12 - Gestione della Memoria nella Programmazione Orientata agli Oggetti

Programmazione e analisi di dati
Modulo A: Programmazione in Java

Paolo Milazzo

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

Corso di Laurea Magistrale in Informatica Umanistica
A.A. 2017/2018

Premessa (1)

Le lezioni di questo corso (e conseguentemente le slides) sono organizzate in modo da cercare di far comprendere i concetti chiave di ogni argomento trattato

La presentazione degli argomenti segue un "filo" che mira a massimizzare (auspicabilmente) la comprensione di tali concetti nel breve tempo delle lezioni

- Esempio motivante $\rightarrow$ soluzione specifica $\rightarrow$ generalizzazione/spiegazione

Nei libri di testo di solito la presentazione degli argomenti è strutturata in maniera diversa (più nozionistica e organica)

- Inquadramento $\rightarrow$ definizioni $\rightarrow$ motivazioni $\rightarrow$ esempi

Premessa (2)

RACCOMANDAZIONE

dopo aver seguito la lezione
rivedete gli argomenti trattati su un libro

Ciò vi sarà utile per:

- mettere in ordine i concetti visti a lezione
- capire cose che non vi sono chiare
- eventuali approfondimenti
- vedere altri esempi

Sommario

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 condi
  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 void 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`

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

Anche in questo caso sarebbe più pratico se la variabile `tasso` fosse condivisa da tutte le classi (quindi `static`)

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 sufficiente fare:

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

Note:

- Le variabili statiche possono essere riferite usando il nome della classe invece che il nome di un oggetto
- Si può comunque usare anche il nome di un oggetto (es. `cc.tasso+=0.01`)
- La variabile `tasso` può essere usata anche se non esistono oggetti di tipo `ContoCorrente`

Metodi statici

Anche un metodo può essere dichiarato `static`

Un metodo "statico" può accedere solo a variabili "statiche"

- non può utilizzare variabili d'istanza (ossia, non `static`)

```
public static int somma(int x, int y) { return x+y; }
```

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()`
- .....

Gestione memoria nella JVM (1)

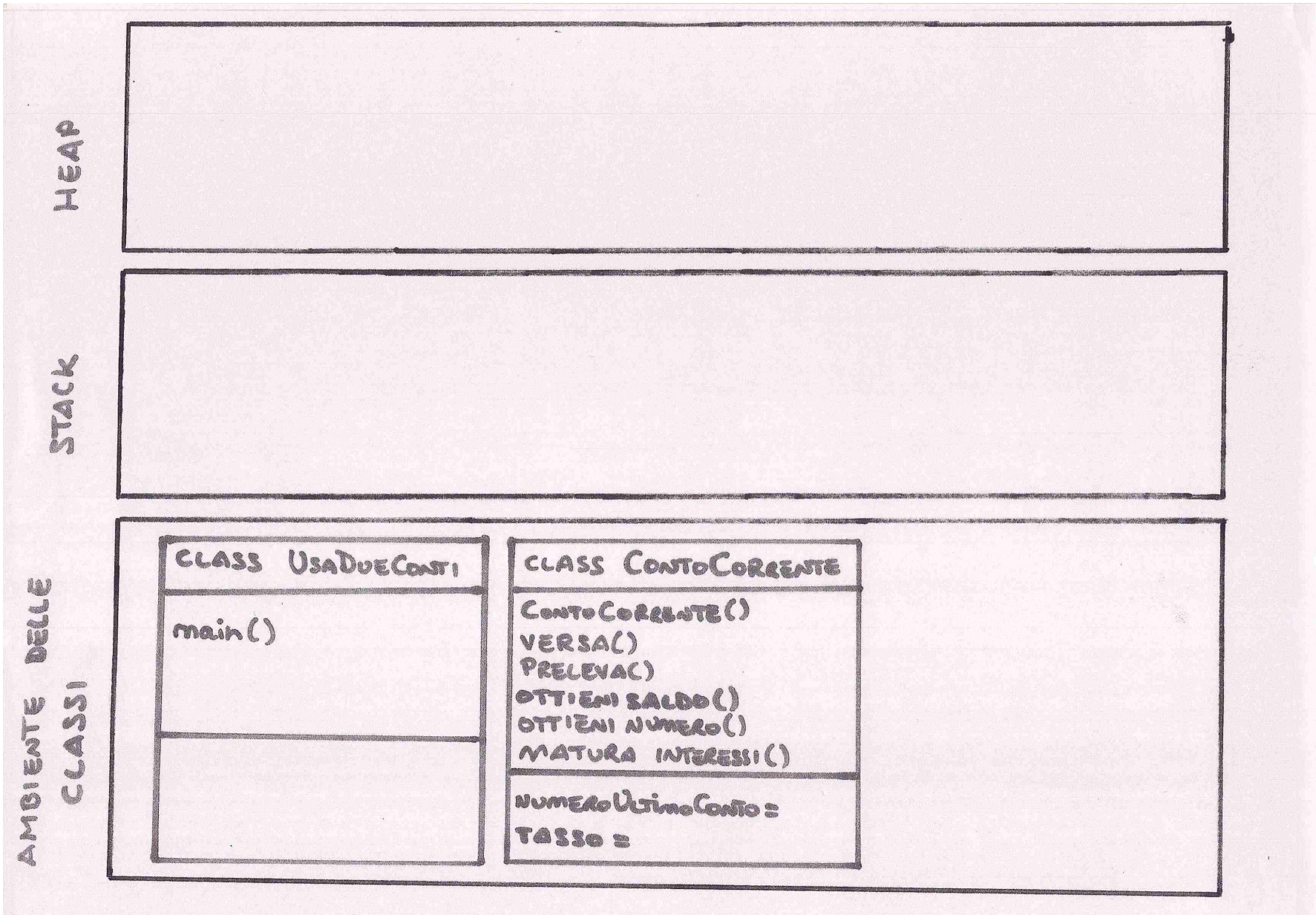
Per capire meglio come funzionano classi e oggetti diamo uno sguardo "sotto il cofano" della Java Virtual Machine (JVM)

