Unità B1

Le basi della programmazione a oggetti

© 2007 SEI-Società Editrice Internazionale, Apogeo

Obiettivi

- Formalizzare **oggetti** del mondo reale in oggetti del **dominio applicativo** che possano essere utilizzati dall'applicazione, individuando solo gli aspetti che interessano e tralasciando tutto ciò che è superfluo
- Riuscire a fare **interagire** gli oggetti tra loro, al fine di raggiungere l'obiettivo per il quale è stata creata l'applicazione
- Implementare **codice** in modo tale che sia il più possibile conforme agli **standard** di programmazione

© 2007 SEI-Società Editrice Internazionale, Apogeo

OOP

- La **programmazione orientata agli oggetti** (Object Oriented Programming) ha l'obiettivo di formalizzare gli **oggetti** del mondo reale e di costruire con questi un **mondo virtuale**.
- Questa parte di mondo che viene ricostruita in modo virtuale è detta **dominio applicativo**.

© 2007 SEI-Società Editrice Internazionale, Apogeo

Gli oggetti del mondo reale

- Quotidianamente interagiamo con oggetti del mondo che ci circonda
- Oggetti:
  - animali
  - piante
  - tutti gli oggetti inanimati del mondo reale
  - un pensiero, una filosofia o più in generale un'entità astratta.
- Un esempio di oggetto astratto: il voto

© 2007 SEI-Società Editrice Internazionale, Apogeo

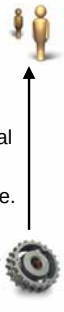
Distinguere gli oggetti

- Esempio: un bicchiere
- Ne sappiamo definire le caratteristiche e conosciamo anche quali azioni si possono fare con esso.
- Possiamo definirne la forma, il colore, il materiale di cui è fatto e possiamo dire se è pieno o vuoto.
- Sappiamo anche che si può riempire e svuotare.
- Abbiamo definito un oggetto attraverso
  - le sue caratteristiche
  - le operazioni che può compiere

© 2007 SEI-Società Editrice Internazionale, Apogeo

Il livello di astrazione

- I linguaggi di programmazione si sono evoluti in modo che i codici sorgenti potessero **astrarsi** sempre più dal modo in cui gli stessi, una volta compilati, sarebbero stati eseguiti.
- Nella OOP non ci si vuole più porre i problemi dal punto di vista del **calcolatore**, ma si vogliono risolvere facendo interagire oggetti del dominio applicativo come fossero oggetti del mondo reale.
- L'obiettivo è di dare uno strumento al programmatore, per formalizzare soluzioni ai propri problemi, pensando come una **persona** e senza doversi sforzare a pensare come una macchina.



© 2007 SEI-Società Editrice Internazionale, Apogeo

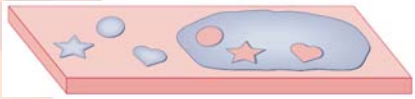
Il processo di astrazione: le classi

- Per popolare il dominio applicativo utilizzato dall'applicazione è necessario **creare gli oggetti**, e per fare questo è necessario definire le **classi**.
- Una classe è lo strumento con cui si identifica e si crea un oggetto.

© 2007 SEI-Società Editrice Internazionale, Apogeo

Una classe è un modello per la creazione di oggetti.

- La classe è paragonabile allo stampo
- gli oggetti sono i biscotti ottenuti con quello stampo



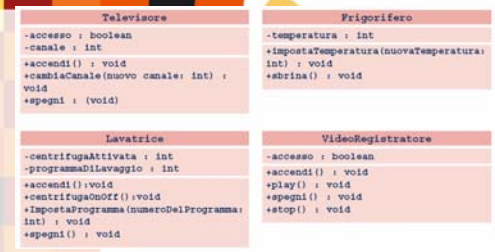
© 2007 SEI-Società Editrice Internazionale, Apogeo

Classi e tipi di dato

- Una classe è a tutti gli effetti un tipo di dato (come gli interi e le stringhe e ogni altro tipo già definito)
- Nella programmazione orientata agli oggetti, è quindi possibile sia utilizzare tipi di dato esistenti, sia definirne di nuovi tramite le classi

© 2007 SEI-Società Editrice Internazionale, Apogeo

Diagramma delle classi



La prima sezione contiene il **nome** della classe, la seconda sezione definisce i suoi **attributi**, mentre più in basso sono definiti i **metodi**, le operazioni che si possono compiere sull'oggetto di quel tipo.

