno del blocco principale della definizione di una classe in qualunque posizione (ma preferibilmente o tutte prima dei metodi o tutte dopo)
- *SpecificatoreDiAccesso Tipo nomeVariabileIstanza;*
- È buona norma dichiarare le variabili istanza con lo specificatore **private**
- **private** impedisce a qualunque metodo di qualunque classe diversa da quella che si sta definendo di accedere alla variabile istanza

29/10/2004

Laboratorio di Programmazione - Luca Tesei

17

Incapsulamento

- Dichiarando le variabili istanza tutte private l'unico modo per accedervi è quello di chiamare i metodi della classe
- La lettura e la modifica dello stato di ogni oggetto è quindi soggetta alle regole imposte da questi metodi
- Questo processo di mascheramento dello stato si chiama incapsulamento
- È molto utile nelle situazioni in cui è cruciale la sicurezza dello stato e il mantenimento della sua consistenza

29/10/2004

Laboratorio di Programmazione - Luca Tesei

18

Miglioramento di Greeter

- Dotiamo gli oggetti della classe **Greeter** di una variabile istanza di tipo **String** che conterrà il nome da salutare:

```
public class Greeter {  
    private String name; //Variabile Istanza  
    public String sayHello() {  
        String message = "Hello, " + name + "!";  
        return message;  
    }  
}
```

Il + è l'operatore di concatenazione tra stringhe

Variabili private

- Dato che la variabile istanza **name** è stata dichiarata **private**, solo i metodi della classe **Greeter** possono usarla:

```
public class GreeterTest {  
    public static void main(String [] args) {  
        Greeter worldGreeter = new Greeter();  
        System.out.println(worldGreeter.name);  
    }  
}
```

Errore

Costruttore

- Alla creazione di ogni nuovo oggetto della classe **Greeter** così specificata è bene che sia specificato il nome
- Definiamo un costruttore apposito che si occupa di assegnare la variabile istanza al momento della creazione dell'oggetto
- Definiamo anche un costruttore di default che, in mancanza di un nome fornito saluta il mondo intero, come la versione precedente

Nuova Greeter con costruttori

```
public class NameGreeter {  
    private String name; // Il nome da salutare  
  
    public NameGreeter() {  
        name = "World";  
    }  
  
    public NameGreeter(String aname) {  
        name = aname;  
    }  
  
    public String sayHello() {  
        String message = "Hello, " + name + "!";  
        return message;  
    }  
}
```

Costruttore di default ridefinito (si può sempre ridefinire quello che assegna i valori di default).

Costruttore con parametro nome

Collaudo

```
public class NameGreeterTest {  
    public static void main(String argv[]) {  
        NameGreeter worldGreeter = new  
            NameGreeter();  
        NameGreeter lucaGreeter = new  
            NameGreeter("Luca");  
        System.out.println(  
            worldGreeter.sayHello());  
        System.out.println(lucaGreeter.sayHello());  
    }  
}
```

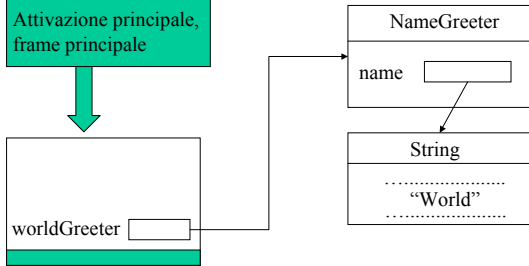
Stampa:
Hello, World!
Hello, Luca!

Stato ai vari punti del programma

```
public class NameGreeterTest {  
    public static void main(String argv[]) {  
        NameGreeter worldGreeter = new  
            NameGreeter();  
        NameGreeter lucaGreeter = new  
            NameGreeter("Luca");  
        System.out.println(  
            worldGreeter.sayHello());  
        System.out.println(lucaGreeter.sayHello());  
    }  
}
```

Stato ai vari punti del programma

Attivazione principale,
frame principale



29/10/2004

Laboratorio di Programmazione - Luca Tesei

25

Stato ai vari punti del programma

```
public class NameGreeterTest {
    public static void main(String argv[]) {
        NameGreeter worldGreeter = new
            NameGreeter();
        NameGreeter lucaGreeter = new
            NameGreeter("Luca");
        System.out.println(
            worldGreeter.sayHello());
        System.out.println(lucaGreeter.sayHello());
    }
}
```