La memoria usata dalla JVM è concettualmente divisa in tre parti

- Ambiente delle classi: area di memoria in cui vengono caricate (allocate) tutte le classi che costituiscono il programma
- Stack: area di memoria in cui vengono caricati (allocati) i record di attivazione dei metodi, e quindi tutte le variabili locali
- Heap: area di memoria in cui vengono caricati (allocati) tutti i vari oggetti creati nel programma, man mano che vengono creati.

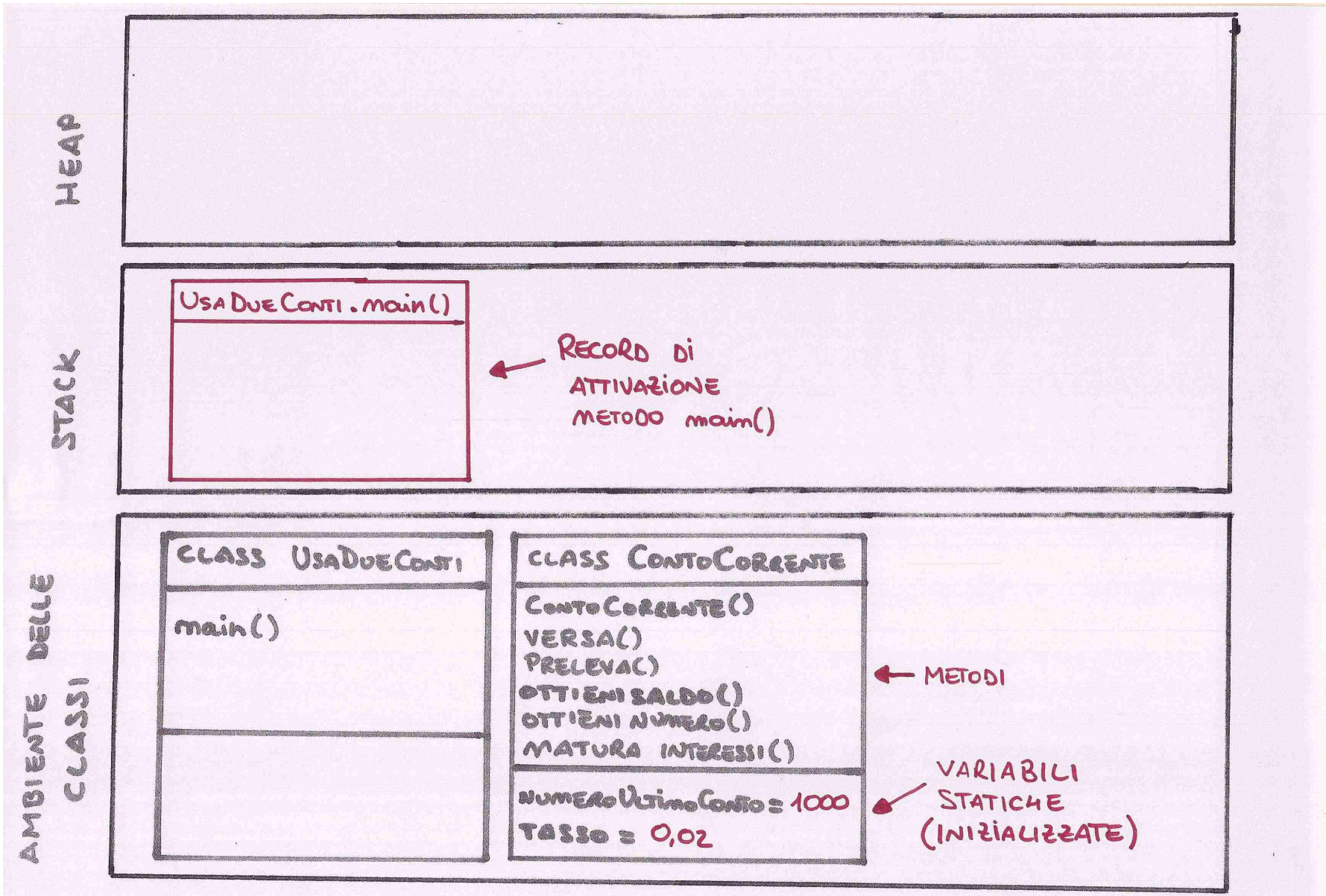
Gestione memoria nella JVM (2)

