OOP: esempio del conto corrente

(C. Horstmann)

Primo esempio di programmazione con oggetti (1)

Scriviamo un programma che usa un conto corrente Consiste di due classi:

- UsaConto che contiene il main del programma
- ContoCorrente che descrive gli oggetti che rappresentano i conti correnti

```
public class UsaConto {
public static void main(String[] args) {
// crea un nuovo conto corrente inizializzato con 1000 euro
ContoCorrente cc = new ContoCorrente(1000); // versa 700 euro
cc.versa(700);
// fa un po' di prelievi, controllando prima il saldo
if (cc.saldo>200) cc.preleva(200);
if (cc.saldo>900) cc.preleva(900);
System.out.println("Saldo finale: " + cc.saldo);
 }
}
}
```

Primo esempio di programmazione con oggetti (2)

```
public class ContoCorrente {
// variabile che memorizza lo stato del conto
public double saldo;
// costruttore della classe
public ContoCorrente(double saldoIniziale) { saldo=saldoIniziale;
}
// metodo per il versamento di somme
public void versa(double somma) { saldo+=somma;
}
// metodo per il prelievo di somme
public void preleva(double somma) { saldo-=somma;
}
}
```

Primo esempio di programmazione con oggetti (3)

Osservazioni: La classe UsaConto non è molto diversa dai programmi che abbiamo scritto fino ad ora... In UsaConto:

- cc.versa() è una invocazione di un metodo, cc.saldo è una lettura di una variabile In ContoCorrente:

- non c'è il main (ce n'e' uno solo per tutto il programma)
- c'è una variabile (saldo) che rappresenta lo stato del conto
- ci sono due metodi (versa() e preleva()) che descrivono le funzionalità del conto
- c'e' un metodo speciale (ContoCorrente()) che inizializza il conto I il metodo ContoCorrente() è detto costruttore e viene richiamato quando si usa il comano new (non prevede tipo di ritorno)
- i metodi e la variabile sono pubblici (public) quindi possono essere usati anche da altre classi (e.g. UsaConto)
- nei metodi non si usa il modificatore static (capiremo più avanti perché)

Primo esempio di programmazione con oggetti (4)

Vediamo ora come gestire più conti correnti

```
public class UsaDueConti {
public static void main(String[] args) {
// crea un nuovo conto corrente inizializzato con 1000 euro
ContoCorrente conto1 = new ContoCorrente(1000);
// crea un nuovo conto corrente inizializzato con 200 euro
ContoCorrente conto2 = new ContoCorrente(200); // preleva 700 euro dal
primo conto...
conto1.preleva(700);
// ...e li versa nel secondo
conto2.versa(700);
System.out.println("Saldo primo conto: " + conto1.saldo);
System.out.println("Saldo secondo conto: " + conto2.saldo);
}
}
```

L'esecuzione di un programma a oggetti (1))

In un programma basato su oggetti, lo stato non è più uno stato unico globale (come nel caso della programmazione imperativa) ma è composto da tutti gli stati interni di tutti gli oggetti attivi

L'esecuzione di un programma a oggetti (2)

Per descrivere l'esecuzione di un programma a oggetti ci si pùo concentrare sulle invocazioni di metodi dei vari oggetti trascurando i dettagli della loro implementazione (i singoli comandi che contengono) Un diagramma come questo fa capire che il metodo pagaStipendio() di della classe Dipendente richiama il metodo versa() di ContoCorrente

OOP e Ingegneria del Software

La programmazione orientata agli oggetti semplifica la realizzazione di programmi complessi:

- 1. Si identificano le varie entità da rappresentare tramite classi e oggetti
- 2. Si specificano le variabili e i metodi di ogni classe accessibili dalle altre classi (ossia l'interfaccia pubblica della classe)
- 3. Si implementano le varie classi separatamente, concentrandosi su una per volta

- 4. Le varie classi possono essere implementate da persone diverse indipendentemente La disciplina che si occupa di organizzare questo lavoro è l'Ingegneria del Software Definisce notazioni (diagrammi), metodologie e procedure che rendono il processo di sviluppo di software complessi più e ciente e a dabile.

