Corso di Studi in Informatica
Programmazione 1 – corso B
prof. Elio Giovannetti

Introduzione alla Programmazione in Java attraverso un esempio commentato

Modello di memoria centrale durante l'esecuzione di un programma Java

L'interprete Java (cioè il programma java.exe) mantiene 3 aree di memoria logicamente distinte:

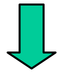
- la method area o area delle classi: contiene le definizioni delle classi usate nel programma, cioè, per ciascuna classe:
  - le descrizioni dei campi degli oggetti (ma non i campi stessi), cioè i loro nomi, tipi, visibilità e eventualmente valori iniziali;
  - i campi statici, cioè proprio i contenitori;
  - i metodi, con le istruzioni che li compongono;
- lo heap (cioè *mucchio*): contiene gli oggetti che vengono via via creati durante l'esecuzione tramite l'istruzione new, cioè i loro campi di oggetto (detti anche variabili di istanza);
- lo stack o pila: contiene, durante ogni esecuzione di una procedura (metodo o costruttore), un frame (pronuncia *freim*) contenente i parametri e le variabili locali di quella procedura (vedi più avanti).

file Conto.java

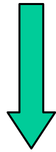
```
public class Conto {
  private static int numContiCreati;
  private double saldo;
  private double tasso;

  public Conto() {
    saldo = 0;
    tasso = 0;
    numContiCreati++;
  }

  public Conto(double saldoIniziale, double tasso) {
    saldo = saldoIniziale;
    this.tasso = tasso;
    numContiCreati++;
  }
  …
}
```

Definizione (semplice) di classe (ClassDeclaration)

visibilità (opzionale)



nome



```
public class Conto  {
   Definizioni costituenti il corpo della classe
   (ClassBodyDeclarations)
}
```

Nota Bene: secondo le nostre convenzioni di stile, la chiusa-graffa deve essere allineata con il primo carattere del costrutto sintattico (in questo caso la p di public); ciò è fatto automaticamente da TextPad.

Campi dell'oggetto, o variabili dell'oggetto o di istanza

```
public  class  Conto  {
  ...
  private double saldo;
  private double tasso;
  ...
}
```

visibilità, tipo, e nome del contenitore (cioè del campo)

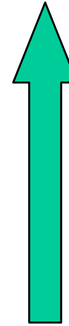
Ogni oggetto della classe **Conto** che sarà creato sullo heap (durante l'esecuzione di un programma utilizzante tale classe) sarà costituito da due campi di nome **saldo** e **tasso**, capaci di contenere numeri con la virgola, e non accessibili da procedure esterne alla classe **Conto**.

I campi di un oggetto sono detti anche variabili dell'oggetto o variabili di istanza.

Durante l'esecuzione ci saranno quindi tante variabili **saldo** e **tasso** quanti saranno gli oggetti (inizialmente nessuno).

Campi statici o variabili di classe

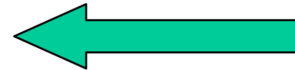
```
public class Conto {
  private static int numContiCreati;
  private double saldo;
  private double tasso;
  ...
}
```



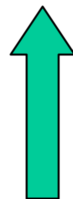
quando la classe conto sarà caricata in memoria centrale nell'area delle classi, essa avrà al suo interno una cella `numContiCreati`, di cui esisterà un'unico esemplare per tutta la classe, indipendentemente dal numero degli oggetti creati.

Costruttori

```
public class Conto {
   private static int numContiCreati;
   private double saldo;
   private double tasso;

   public Conto() {
      saldo = 0;
      tasso = 0;
      numContiCreati++;
   }
...
}
```



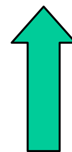
NOTA BENE
i costruttori non hanno tipo del risultato, nemmeno void.



costruttore, cioè procedura avente lo stesso nome della classe, che può venire invocata solo al momento della creazione di un oggetto; essa inizializzerà a zero i campi *saldo* e *tasso* dell'oggetto creato, e incrementerà il campo statico *numConticreati*.