animazione memoria JVM (classi, stack, oggetti)



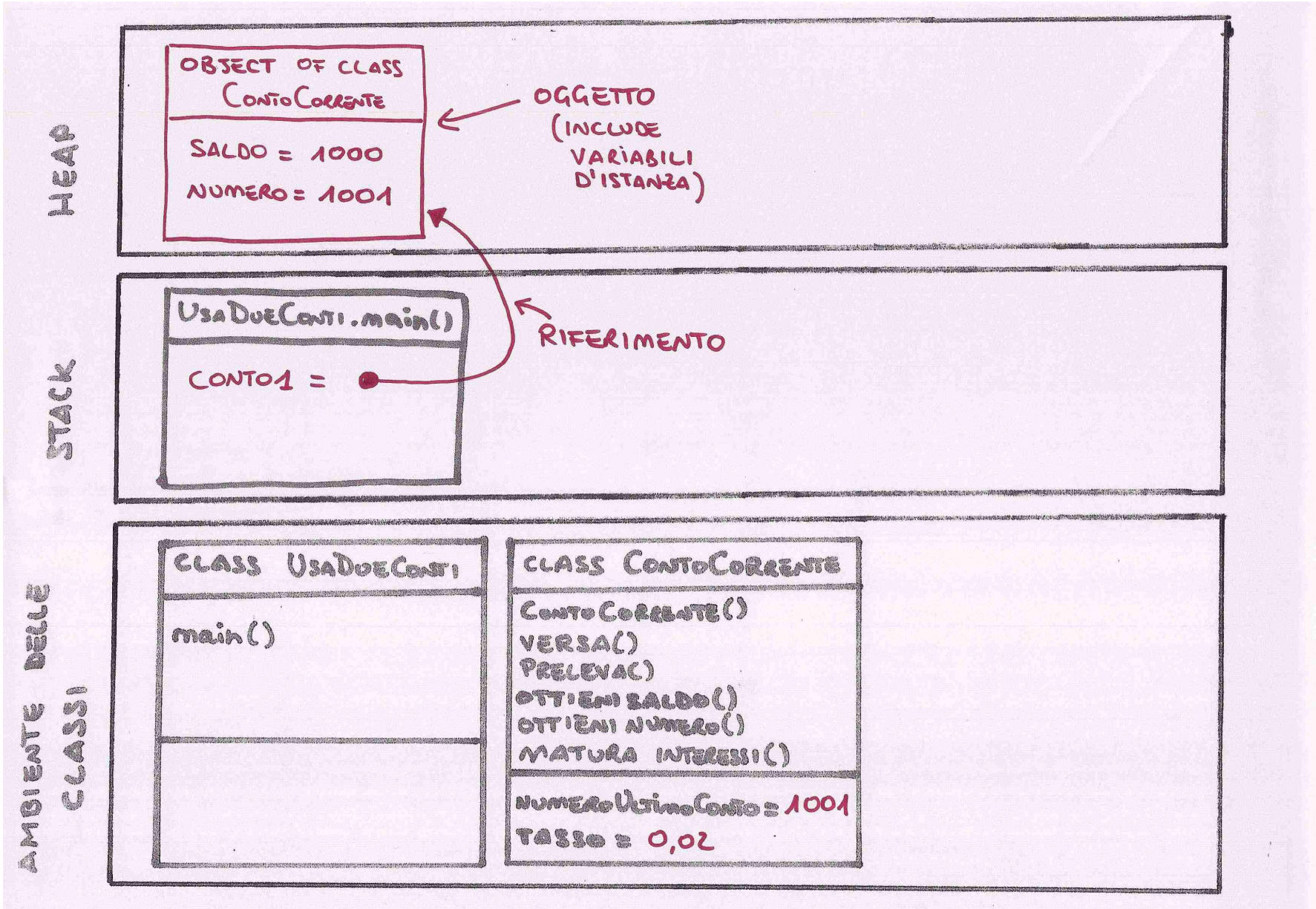
Gestione memoria nella JVM (3)

animazione memoria JVM (classi, stack, oggetti)



