11 - Introduzione alla Programmazione Orientata agli Oggetti (Object Oriented Programming – OOP)

Programmazione e analisi di dati
Modulo A: Programmazione in Java

Paolo Milazzo

Dipartimento di Informatica, Università di Pisa
http://www.di.unipi.it/∼milazzo
milazzo@di.unipi.it

Corso di Laurea Magistrale in Informatica Umanistica
A.A. 2015/2016

Sommario

Programmazione imperativa (1)

Abbiamo visto come programmare utilizzando i seguenti tipi di dati:

- Tipi di dato primitivi (`int`, `double`, `char`, `boolean`, ecc...)
- Le stringhe
- Gli array

I programmi fatti fino ad ora consistevano di una sequenza di

- comandi
- strutture di controllo (cicli, scelte condizionali, ecc...)
- ed eventualmente metodi ausiliari

che consentivano di manipolare i dati per calcolare il risultato voluto.

Questo modo di programmare prende il nome di

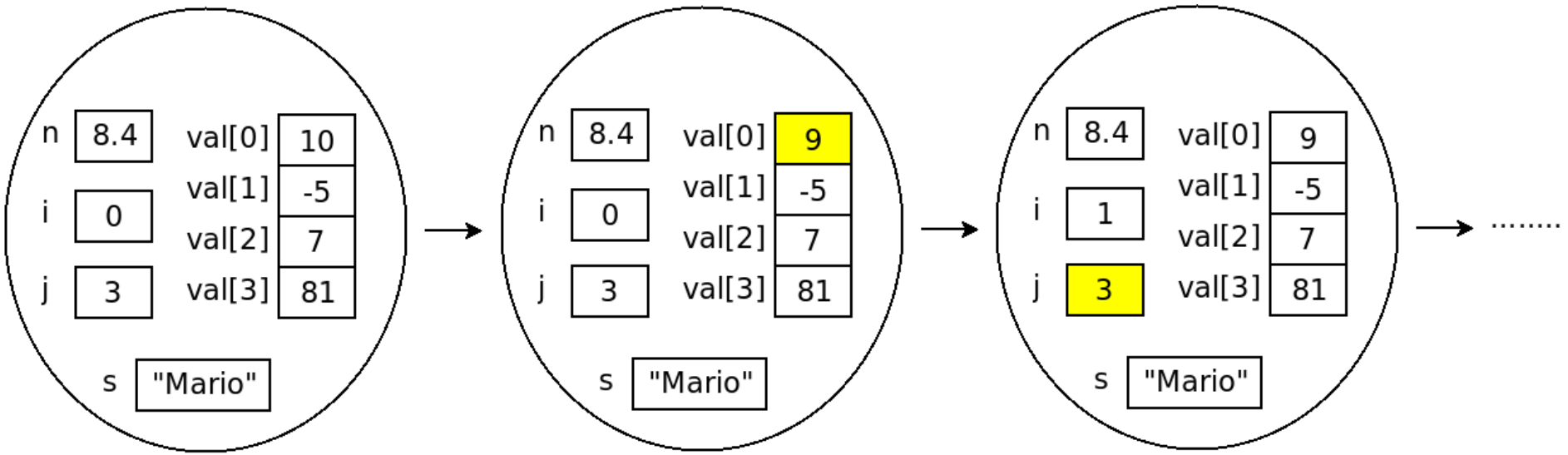
PROGRAMMAZIONE IMPERATIVA

(nota: imperativa in quanto basata su comandi)

Programmazione imperativa (2)

Nella programmazione imperativa:

- Un programma prevede uno stato globale costituito dai valori delle sue variabili
- I comandi del programma modificano lo stato fino a raggiungere uno stato finale (che include il risultato)