Costruttori

```
public class Conto {
  private static int numContiCreati;
  private double saldo;
  private double tasso;
  ...
  public Conto(double saldoIniziale, double tasso) {
    saldo = saldoIniziale;
    this.tasso = tasso;
    numContiCreati++;
  ...
}
```



un altro costruttore: una procedura cui il chiamante deve fornire due valori, con i quali essa inizializzerà i campi *saldo* e *tasso* dell'oggetto; nella procedura (i contenitori di) tali valori avranno nomi *saldoIniziale* e *tasso*; allora per indicare il campo *tasso* dell'oggetto occorre specificare this.tasso

Costruttori: esempio di invocazione

```
public class ProvaConto {
  public static void main(String[] args) {
    Conto contoPaperone = new Conto(1000000, 6.5);
  }
}
```

tipo e nome del contenitore

nel frame del metodo main vi sarà un contenitore in grado di contenere il riferimento a (cioè l'indirizzo di) un oggetto della classe Conto

crea nello heap un oggetto della classe Conto, con i campi come descritti nella definizione di classe, restituisce l'indirizzo di tale oggetto, e inoltre esegue su tale oggetto la procedura costruttore Conto(saldoIniziale, tasso)

l'assegnazione mette in contoPaperone l'indirizzo dell'oggetto

Costruttori

Nota Bene: Le procedure costruttori possono essere invocate solo attraverso una *new*, al momento della creazione di un oggetto.

Un costruttore non può essere invocato su un oggetto preesistente per ri-inizializzarlo:

nell'esempio precedente, dopo l'istruzione

```
Conto contoPaperone = new Conto(1000000, 6.5);
```

non è possibile scrivere:

```
contoPaperone.Conto(2000000, 7.5);
```

È invece possibile scrivere:

```
contoPaperone = new Conto(2000000, 7.5);
```

ma in tal modo in realtà si crea un secondo oggetto e si perde il riferimento al primo.

Metodi

```
public class Conto {
  private static int numContiCreati;
  private double saldo;
  private double tasso;

  public Conto() {
  ...
  }
  ...
  public void deposita(double importo) {
    if(importo < 0)
      System.out.println("errore: valore < 0");
    else
      saldo = saldo + importo;
  }
  ...
}
```

Esaminiamo l'intestazione (header) di un metodo

```
public class Conto {
   ...
   public void deposita(double importo) {
   ...
   }
   ...
}
```

- visibilità → `public`
- non restituisce nulla → `void`
- nome del metodo → `deposita`
- tipo del parametro formale → `double`
- nome del parametro formale → `importo`

Il metodo *deposita* si aspetta un argomento di tipo double; esso dovrà quindi essere invocato passandogli un double, che il metodo automaticamente metterà nella cella *importo* del frame.

Segnatura di un metodo

La segnatura (in ingl. *signature*, pronuncia sigh-natiur) di un metodo è costituita da:

- il nome del metodo;
- il numero e i tipi (non i nomi) dei parametri formali.

Ad esempio, la segnatura del metodo deposita precedente è:

deposita(double)

Se si definisce un metodo

```
public void trasferisci(double importo, Conto conto) {
  ...// trasferisce un importo da this ad un altro conto
}
```

la sua segnatura è:

trasferisci(double, Conto)

Metodi: esempio di invocazione

Nota Bene: Per poter scrivere l'invocazione di un metodo, non c'è bisogno di conoscere il corpo del metodo, cioè non c'è bisogno di sapere che cosa il metodo fa: basta conoscere la sua segnatura, cioè il suo nome e quanti argomenti vuole e di che tipi.

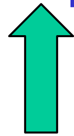
Non serve sapere i nomi dei parametri formali: essi sono di uso interno del metodo, e non interessano al chiamante.

```
public class ProvaConto {
  public static void main(String[] args) {
    Conto contoPaperone = new Conto(1000000, 6.5);
    contoPaperone.deposita(240.61);
  }
}
```

Per poter invocare deposita non c'è bisogno di sapere che il suo parametro si chiama *importo*; basta sapere che è un double, e che 240.61 è appunto un double.