© 2007 SEI-Società Editrice Internazionale, Apogeo

Le classi in Java

```
[modificatore] class [nome della classe]{
  [attributi]
  [metodi]
}
```

```
class MiaClasse {
  String mioAttributo;
  void mioMetodo() {
  }
}
```

© 2007 SEI-Società Editrice Internazionale, Apogeo

Gli oggetti

- Gli oggetti sono le entità di un programma che interagiscono tra loro per raggiungere un obiettivo
- Gli oggetti vengono creati in fase di esecuzione ed ognuno di essi fa parte di una categoria (di una classe)
- Ogni classe può creare più oggetti, ognuno dei quali pur essendo dello stesso tipo è distinto dagli altri
- Un oggetto è l'istanza di una classe

© 2007 SEI-Società Editrice Internazionale, Apogeo

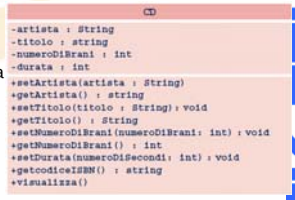
Identità tra oggetti

- Anche se due oggetti dello stesso tipo hanno tutti gli attributi con gli stessi valori, non sono uguali, ma sono oggetti distinti
- Sarebbe come dire che due gemelli, solamente perché identici fisicamente, siano la stessa persona: ovviamente è scorretto

© 2007 SEI-Società Editrice Internazionale, Apogeo

Un esempio di classe

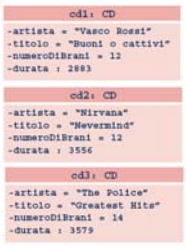
- Se vogliamo catalogare i cd musicali in nostro possesso, abbiamo bisogno di implementare un programma nel cui dominio applicativo è presente la classe CD
- I metodi della classe CD servono per impostare e recuperare i valori degli attributi



© 2007 SEI-Società Editrice Internazionale, Apogeo

Diagramma degli oggetti

- I diagrammi che rappresentano gli oggetti (Object Diagram in UML) mettono in luce i valori che assumono gli attributi



© 2007 SEI-Società Editrice Internazionale, Apogeo

Stato di un oggetto

- L'insieme dei valori degli attributi di un oggetto è chiamato stato dell'oggetto e generalmente può variare in funzione del tempo

© 2007 SEI-Società Editrice Internazionale, Apogeo

Creazione di un oggetto

- Per creare un oggetto si effettua un'istanziazione di una classe.
- In questa fase viene riservato uno spazio di memoria per conservare i valori degli attributi dell'oggetto che si sta creando (per mantenere memorizzato da qualche parte lo stato dell'oggetto)

© 2007 SEI-Società Editrice Internazionale, Apogeo

Istanziare un oggetto in Java

- A seconda del linguaggio utilizzato si impiegano diversi costrutti di programmazione per creare un oggetto
- In Java la creazione di un oggetto si effettua mediante l'istruzione **new**
- Esempio:

```
Bicchiere calice;
calice = new Bicchiere();
```

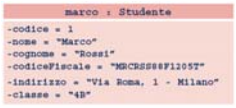
- Oppure:

```
Bicchiere calice = new Bicchiere();
```

© 2007 SEI-Società Editrice Internazionale, Apogeo

Gli attributi di istanza

- Gli attributi di istanza sono quelli posseduti da un oggetto, chiamati anche più semplicemente **attributi**.
- L'attributo di un oggetto è una variabile che ne descrive una caratteristica o proprietà



© 2007 SEI-Società Editrice Internazionale, Apogeo

Attributi costanti

- Un attributo costante è un attributo il cui valore non cambia nel tempo ma resta invariato.
- In Java per dichiarare una costante si utilizza il modificatore final.

```
public class Calendario {
   public final int numeroDeiMesi = 12;
   // Metodi
}
```

- il valore di **`numeroDeiMesi`** non può essere modificato, ma resta invariato nel corso dell'esecuzione del codice.