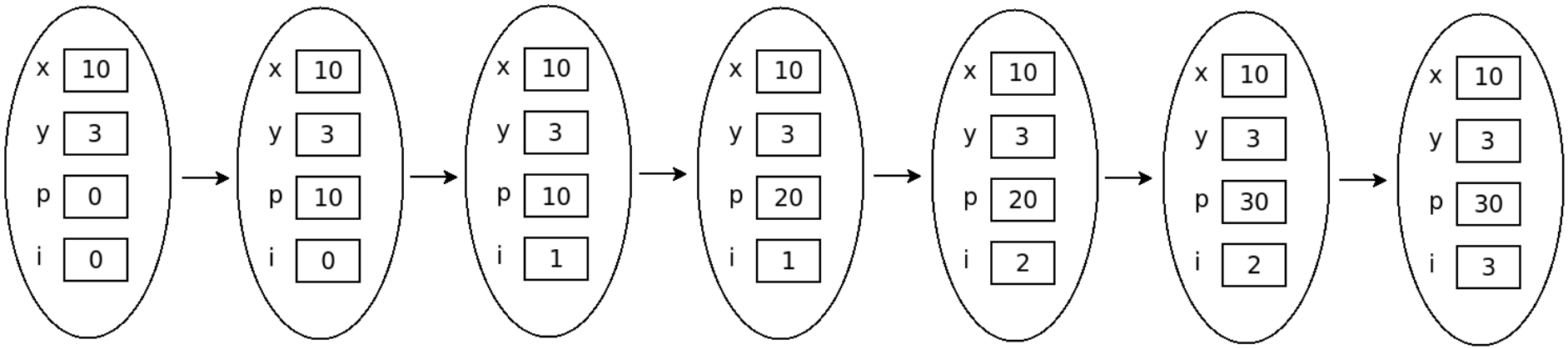


Programmazione imperativa (3)

Ad esempio, il seguente programma (che calcola il prodotto di `x` e `y`):

```
int x=10, y=3, p=0;
for (int i=0; i<y; i++)
  p+=x;
```

ha la seguente dinamica:



Programmazione orientata agli oggetti (1)

Sebbene sia possibile scrivere programmi interessanti con i tipi di dato visti fino ad ora, spesso i programmi hanno bisogno di manipolare strutture dati che rappresentano più fedelmente le entità del mondo reale.

Ad esempio, immaginate di dover scrivere programmi per la gestione di...

- Conti bancari: **ogni** conto bancario ha un proprio saldo, un proprio intestatario, una propria lista di movimenti, ecc...
- Dipendenti: **ogni** dipendente di un'azienda ha una propria matricola, un proprio stipendio, un proprio orario di lavoro, ecc...
- Parchi macchine: **ogni** automobile ha la propria targa, il proprio contachilometri, il proprio storico delle manutenzioni, ecc...
- Rettangoli: **ogni** rettangolo ha la propria base, altezza e posizione nel piano.

Scrivere un programma di questo tipo usando solo interi, array e strighe può diventare abbastanza complicato...

Programmazione orientata agli oggetti (2)

Notate che **ogni** entità del mondo reale (e.g. il conto bancario) prevede un proprio stato interno (e.g. saldo, ecc...) e delle proprie funzionalità (e.g. versamento, prelievo, ecc...)

Per questo motivo un linguaggio di programmazione ORIENTATO AGLI OGGETTI (tipo Java) fornisce meccanismi per definire nuovi tipi di dato basati sul concetto di classe

Una classe definisce un insieme di oggetti (conti bancari, dipendenti, automobili, rettangoli, ecc...).

Un oggetto è una struttura dotata di:

- proprie variabili (che rappresentano il suo stato)
- propri metodi (che realizzano le sue funzionalità)

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
  - 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)