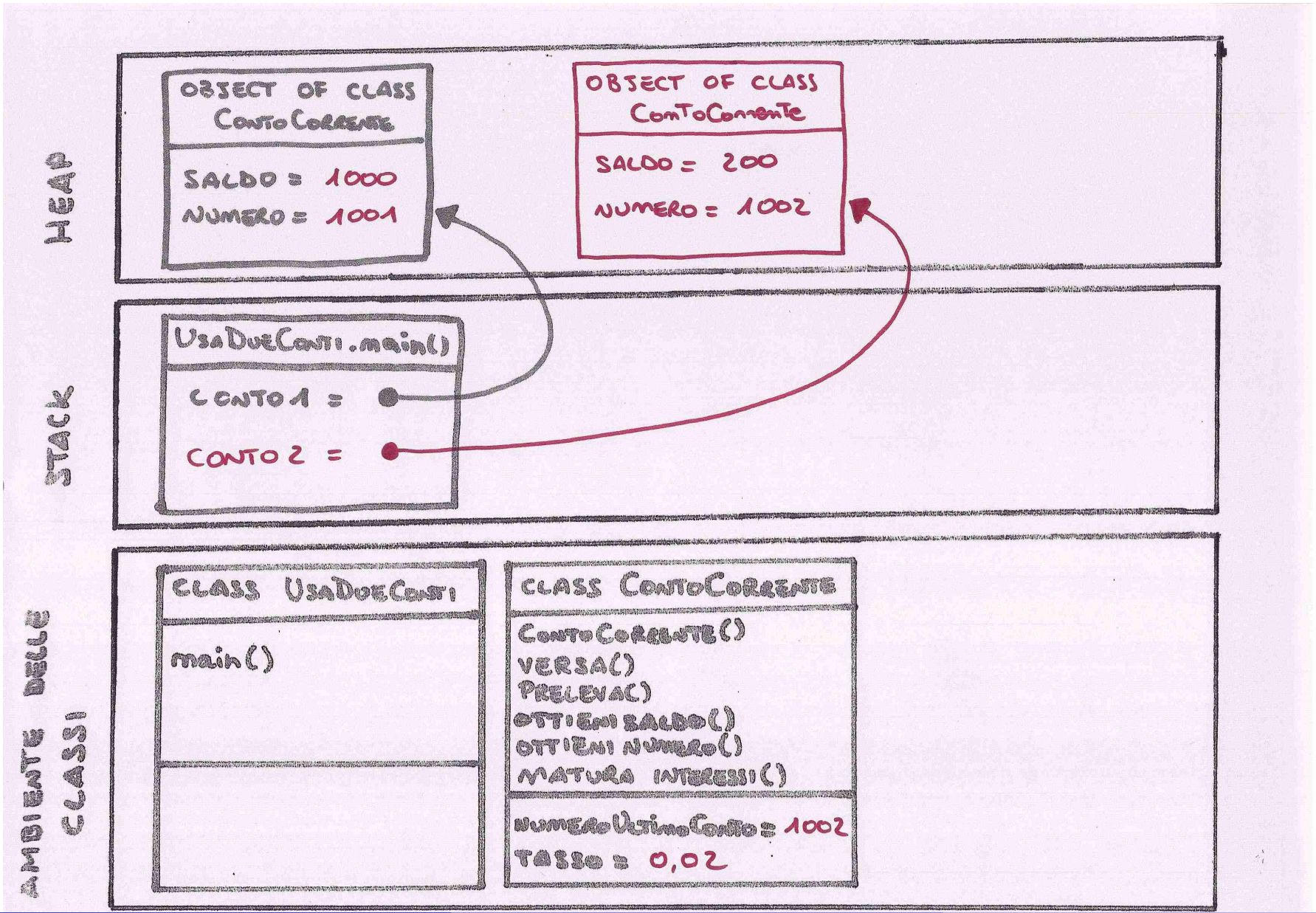
Gestione memoria nella JVM (4)

animazione memoria JVM (classi, stack, oggetti)