© 2007 SEI-Società Editrice Internazionale, Apogeo

Attributi di classe

- Un attributo di classe è un attributo condiviso da tutte le istanze della classe, ovvero da tutti gli oggetti creati con essa.
- In Java per dichiarare un attributo di classe si utilizza il modificatore **`static`**.

```
public class Gatto {
   public static int numeroDiGatti = 0;
   public Gatto() {
      numeroDiGatti ++;
   }
}
```

- Ogni volta che viene creato un oggetto di tipo **`Gatto`**, il contatore **`numeroDiGatti`** è automaticamente incrementato di uno.
- La sintassi per accedere ad un attributo di classe è:
  `<NomeClasse>.<NomeAttributo>`
  per esempio
  `System.out.print(Gatto.numeroDiGatti);`

© 2007 SEI-Società Editrice Internazionale, Apogeo

Metodi: le azioni degli oggetti

- Un metodo è un'azione che l'oggetto può eseguire.
- In Java la dichiarazione di un metodo è composta da:
  - Modificatore
  - Nome del metodo
  - Tipo di dato da ritornare
  - Tipo e nome dei parametri di ingresso
  - Eventuali eccezioni sollevate
- Tutto questo è detto **firma del metodo**.

© 2007 SEI-Società Editrice Internazionale, Apogeo

Metodi di istanza

- Un metodo di istanza è un metodo che, per essere utilizzato, ha bisogno della creazione di un oggetto della classe a cui appartiene su cui essere invocato.
- Un metodo di istanza è anche chiamato semplicemente **metodo**.

© 2007 SEI-Società Editrice Internazionale, Apogeo

Esempio di firma

- **`public int studia(String testo) throws HoStudiatoTroppoException`**
- **`public`** è il modificatore
- **`int`** è il tipo del metodo
- **`studia`** è il nome del metodo
- **`String testo`**
  è il tipo e nome dei parametri
- **`HoStudiatoTroppoException`**
  è la possibile eccezione sollevata

© 2007 SEI-Società Editrice Internazionale, Apogeo

Formalizzare i metodi

```
public void mangia(int[] portate) {

  for(int i; i < portate.length; i++) {

    calorie += portate[i];
  }
}

public void bevi(Bicchiere bicchiere) {

  bicchiere.svuota();
  liquidi += bicchiere.capienza;
}

public int corri(int kilometri) {

  calorie -= kilometri * peso * 0.9;
  acidoLattico += kilometri * 10;

  return calorie;
}

public void defaticare() {

  acidoLattico -= 50;
}
```

Un esempio: attributi

- Si vuole realizzare una classe che permetta di gestire e risolvere equazioni di secondo grado
- In una equazione individuiamo tre **attributi**: **`a, b, c`** che rappresentano i coefficienti di **x^2**, di **`x`** ed il termine noto
- L'equazione **$3x^2-2x+1=0$** avrà come attributi i valori 3, -2 e 1

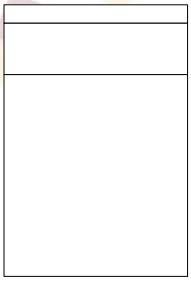
© 2007 SEI-Società Editrice Internazionale, Apogeo

Un esempio: metodi

- Definiamo un insieme di metodi che ci permetta di:
  - Modificare i valori dei coefficienti
  - Ottenere i valori dei coefficienti
  - Conoscere il tipo di equazione
  - Ottenere la prima soluzione
  - Ottenere la seconda soluzione

© 2007 SEI-Società Editrice Internazionale, Apogeo

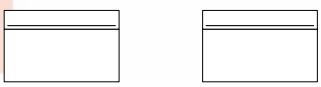
Diagramma UML della classe



© 2007 SEI-Società Editrice Internazionale, Apogeo

Esercizio

- Implementare in Java la classe Equazione
- Istanziare due equazioni:
  - $5x^2-3x+2=0$
  - $2x^2-4=0$



© 2007 SEI-Società Editrice Internazionale, Apogeo

Metodi di classe

- Un metodo di classe è un metodo invocabile sulla classe stessa senza dovere necessariamente istanziare un oggetto.
- I metodi di classe sono principalmente utilizzati per inglobare al loro interno algoritmi, o in generale operazioni che non cambiano lo stato di un oggetto.