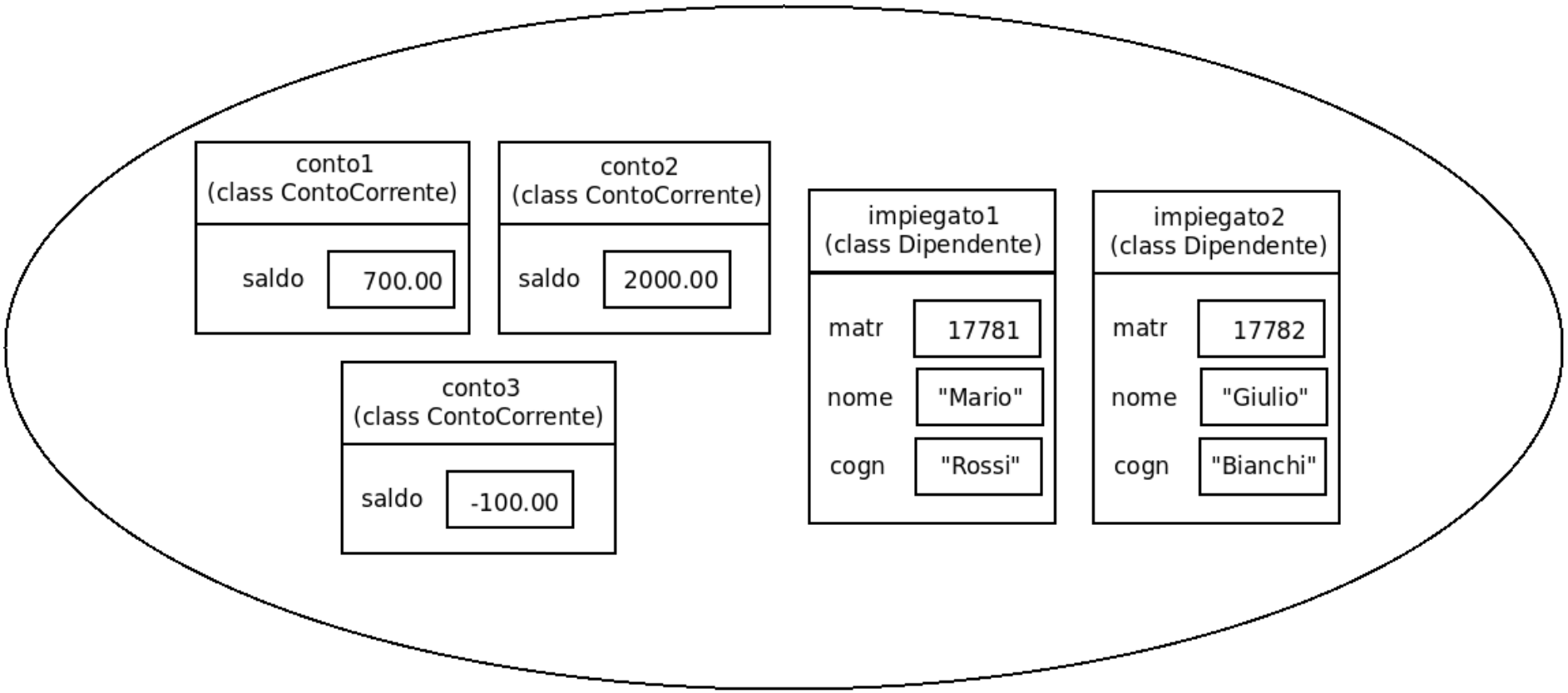
Vediamo ora come gestire più conti correnti

```
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

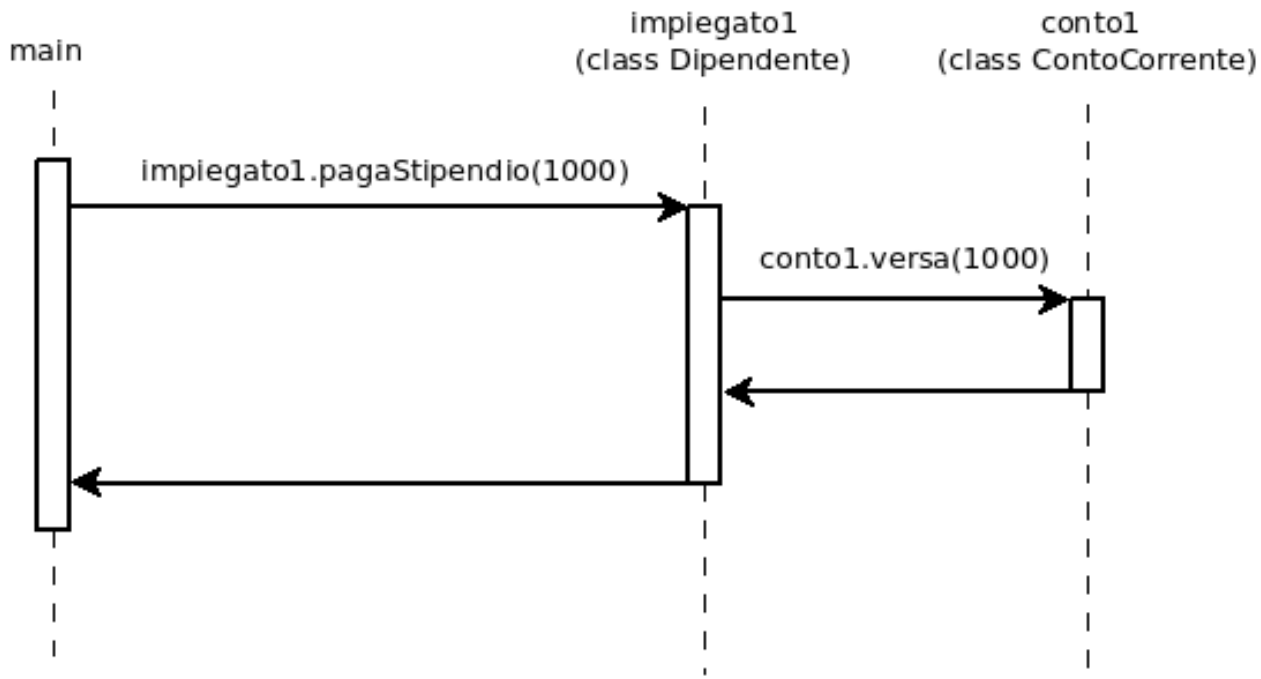


L'esecuzione di un programma a oggetti (2)

Per descrivere l'esecuzione di un programma a oggetti

- ci si può 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 efficiente e affidabile.

Sommario

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`:

```
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...