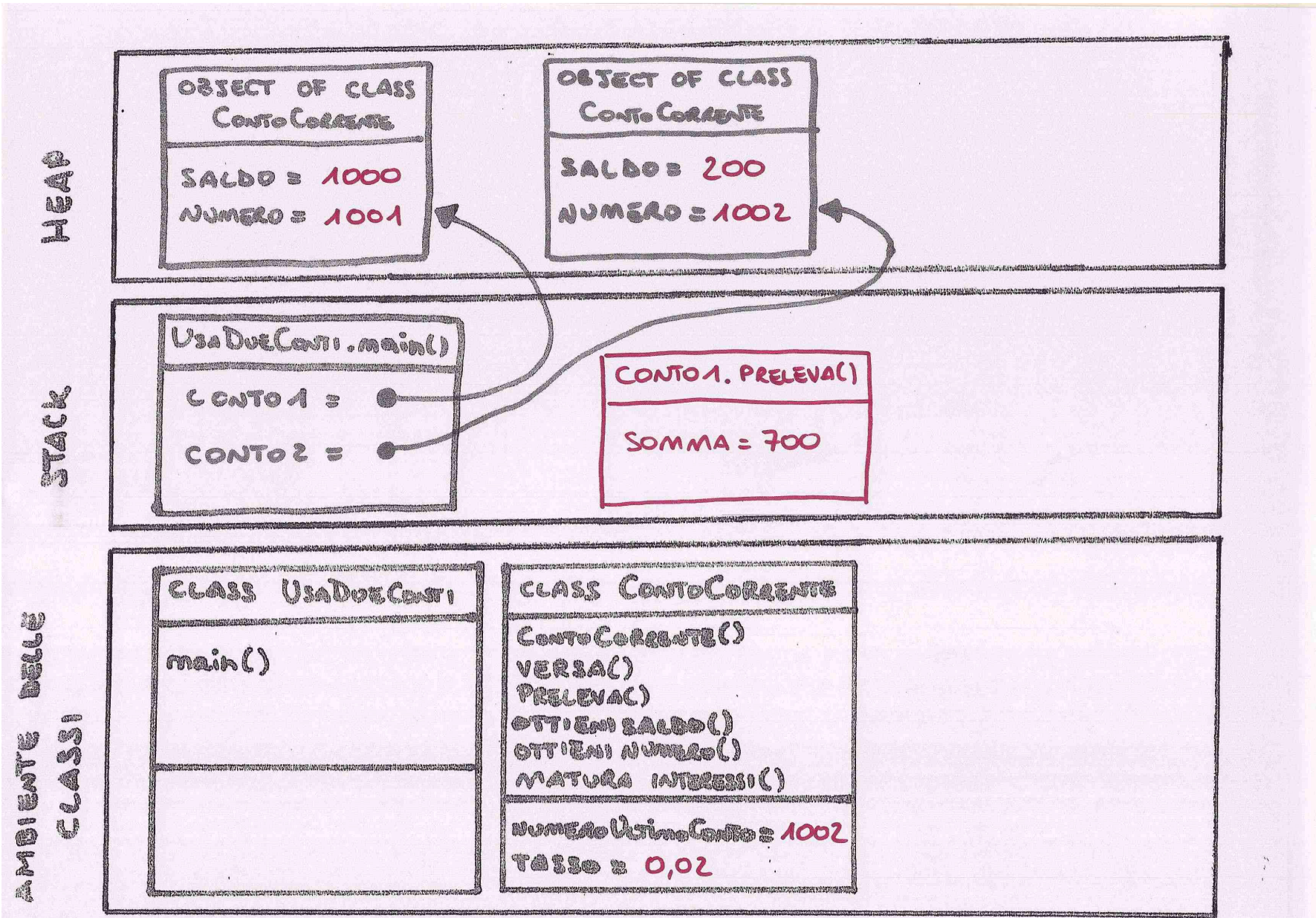
Gestione memoria nella JVM (5)

animazione memoria JVM (classi, stack, oggetti)