© 2007 SEI-Società Editrice Internazionale, Apogeo

Metodi di classe, quando?

- Quando si devono modificare o leggere attributi di classe riguardanti informazioni inerenti a tutti gli oggetti della classe.
- Quando non ha senso creare oggetti di una certa classe, in quanto questa possiede solo metodi di utilità

© 2007 SEI-Società Editrice Internazionale, Apogeo

Metodi di classe: esempio

- In Java i metodi di classe si implementano utilizzando il modificatore **static**

```
public class Matematica {
   public static int somma(int addendo1,
   int addendo2) {
     return addendo1 + addendo2;
   }
}
```

- Per invocare un metodo static si utilizza la tradizionale notazione puntata, ma al posto del nome dell'oggetto si inserisce il nome della classe:

```
int risultato = Matematica.somma(3, 5);
```

© 2007 SEI-Società Editrice Internazionale, Apogeo

Il metodo main

- Il metodo **`static main`** è il il primo metodo dell'applicazione che viene eseguito.
- Questo metodo è invocato automaticamente quando si esegue una classe.
- Se si tenta di eseguire una classe priva di un metodo main si ottiene un errore.
- Il main è il metodo all'interno del quale in genere si istanziano i primi oggetti che si fanno interagire tra loro.

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

© 2007 SEI-Società Editrice Internazionale, Apogeo

Overloading

- In alcuni casi è utile avere un metodo che possa essere chiamato sia con parametri, sia senza, oppure con numero e tipo di parametri differenti.
- Nel caso di due o più metodi con lo stesso nome ma con parametri differenti si parla di **overloading**

```
public int somma(int addendo1, int addendo2) {
  return addendo1 + addendo2;
}
public float somma(float addendo1, float addendo2) {
  return addendo1 + addendo2;
}
```

- L'overloading consente di **sovraccaricare** il metodo con più di un significato.

© 2007 SEI-Società Editrice Internazionale, Apogeo

Metodo costruttore

- Il **costruttore** è un metodo particolare che viene invocato alla creazione dell'oggetto e che contiene tutte le istruzioni da eseguire per la sua inizializzazione.

© 2007 SEI-Società Editrice Internazionale, Apogeo

Costruttori in Java

- In Java i metodi costruttore:
  - Devono avere lo **stesso nome** della classe a cui appartengono.
  - Possono anche essere **vuoti** o non essere definiti. In questi casi, sull'oggetto creato non sarà effettuata alcuna operazione di inizializzazione. In particolare, se non è definito viene utilizzato il costruttore di default, assegnato automaticamente dalla JVM.
  - Possono avere **parametri** di input che serviranno per effettuare le operazioni di inizializzazione alla creazione dell'oggetto.
  - Possono esistere **più costruttori** con lo stesso nome, ma con numero e tipo di parametri differenti. In questo modo, sarà possibile creare l'oggetto invocando uno dei costruttori all'atto della creazione e passargli i parametri necessari nell'ordine e in numero uguale a quanto definito nella dichiarazione.

© 2007 SEI-Società Editrice Internazionale, Apogeo

Esempio

```
public class Bicchiere {

  public String forma;
  public String materiale;
  public boolean pieno;

  public Bicchiere() {
    pieno = false;
  }

  public Bicchiere(String nuovaForma, String nuovoMateriale) {
    forma = nuovaForma;
    materiale = nuovoMateriale;
    pieno = false;
  }
  …
}
```

© 2007 SEI-Società Editrice Internazionale, Apogeo

Modificatori

- **public**: consente a qualunque classe o oggetto di qualsiasi tipo di avere accesso all'attributo o al metodo a cui è applicato.
- **protected**: consente l'accesso solo alle classi e agli oggetti il cui tipo è una sottoclasse di quella in cui è utilizzato. Le sottoclassi saranno trattate in successive lezioni.
- **private**: consente l'accesso solo agli oggetti della classe stessa in cui è utilizzato.
- Visibilità di default: si applica senza anteporre alcun modificatore; consente a tutte le classi appartenenti allo stesso package di accedere all'attributo o al metodo.