OOP Highlights (1)

Riprendiamo l'esempio del conto corrente:

```
public class ContoCorrente {
public double saldo;
public ContoCorrente(double saldoIniziale) { saldo=saldoIniziale;
}
public void versa(double somma) { saldo+=somma;
}
public void preleva(double somma) { saldo-=somma;
}
}
```

OOP Highlights (2)

Riprendiamo anche il primo main che abbiamo considerato:

```
public class UsaConto {
public static void main(String[] args) {
// crea un nuovo conto corrente inizializzato con 1000 euro
ContoCorrente cc = new ContoCorrente(1000); // versa 700 euro
cc.versa(700);
// fa un po' di prelievi, controllando prima il saldo
if (cc.saldo>200) cc.preleva(200); if (cc.saldo>900) cc.preleva(900);
System.out.println("Saldo finale: " + cc.saldo); }
}
```

OOP Highlights (3)

Arricchiamo un po' il comportamento dei metodi: Tracciamo i movimenti stampando dei messaggi

```
public class ContoCorrente {
public double saldo;
public ContoCorrente(double saldoIniziale) { saldo=saldoIniziale;
}
public void versa(double somma) {
saldo+=somma;
System.out.println("Versati: " + somma + " euro");
}
public void preleva(double somma) {
saldo-=somma;
```

```
System.out.println("Prelevati: " + somma + " euro");
}
}
```

OOP Highlights (4)

Abbiamo modificato la classe... dobbiamo mettere mano anche al main (e/o alle altre classi che la usano)? NO! Highlight: se non modifichiamo l'interfaccia pubblica (nomi dei metodi, parametri, valori di ritorno) possiamo modificare la classe senza compromettere il resto del programma Più facile fare manutenzione e aggiornamenti a parti del programma!

OOP Highlights (5)

Modifiche all'interfaccia pubblica richiedono invece di modificare anche i chiamanti (in questo caso il main) Esempio: consentiamo il prelievo solo se c'e' la disponibilità

```
public class ContoCorrente {
public double saldo;
public ContoCorrente(double saldoIniziale) { saldo=saldoIniziale;
}
public void versa(double somma) {
saldo+=somma;
System.out.println("Versati: " + somma + " euro");
}
// restituisce false se non ci sono abbastanza soldi
public boolean preleva(double somma) { if (saldo<somma) return false;
else {
saldo-=somma;
System.out.println("Prelevati: " + somma + " euro"); return true;
}
}
```
```
                                                    }

## OOP Highlights (6)
Modifichiamo di conseguenza il main:
```java

public class UsaConto {
public static void main(String[] args) {
// crea un nuovo conto corrente inizializzato con 1000 euro
ContoCorrente cc = new ContoCorrente(1000); // versa 700 euro
cc.versa(700);
// fa un po' di prelievi, controllando prima il saldo. // posso tenere
conto o meno del risultato (true/false) if (!cc.preleva(200))
System.out.println("Fallito"); cc.preleva(900);
System.out.println("Saldo finale: " + cc.saldo);
}
}
```
```

OOP Highlights (7)

Prima abbiamo aggiunto la stampa dei messaggi per tracciare le operazioni sul conto... ma chi vieta all'utilizzatore di questa classe di modificare a mano il saldo? cc.saldo=10000000;

OOP Highlights (8)

Il saldo è pubblico (public) se vogliamo evitare che sia modificabile dall'esterno della classe lo dobbiamo trasformare in privato (private)

```
public class ContoCorrente {
// ora e' visibile solo all'interno di questa classe
private double saldo;
public ContoCorrente(double saldoIniziale) {
        saldo=saldoIniziale;
}
public void versa(double somma) {
saldo+=somma;
System.out.println("Versati: " + somma + " euro");
}
public boolean preleva(double somma) {
        if (saldo<somma) return false;
        else {
        saldo-=somma;
        System.out.println("Prelevati: " + somma + " euro");
        return true;
        }
        }
}
```

OOP Highlights (9)

Ora siamo sicuri che le modifiche al saldo avverranno solo tramite i metodi Ma.... come farà il main a stampare il saldo? Idee?

OOP Highlights (10)

Soluzione: Aggiungiamo un metodo

```
public class ContoCorrente {
private double saldo;
public ContoCorrente(double saldoIniziale) { saldo=saldoIniziale;
}
public void versa(double somma) {
saldo+=somma;
System.out.println("Versati: " + somma + " euro");
}
public boolean preleva(double somma) { if (saldo<somma) return false;
else {
```

```
saldo-=somma;
System.out.println("Prelevati: " + somma + " euro"); return true;
} }
// restituisce il saldo a chi ne ha bisogno
public double ottieniSaldo() { return saldo;
}
}
```

OOP Highlights (11)

Il metodo che abbiamo aggiunto consente di accedere al saldo solo “in lettura” In questo modo il valore del saldo è sempre sotto controllo dei metodi Highlight: la proprietà per cui i dati che rappresentano lo stato interno di un oggetto possono essere accessibili solo tramite i metodi dell’oggetto stesso è detta INCAPSULAMENTO L’incapsulamento consente di gestire un oggetto come una “scatola nera” (black box). Dall’esterno si sa cosa fa un oggetto, ma non come lo fa...