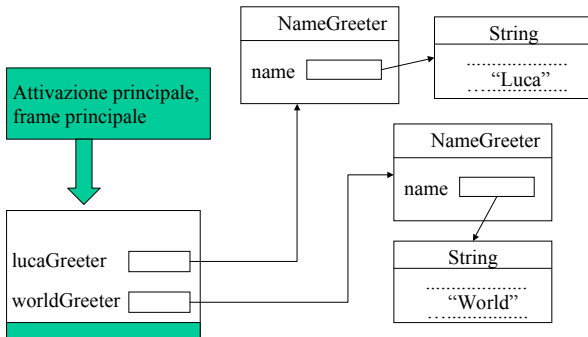
29/10/2004

Laboratorio di Programmazione - Luca Tesei

26

Stato ai vari punti del programma

Attivazione principale,
frame principale



29/10/2004

Laboratorio di Programmazione - Luca Tesei

27

Stato ai vari punti del programma

```
public class NameGreeterTest {
    public static void main(String argv[]) {
        NameGreeter worldGreeter = new
            NameGreeter();
        NameGreeter lucaGreeter = new
            NameGreeter("Luca");
        System.out.println(
            worldGreeter.sayHello());
        System.out.println(lucaGreeter.sayHello());
    }
}
```

29/10/2004

Laboratorio di Programmazione - Luca Tesei

28

Stato ai vari punti del programma

```
public String sayHello() {
    String message = "Hello, " + name + "!";
    return message;
}
```

- Il metodo **sayHello** è invocato sull'oggetto riferito/puntato da **worldGreeter**
- Viene creata un'attivazione nuova per eseguire il metodo
- Il frame principale della nuova attivazione contiene automaticamente:
 - un riferimento di nome **this** che riferisce l'oggetto su cui è stato invocato il metodo
 - I parametri del metodo con i valori che sono stati passati

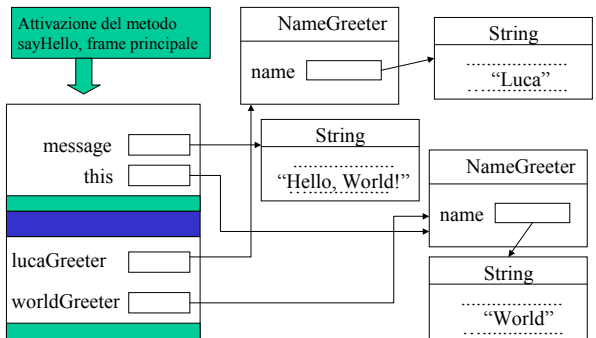
29/10/2004

Laboratorio di Programmazione - Luca Tesei

29

Stato ai vari punti del programma

Attivazione del metodo
sayHello, frame principale



29/10/2004

Laboratorio di Programmazione - Luca Tesei

30

Stato ai vari punti del programma

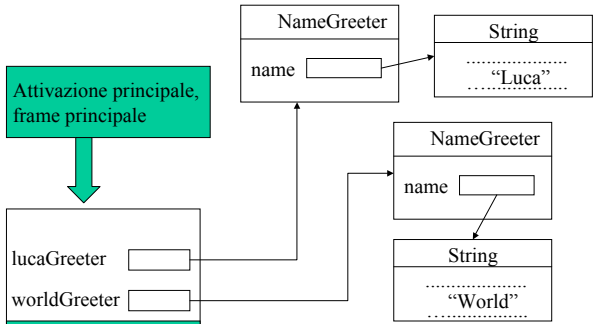
```
public class NameGreeterTest {
    public static void main(String argv[]) {
        NameGreeter worldGreeter = new
                                NameGreeter();
        NameGreeter lucaGreeter = new
                                NameGreeter("Luca");
        System.out.println(
            worldGreeter.sayHello());
        System.out.println(lucaGreeter.sayHello());
    }
}
```