Metodi: esempio di invocazione

```
public class provaConto {
  public static void main(String[] args) {
    Conto contoPaperone = new Conto(1000, 6.5);
    contoPaperone.deposita(240.61);
```



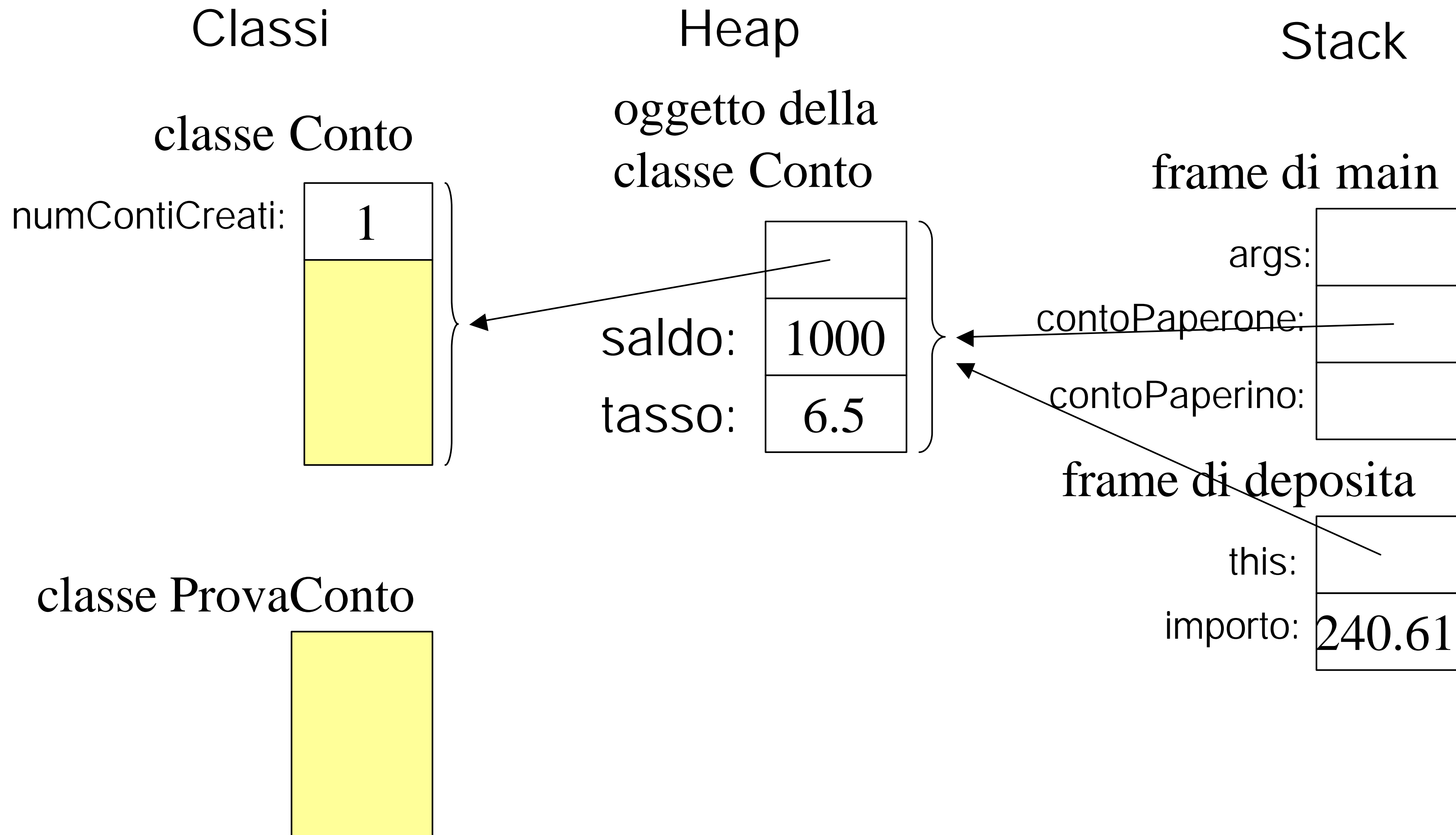
riferimento all'oggetto per cui il metodo viene invocato