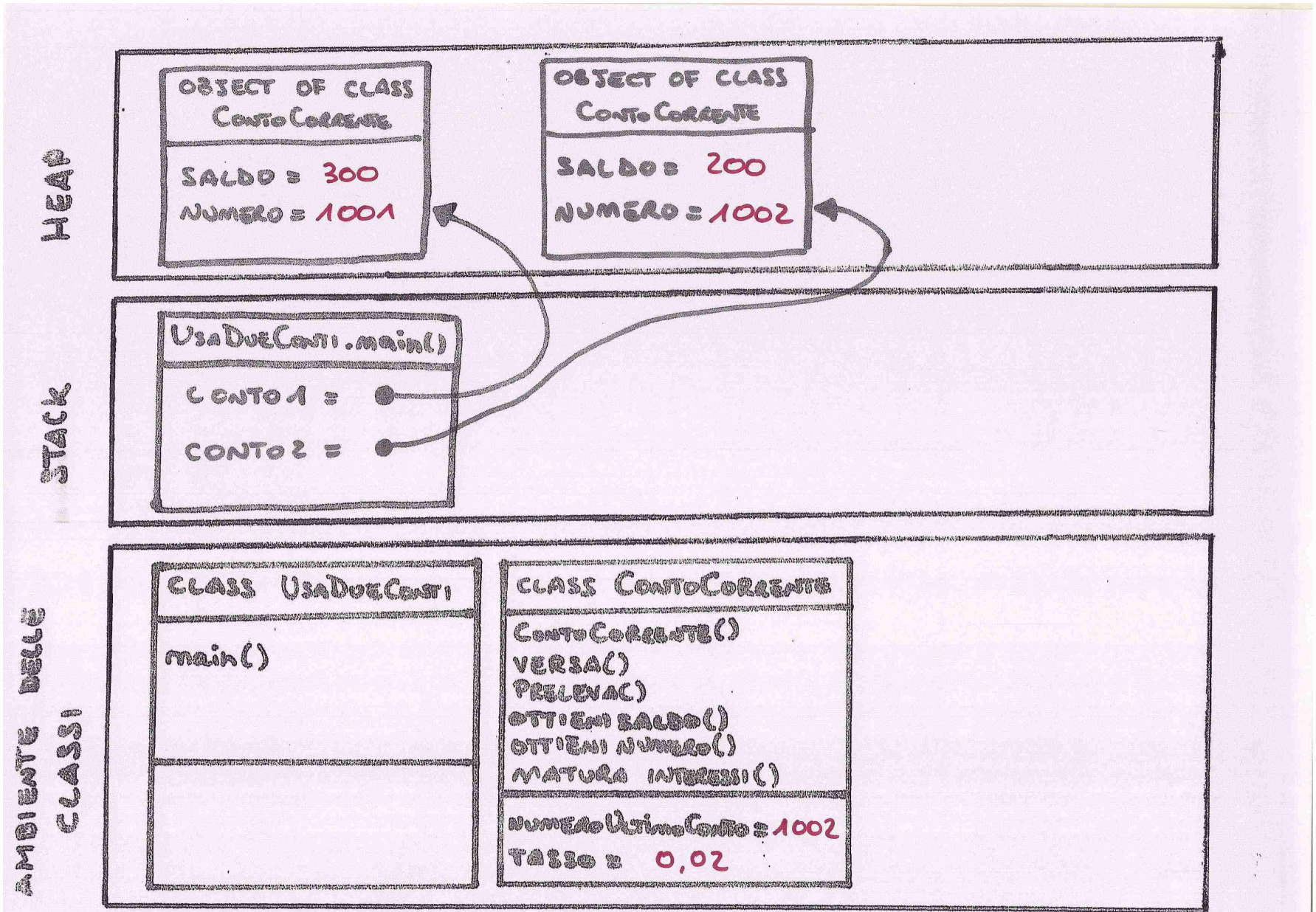
Gestione memoria nella JVM (6)

animazione memoria JVM (classi, stack, oggetti)



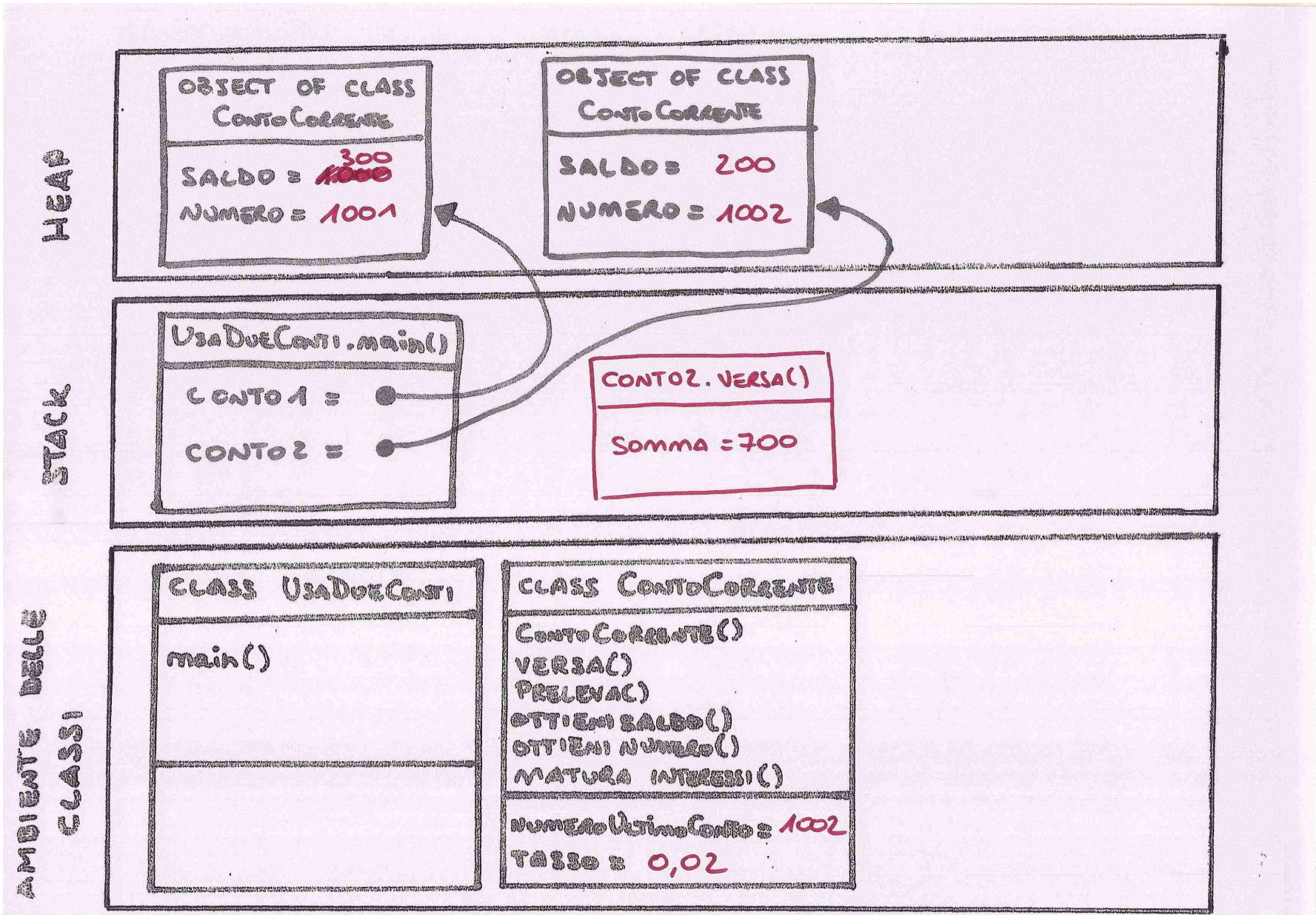
Gestione memoria nella JVM (7)