#Gestione della Memoria nella Programmazione Orientata agli Oggetti

- 1 Condivisione di variabili tra classi (variabili statiche) 2 Gestione della memoria nella Java Virtual Machine 3 Riferimenti a oggetti 4 Un esempio complesso (da vedere a casa...)

Esempio: conti correnti (1)

Riprendiamo l’esempio del conto corrente

```
public class ContoCorrente {
private double saldo;
public ContoCorrente(double saldoIniziale) { saldo=saldoIniziale;
}
public void versa(double somma) {
saldo+=somma;
System.out.println("Versati: " + somma + " euro");
}
public boolean preleva(double somma) { if (saldo<somma) return false;
else {
saldo-=somma;
System.out.println("Prelevati: " + somma + " euro"); return true;
} }
public double ottieniSaldo() { return saldo;
}
}
```

Esempio: conti correnti (2)

E un relativo main

```
public class UsaDueConti {
public static void main(String[] args) {
```

```
// crea un nuovo conto corrente inizializzato con 1000 euro
ContoCorrente conto1 = new ContoCorrente(1000);
// crea un nuovo conto corrente inizializzato con 200 euro
ContoCorrente conto2 = new ContoCorrente(200); // preleva 700 euro dal
primo conto...
conto1.preleva(700);
// ...e li versa nel secondo
conto2.versa(700);
System.out.println("Saldo primo conto: " + conto1.ottieniSaldo())
System.out.println("Saldo secondo conto: " + conto2.ottieniSaldo() }
}
```

Condividere variabili (1)

Supponiamo ora di voler attribuire ad ogni conto un numero identificativo il numero del conto.... dobbiamo aggiungere una variabile alla classe!

Condividere variabili (2)

```
public class ContoCorrente { private double saldo;
// memorizza il numero del conto
private int numero;
// inizializza anche il numero del conto
public ContoCorrente(double saldoIniziale, int numeroConto) {
saldo=saldoIniziale;
numero=numeroConto;
}
public void versa(double somma) { ...come prima... } public boolean
preleva(double somma) { ...come prima... } public double ottieniSaldo() {
return saldo; }
// fornisce il numero del conto
public double ottieniNumero() { return numero; }
}
```

Condividere variabili (3)

Modifichiamo di conseguenza il main

```
public class UsaDueConti {
public static void main(String[] args) {
// crea un nuovo conto NUMERO 10001 con 1000 euro
ContoCorrente conto1 = new ContoCorrente(1000,10001); // crea un nuovo
conto NUMERO 10002 con 200 euro
ContoCorrente conto2 = new ContoCorrente(200,10002); // preleva 700 euro
dal primo conto...
conto1.preleva(700);
// ...e li versa nel secondo
conto2.versa(700);
```

```
// ORA QUI POSSIAMO USARE IL NUMERO

System.out.print("Conto " + conto1.ottieniNumero());
System.out.println(" saldo : "+conto1.ottieniSaldo());

System.out.print("Conto " + conto2.ottieniNumero());
System.out.println(" saldo : "+conto2.ottieniSaldo());
}
}
```

Variabili statiche (1)

In questo modo il numero del conto deve essere deciso dal chiamante (e.g. main) Il main è responsabile di gestire i numeri dei conti Che succede se il main attribuisce lo stesso numero a due conti diversi? Sarebbe meglio se al momento della creazione un conto potesse generare il proprio numero da se Ad esempio incrementando di uno il numero dell'ultimo conto corrente creato

Variabili statiche (2)

Per rendere possibile ciò è necessaria un'informazione condivisa da oggetti ContoCorrente diversi Serve una variabile contatore che sia visibile a tutti gli oggetti ContoCorrente Tale variabile "condivisa" conterrà il numero dell'ultimo conto creato Un nuovo oggetto incrementerà la variabile condivisa di 1 e userà tale valore come proprio numero di conto Una variabile condivisa da tutti gli oggetti di una certa classe la si ottiene con il modificatore static