© 2007 SEI-Società Editrice Internazionale, Apogeo

Modificatori: tabella

Tabella 1 I differenti livelli di visibilità

Livello di visibilità	Tutte le classi	Package	Sottoclassi	Classe
public	*	*	*	*
default		*		*
protected			*	*
private				*

© 2007 SEI-Società Editrice Internazionale, Apogeo

Valore e riferimento

- Un esempio:

```
int a, b;
a = 3
b = a;
a = 5;
System.out.print(b);
```

- Viene visualizzato il valore 3
- Le variabili di un tipo base contengono un valore

© 2007 SEI-Società Editrice Internazionale, Apogeo

Valore e **riferimento**

- Un altro esempio:

```
Bicchiere biccUno, biccDue;
biccUno = new Bicchiere("calice", "vetro");
biccDue = biccUno;
biccUno.forma = "coppa";
System.out.print(bicc2.forma);
```

- Viene visualizzato "coppa"
- Gli oggetti sono un **riferimento** ad una zona di memoria
- In questo caso **`biccUno`** e **`biccDue`** sono due riferimenti allo stesso oggetto

© 2007 SEI-Società Editrice Internazionale, Apogeo

Incapsulamento

- L'**incapsulamento (***information hiding)* è un concetto fondamentale dell'ingegneria del software.
- Questo principio prevede che si possa **accedere** alle informazioni di un oggetto **unicamente attraverso i suoi metodi**.
- In Java la tecnica di programmazione che consente di applicare l'incapsulamento si avvale dei **modificatori di visibilità** per nascondere gli attributi di un oggetto al mondo esterno.
- Mettere in atto questa tecnica significa **non avere** mai **attributi** di un oggetto di tipo **public**, salvo eccezioni particolari per costanti o attributi di classe da gestire in base al caso specifico.

© 2007 SEI-Società Editrice Internazionale, Apogeo

Accesso agli attributi

- Per accedere dall'esterno agli attributi, si inseriscono metodi **public** che possono essere chiamati da chiunque per impostare o richiedere il valore dell'attributo.
- I metodi hanno di solito un nome particolare:
  - set (seguito dal nome dell'attributo) per modificarne (settare) il valore
  - get (seguito dal nome dell'attributo) per recuperare (get) il valore

© 2007 SEI-Società Editrice Internazionale, Apogeo

Esempio

```
private int codice;

public void setCodice(int nuovoCodice) {
  codice = nuovoCodice;
}

public int getCodice() {
  return codice;
}
```

© 2007 SEI-Società Editrice Internazionale, Apogeo

Incapsulamento: perché?

- Potrebbe sembrare che non vi sia alcuna differenza rispetto ad accedere direttamente agli attributi
- Sembra che questa tecnica serva solo a rendere più complessa la loro gestione
- Le motivazioni sono:
  - un maggiore controllo sulle operazioni effettuate sugli attributi, limitando l'utilizzo improprio che se ne può fare e guadagnando così in sicurezza.
  - La possibilità di nascondere il modo in cui i dati sono memorizzati negli attributi

© 2007 SEI-Società Editrice Internazionale, Apogeo

Controllo sui valori inseriti

```
public void setCodice(int codice) throws CodiceErratoException {
 if( (codice >= 100) && (codice <= 1000000) ) {
  this.codice = codice;
 } else {
  throw new CodiceErratoException();
 }
}
```

© 2007 SEI-Società Editrice Internazionale, Apogeo

Convenzioni sui nomi delle classi

- Il **nome** di **classe** dovrebbe iniziare sempre con la lettera **maiuscola**.
- Nel caso di nomi composti, si utilizzano le maiuscole per le iniziali di ogni parola che compone il nome.
- Nel caso di acronimi, il nome sarà interamente maiuscolo.
  - `Persona`
  - `IndirizzoDiCasa`
  - `HTTPMessage`