argomento passato al metodo

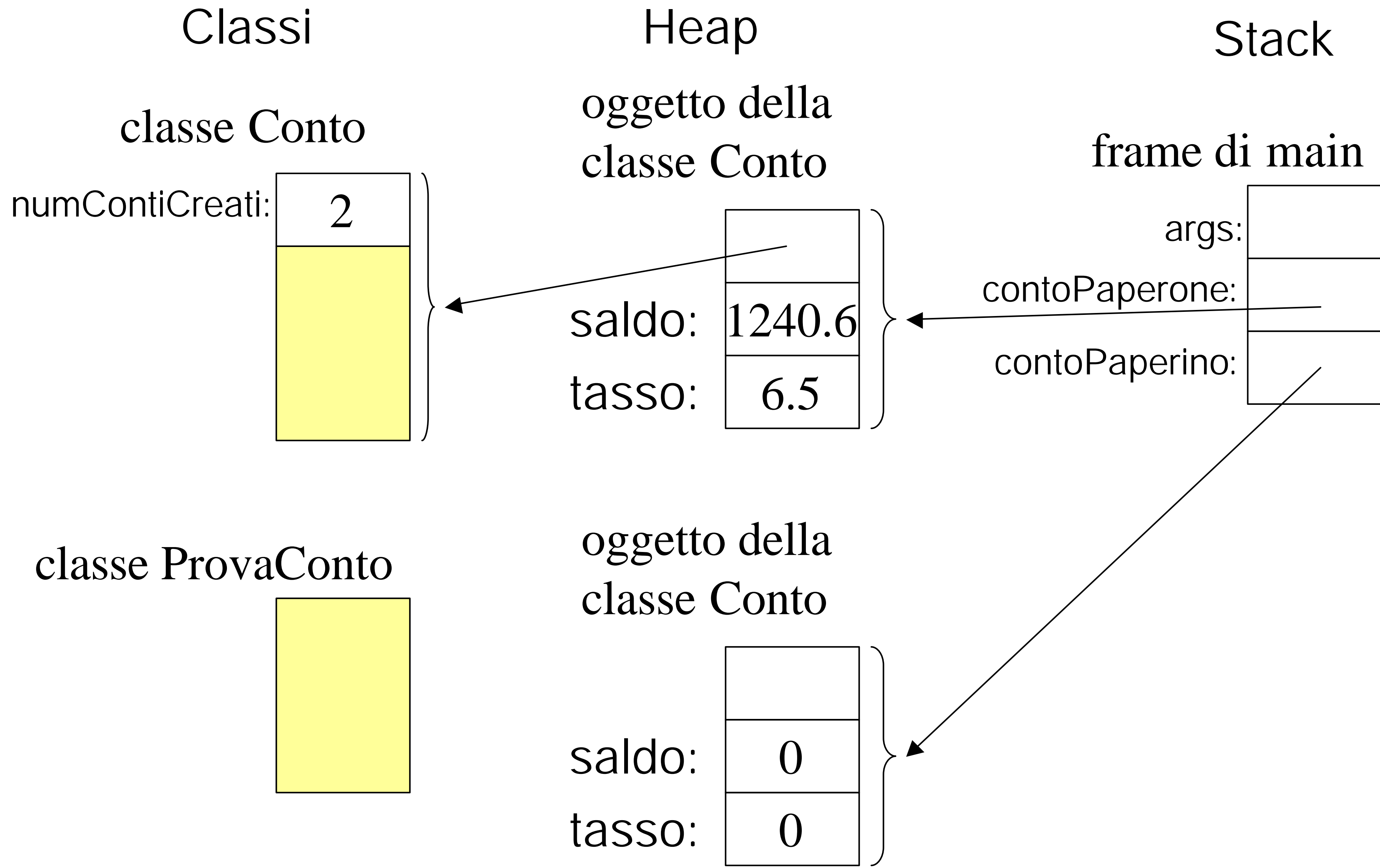
il metodo main chiama il metodo *deposita* per l'oggetto il cui indirizzo è memorizzato nella variabile contoPaperone; sullo stack, sopra al *frame* di main si impila un frame per deposita, che è un contenitore composto di una cella per l'indirizzo dell'oggetto su cui agire, e di una cella di nome *importo* in cui viene automaticamente messo l'argomento 240.61.

```
   Conto contoPaperino = new Conto(); ...
  }
}
```