29/10/2004

Laboratorio di Programmazione - Luca Tesei

31

Stato ai vari punti del programma



29/10/2004

Laboratorio di Programmazione - Luca Tesei

32

Chiamata dei metodi

- Viene creata una nuova attivazione: tutte le variabili di frame visibili immediatamente prima della chiamata diventano invisibili/congelate (tra diverse attivazioni non c'è visibilità)
- Nel primo frame della nuova attivazione vengono inseriti:
 - Riferimento **this**
 - Parametri del metodo: si usano i nomi dei parametri dati nella definizione come variabili di frame a cui è stato associato il valore passato

29/10/2004

Laboratorio di Programmazione - Luca Tesei

33

Chiamata dei metodi

- Lo heap non subisce modifiche, ma tutti gli oggetti riferiti solo da variabili di frame che si trovano in altre attivazioni sono momentaneamente irraggiungibili
- Durante l'esecuzione del metodo i parametri sono vere e proprie variabili di frame e possono quindi essere anche assegnate (passaggio dei parametri per valore)

29/10/2004

Laboratorio di Programmazione - Luca Tesei

34

Abbreviazioni

```
public String sayHello() {
    String message = "Hello, " + name + "!";
    return message;
}
```

- L'identificatore **name** dovrebbe essere scritto come **this.name** poiché si deve riferire alla variabile istanza
- Tuttavia, quando non ci sono conflitti sui nomi (uno dei parametri del metodo, o una variabile in esso dichiarata, potrebbe avere lo stesso nome di una variabile istanza) il **this** si può omettere

29/10/2004

Laboratorio di Programmazione - Luca Tesei

35

Conflitto sui nomi

```
public String sayHello() {
    String name = "Hello, " + this.name + "!";
    return name;
}
```

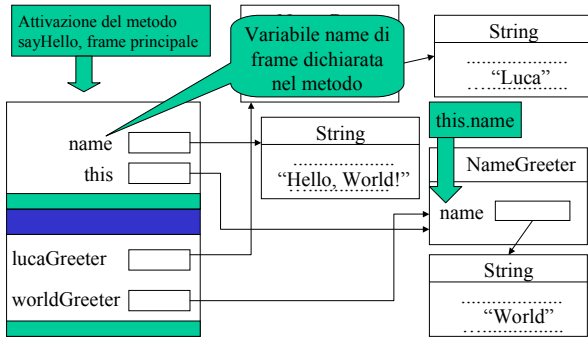
- Siamo costretti a specificare **this.name** per indicare la variabile istanza **name** dell'oggetto su cui il metodo è stato invocato.
- Il nome **name** da solo rappresenta la variabile di frame

29/10/2004

Laboratorio di Programmazione - Luca Tesei

36

Stato ai vari punti del programma



29/10/2004

Laboratorio di Programmazione - Luca Tesei

37

Un nuovo metodo

- Aggiungiamo alla classe **NameGreeter** un'ulteriore funzionalità:
- Un metodo **setName** che prende come parametro una stringa che assegna alla variabile istanza **name**
- In questo modo possiamo cambiare tutte le volte che vogliamo il nome che un oggetto della classe **NameGreeter** saluta

29/10/2004

Laboratorio di Programmazione - Luca Tesei

38

SetNameGreeter

```
public class SettableNameGreeter {
    private String name; // Nome da salutare
    public SettableNameGreeter() {
        name = "World!"; // Nome di default
    }
    public SettableNameGreeter(String aname) {
        name = aname;
    }
    public String sayHello() {
        String message = "Hello, " + name + "!";
        return message;
    }
    public void setName(String newName) {
        name = newName;
    }
}
```

29/10/2004

Laboratorio di Programmazione - Luca Tesei

39

SetNameGreeter

- Si noti che il metodo **setName** riduce l'incapsulamento della variabile privata **name**
- Siccome è un metodo pubblico chiunque può invocarlo e cambiare il valore della variabile istanza privata **name**
- È una scelta di progetto quella di fornire questa possibilità oppure no
- In genere, insieme a un metodo "set" è accoppiato il corrispondente metodo "get" che serve a leggere il valore di una variabile privata: **public String getName() { return name; }**