© 2007 SEI-Società Editrice Internazionale, Apogeo

Nomi attributi, metodi e costanti

- I **nomi** di **attributi** e **metodi** dovrebbero iniziare con lettera **minuscola**.
  - `nome`
  - `codiceFiscale`
  - `hTTPHeader`
  - `esegui()`
  - `scriviSuFile()`
- Le **costanti** devono essere scritte in maiuscolo. I nomi composti devono avere le parti del nome separate da _.
  - `PI_GRECO`
  - `RADICE_QUADRATA_DI DUE`

© 2007 SEI-Società Editrice Internazionale, Apogeo

Convenzioni JavaBean

- Ogni attributo è definito **private**
- Ogni attributo ha una coppia di metodi **public** per impostarne e richiederne il valore.
- Il nome di questi metodi è composto dai prefissi **get** e **set**, a cui va aggiunto il nome dell'attributo.

```
private String indirizzo;
public void setIndirizzo(String indirizzo) {
  this.indirizzo = indirizzo;
}
public String getIndirizzo() {
  return indirizzo;
}
```

© 2007 SEI-Società Editrice Internazionale, Apogeo

Interazione tra gli oggetti

- Per comunicare, gli oggetti possono utilizzare i metodi, scambiandosi messaggi l'uno con l'altro.
- Quando un oggetto invoca un metodo di un altro, quest'ultimo reagisce eseguendo il metodo opportuno.
- L'invocazione dei metodi può richiedere parametri di input di qualsiasi tipo, compresi quindi oggetti del nostro dominio applicativo.
- Un oggetto potrà quindi essere in grado di passarne un altro attraverso un metodo, o addirittura potrà passare se stesso.
- Un messaggio ha la seguente sintassi:

```
<NomeOggetto>.<nomeMetodo>(<paramteri>)
```

© 2007 SEI-Società Editrice Internazionale, Apogeo

this

- In alcuni casi un oggetto ha la necessità di **riferirsi a se stesso**, per esempio all'interno di un suo metodo o del metodo costruttore.
- Questo può accadere perché l'oggetto deve riferirsi a un suo membro (attributo o metodo che sia) oppure deve passare se stesso come parametro durante l'invocazione di un metodo di un altro oggetto.
- In Java, per effettuare questa operazione, un oggetto può utilizzare la parola chiave **this**.

```
public Bicchiere(String forma, String
  materiale) {
  this.forma = forma;
  this.materiale = materiale;
  pieno = false;
}
```

© 2007 SEI-Società Editrice Internazionale, Apogeo

Sintesi (1)

- Analizzando un problema, è necessario individuare gli **oggetti** da implementare e le **caratteristiche** che li contraddistinguono, per popolare il dominio applicativo dell'applicazione.
- In una seconda fase, occorre **implementare** le classi che rappresentano gli oggetti, **incapsulando** al loro interno gli attributi e fornendole di metodi per eseguire azioni.
- Per impostare la **visibilità** corretta ad attributi e metodi a seconda del loro utilizzo e significato, ci si può servire dei **modificatori**.
- I **metodi costruttori** sono eseguiti in fase di istanziazione degli oggetti, devono avere lo stesso nome della classe e possono essere più di uno, con parametri di input differenti.
- La **firma di un metodo** è costituita dall'intera dichiarazione del metodo e non solo dal suo nome.
- All'interno di una stessa classe possono coesistere metodi con nome identico ma con firme diverse. In questo caso, il metodo viene sovraccaricato di più significati e si parla di **overloading**.

© 2007 SEI-Società Editrice Internazionale, Apogeo

Sintesi (2)

- Il **metodo main** è il metodo invocato dalla JVM per avviare l'esecuzione dell'applicazione.
- Attraverso i metodi, è possibile far **dialogare** gli oggetti che in questo modo possono scambiarsi messaggi.
- Un oggetto, per riferirsi a se stesso, può utilizzare la **parola chiave this**. Questa tecnica può essere utile, per esempio, per fare in modo che un oggetto passi se stesso a un metodo di un altro oggetto.

© 2007 SEI-Società Editrice Internazionale, Apogeo