Stato della memoria all'istante dell'invocazione di *deposita*



Stato della memoria dopo l'esecuzione dell'istruzione successiva di main



Corpo di un metodo

Le istruzioni che compongono il corpo di un metodo vengono eseguite ogni volta che il metodo viene invocato.

Torniamo, ad esempio, all'istante dell'invocazione
contoPaperone.deposita(240.6);
all'interno del metodo main, e quindi alla situazione della memoria rappresentata nella diapositiva 66:
negl'istanti successivi saranno eseguite le istruzioni costituenti il corpo del metodo **void deposita(double importo)** cioè

```
if(importo < 0)
  System.out.println("errore: valore < 0");
else
  saldo = saldo + importo;
```

il valore contenuto nel contenitore *importo* (che sta nel frame di *deposita*) viene inviato all'ALU, che esegue l'operazione di confronto con 0 e restituisce il risultato *true* o *false*; se *true* viene eseguita l'istruzione *println* ecc.; se *false* viene eseguita l'istruzione **saldo = saldo + importo;**

esecuzione dell'istruzione **saldo = saldo + importo;**

```
public void deposita(double importo) {
  if(importo < 0)
    System.out.println("errore: valore < 0");
  else
    saldo = saldo + importo;
}
```

metaforicamente, l'interprete java va a cercare un contenitore di nome *saldo*:

poiché nel frame di *deposita* non vi è nessun contenitore di tal nome, va a cercarlo dentro l'oggetto il cui riferimento gli è stato passato come parametro implicito *this*, cioè in questo caso l'oggetto il cui indirizzo è contenuto in *contoPaperone*;

prende quindi il contenuto della cella *saldo* di tale oggetto, prende inoltre il contenuto della cella *importo* che invece è un contenitore locale di *deposita* che sta nel suo frame, ordina alla ALU di fare la somma dei due valori, e ne rimette il risultato nella cella *saldo*.

Metodi statici

I metodi qualificati *static* non ricevono il parametro implicito *this*, e quindi non possono accederne ai campi, o tanto meno modificarli; possono accedere soltanto (eventualmente per modificarli) ai campi statici.

Essi possono (e anzi, nella buona programmazione, *devono*) essere invocati *su un nome di classe invece che su un oggetto*.

Insomma, i metodi statici sono, per così dire, metodi propri della classe, e non degli oggetti di tale classe.

Esempio: un metodo che semplicemente restituisca oppure azzeri il contatore dei conti creati, che è un campo statico, può (e anzi, nella buona programmazione, *deve*) essere dichiarato *static*.

```
public static int numContiCreati() {
  return numContiCreati;
}

public static void resetNumContiCreati() {
  numContiCreati = 0;
}
```

È buona norma stilistica, dopo aver scritto tutte le definizioni dei campi in righe successive (un campo per riga), lasciare una riga bianca prima del primo metodo, e poi separare un metodo dall'altro per mezzo di una riga bianca.

È buona norma stilistica, anche se non è obbligatorio, mettere i campi statici prima dei campi ordinari (cioè non statici), e analogamente mettere i metodi statici prima degli altri metodi.

Infine, dopo aver scritto tutte le definizioni di metodi, naturalmente si chiude il corpo della classe con una *chiusa graffa*.