29/10/2004

Laboratorio di Programmazione - Luca Tesei

40

Collaudo

```
public class SettableNameGreeterTest {
    public static void main(String argv[]) {
        SettableNameGreeter myGreeter = new
            SettableNameGreeter("Pippo");
        // Stampa "Hello, Pippo!"
        System.out.println(myGreeter.sayHello());
        myGreeter.setName("Pluto");
        // Stampa "Hello, Pluto!"
        System.out.println(myGreeter.sayHello());
    }
}
```

29/10/2004

Laboratorio di Programmazione - Luca Tesei

41

Un conflitto di nome con il parametro

- ```
...
public void setName (String name) {
 this.name = name;
}
...
```
- Alla chiamata del metodo il parametro diventa una variabile di frame di nome **name** e il nome **name** si riferisce ad essa
  - Per indicare la variabile istanza bisogna utilizzare il riferimento **this**

29/10/2004

Laboratorio di Programmazione - Luca Tesei

42

## Documentare l'interfaccia pubblica

- Commentare il codice è sempre consigliabile
- Ma nel caso dell'interfaccia pubblica di una classe la documentazione è d'obbligo!
- Con un'adeguata documentazione la classe può venire usata da chiunque, anche se non ne conosce l'implementazione (è quello che facciamo sempre quando consultiamo le API)
- La jdk ha un tool, **javadoc**, che genera automaticamente le API

## Javadoc

- Inseriamo dei commenti nel codice con una determinata struttura
- Eseguiamo javadoc sui file sorgenti:  
`C:\Java> javadoc *.java`
- Otteniamo una serie di pagine html, nello stesso stile delle API, che documenta le nostre classi

## Formato dei commenti per javadoc

- Commento delle classi pubbliche
  - Va inserito prima della classe
  - **/\*\* Spiegazione di cosa fa la classe**  
    **@autor Nome-autore**  
    **... Altri tag @**  
    **\*/**
- Commento delle variabili istanza pubbliche
  - Va inserito prima della definizione
  - **/\*\* Spiegazione di cosa rappresenta la variabile istanza \*/**

## Formato dei commenti per javadoc

- Commento dei metodi pubblici
  - Va inserito prima della definizione del metodo
  - **/\*\* Spiegazione di cosa fa il metodo**  
    **@param nome-parametro1 spiegazione di cosa rappresenta il parametro**  
    **@param nome-parametro2 spiegazione di cosa rappresenta il parametro**  
    **@return spiegazione di cosa ritorna il metodo**  
    **\*/**

## Esercizio

- Provare ad eseguire javadoc e ad ottenere la documentazione html per la classe vista

## Esercizio – Bank Account

- Data la seguente interfaccia pubblica di classe:
  - Implementarla (inserire variabili istanza private, metodi di comodo, decidere come effettuare i calcoli)
  - Collaudarla (Scrivere una classe test per provare tutte le funzionalità pubbliche)
- L'applicazione tratta di conti bancari
- Deve permettere la creazione di conti, il deposito, il prelievo e la visualizzazione del saldo

## Esercizio – Bank Account

```
/** Un conto bancario ha un saldo che
 puo' essere modificato da depositi e
 prelievi
 */
public class BankAccount {
 /** Costruisce un conto bancario con saldo
 uguale a zero
 */
 public BankAccount() {
 }
}
```

(All'interno dei commenti si può scrivere html)

## Esercizio – Bank Account

```
/** Costruisce un conto bancario con un
 saldo assegnato
 @param initialBalance il saldo iniziale
 */
public BankAccount(double initialBalance) {
}

/** Versa denaro nel conto bancario
 @param amount l'importo da versare
 */
public void deposit(double amount) {
}
```

## Esercizio – Bank Account

```
/** Preleva denaro dal conto bancario
 @param amount l'importo da prelevare
 */
public void withdraw(double amount) {
}

/** Ispeziona il valore del saldo attuale
 del conto bancario
 @return il saldo attuale
 */
public double getBalance() {
}
}
```

## Documentazione interfaccia

- Si noti che questa è la definizione dell'interfaccia pubblica della classe **BankAccount**
- Manca del tutto l'implementazione
- È possibile comunque invocare javadoc anche solo su questo file e ottenere la documentazione
- Esercizio: farlo