animazione memoria JVM (classi, stack, oggetti)



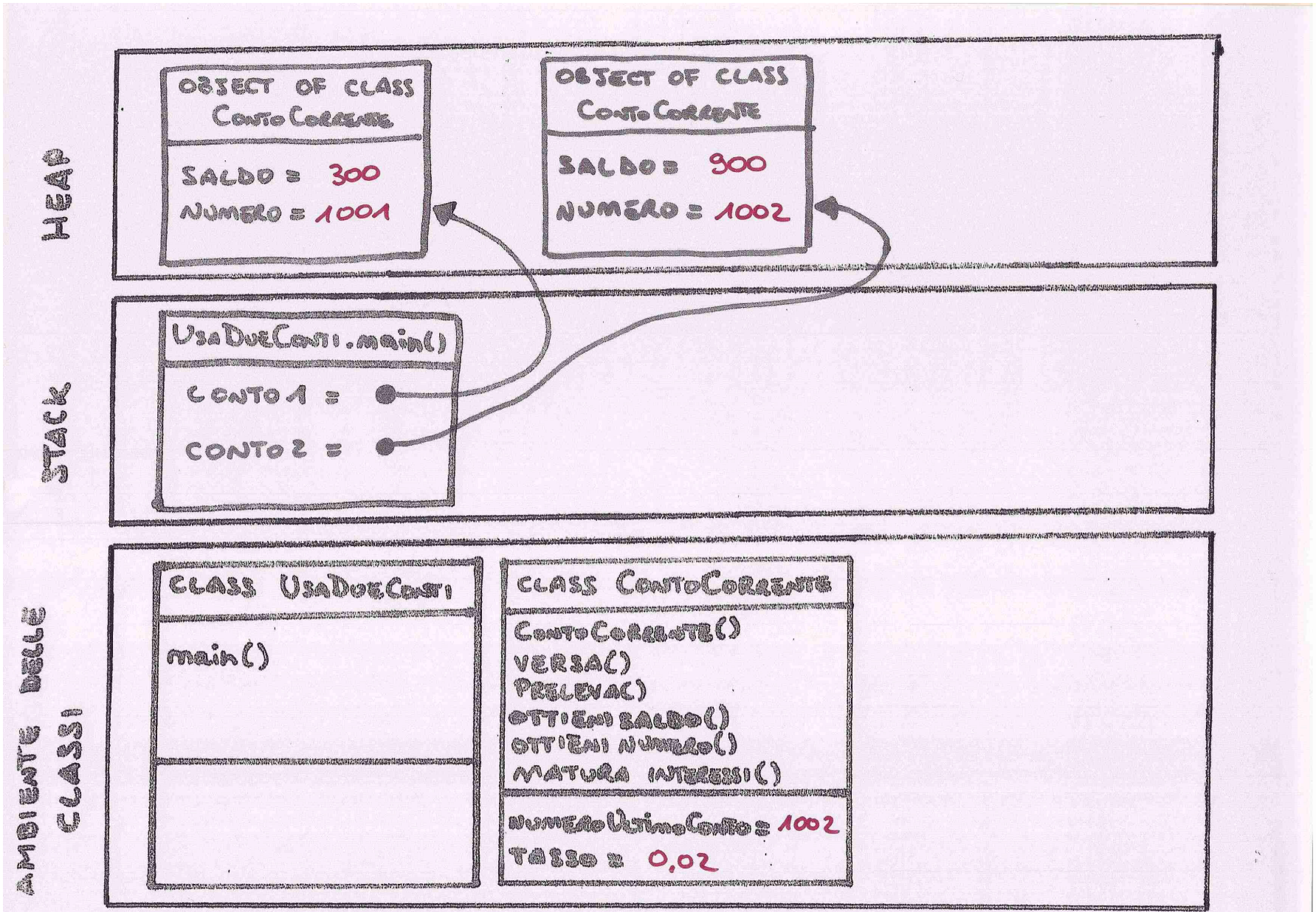
Gestione memoria nella JVM (8)