Variabili statiche (3)

```
public class ContoCorrente { private double saldo;
// memorizza il numero del conto
private int numero;
// variabile condivisa (inizializzata a 1000) private static int
numeroUltimoContoCreato = 1000;
// il numero del conto viene inizializzato usando la variabile cond
public ContoCorrente(double saldoIniziale) { saldo=saldoIniziale;
numeroUltimoContoCreato++; numero=numeroUltimoContoCreato;
}
public void versa(double somma) { ...come prima... } public boolean
preleva(double somma) { ...come prima... } public double ottieniSaldo() {
return saldo; }
// fornisce il numero del conto
public double ottieniNumero() { return numero; }
}
```

Variabili statiche (4)

```
public class UsaDueConti {
public static void main(String[] args) {
```

```
// crea un nuovo conto (NUMERO AUTOMATICO) con 1000 euro
ContoCorrente conto1 = new ContoCorrente(1000);
// crea un nuovo conto (NUMERO AUTOMATICO) con 200 euro
ContoCorrente conto2 = new ContoCorrente(200); // preleva 700 euro dal
primo conto...
conto1.preleva(700);
// ...e li versa nel secondo
conto2.versa(700);

System.out.print("Conto " + conto1.ottieniNumero());
System.out.println(" saldo : "+conto1.ottieniSaldo());
System.out.print("Conto " + conto2.ottieniNumero());
System.out.println(" saldo : "+conto2.ottieniSaldo());
}
}
```

Variabili statiche (5)

Altro esempio di uso di static Supponiamo di voler aggiungere al nostro programma la gestione degli interessi maturati nei conti Dobbiamo memorizzare il tasso da applicare al conto corrente

Variabili statiche (6)

```
public class ContoCorrente {
private double saldo;
private int numero;
private static int numeroUltimoContoCreato = 1000;
// memorizza il tasso di interesse (inizializzato a 0.02)
// lo dichiaro pubblico per renderlo modificabile dall'esterno public
double tasso = 0.02;
public ContoCorrente(double saldoIniziale) { ...come prima... } public void
versa(double somma) { ...come prima... }
public boolean preleva(double somma) { ...come prima... }
public double ottieniSaldo() { return saldo; }
public double ottieniNumero() { return numero; }
// aggiorna il saldo aggiungendo gli interessi
public maturaInteressi() { saldo += saldo*tasso;
}
}
```

Variabili statiche (7)

Ma... supponiamo che il tasso sia lo stesso per tutti i conti correnti. oppure (vedremo dopo) che ci siano delle "categorie" di tasso (ad esempio: tasso family e tasso business) Per cambiare i tassi di interesse devo prendere un conto corrente per volta e aggiornare la sua variabile tasso Anche in questo caso sarebbe più pratico se la variabile tasso fosse condivisa da tutte le classi (quindi static)

```
// supponendo che contiGestiti sia un array di conti correnti
for (ContoCorrente cc : contiGestiti) cc.tasso+=0.01;
```

Variabili statiche (8)

```
public class ContoCorrente {
private double saldo;
private int numero;
private static int numeroUltimoContoCreato = 1000;
// aggiungo static per condividere questa variabile
public static double tasso = 0.02;
public ContoCorrente(double saldoIniziale) { ...come prima... } public void
versa(double somma) { ...come prima... }
public boolean preleva(double somma) { ...come prima... }
public double ottieniSaldo() { return saldo; }
public double ottieniNumero() { return numero; }
// aggiorna il saldo aggiungendo gli interessi
public maturaInteressi() { saldo += saldo*tasso;
}
}
```

Variabili statiche (9)

Ora per cambiare i tassi di interesse in tutti i conti correnti è su ciente fare: Note: Le variabili statiche possono essere riferite usando il nome della classe invece che il nome di un oggetto Si pùo comunque usare anche il nome di un oggetto (es. cc.tasso+=0.01) La variabile tasso pùo essere usata anche se non esistono oggetti di tipo ContoCorrente

```
ContoCorrente.tasso+=0.01;
```

Metodi statici

Anche un metodo puo` essere dichiarato static Un metodo “statico” pùo accedere solo a variabili “statiche” non pùo utilizzare variabili d’istanza (ossia, non static) Di solito i metodi statici vengono creati per funzionalità che non hanno bisogno di uno stato (state-less) Quindi non hanno bisogno di creare oggetti Possono essere invocati usando il nome della classe Tipicamente sono metodi che ricevono i parametri ed eseguono qualche calcolo generico su essi Abbiamo visto esempi di metodi statici nella classe Math

```
Math.random()
Math.pow()
public static int somma(int x, int y) { return x+y; }
```