animazione memoria JVM (classi, stack, oggetti)



Gestione memoria nella JVM (9)

animazione memoria JVM (classi, stack, oggetti)



Gestione memoria nella JVM (10)

Quindi:

- Nell'ambiente delle classi
  - vengono memorizzati il codice dei metodi e le variabili statiche di tutte le classi del programma
  - sono le parti condivise dai vari oggetti della classe
  - le variabili statiche sono utilizzabili anche in assenza di oggetti
- Nello stack
  - vengono memorizzate le variabili locali dei metodi in esecuzione
  - per le variabili di tipi primitivi viene memorizzato il valore (esempio: `somma`)
  - per le variabili di tipo classe viene memorizzato un riferimento (indirizzo di memoria di un oggetto)
- Nell'heap
  - per ogni oggetto creato vengono memorizzate le variabili d'istanza (ossia, le variabili non statiche)
  - ogni oggetto nell'heap contiene anche il nome della classe di appartenenza

Sommario

Riferimenti (1)

I riferimenti meritano un approfondimento.

Abbiamo visto che una variabile di un tipo primitivo contiene direttamente il valore del dato

- La dichirazione della variabile `x` alloca la memoria necessaria per contenere un `int`

```
int x;
```

- Un assegnamento alla variabile `x` scrive un valore nella memoria precedentemente allocata

```
x = 33;
```

- L'assegnamento di `x` a `y` copia il contenuto della variabile (il valore)

```
int y = x;
```

- La modifica di `y` non modifica `x`

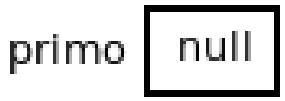
```
y = 40;
```

Riferimenti (2)

Una variabile di un tipo classe contiene invece un riferimento a un oggetto

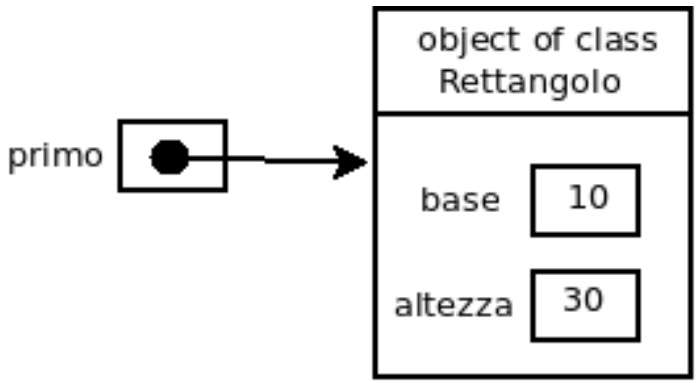
- La dichirazione della variabile `primo` di tipo `Rettangolo` alloca la memoria necessaria per contenere un riferimento (inizializzato a `null`)

```
Rettangolo primo;
```



- La creazione dell'oggetto `primo` alloca un nuovo oggetto e assegna un riferimento alla variabile

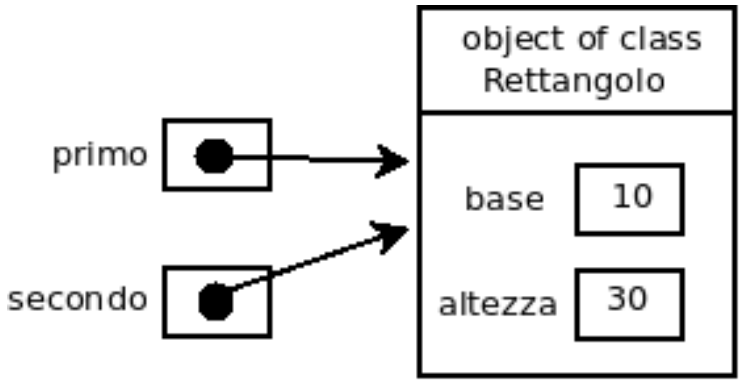
```
primo = new Rettangolo(10,30)
```



Riferimenti (3)

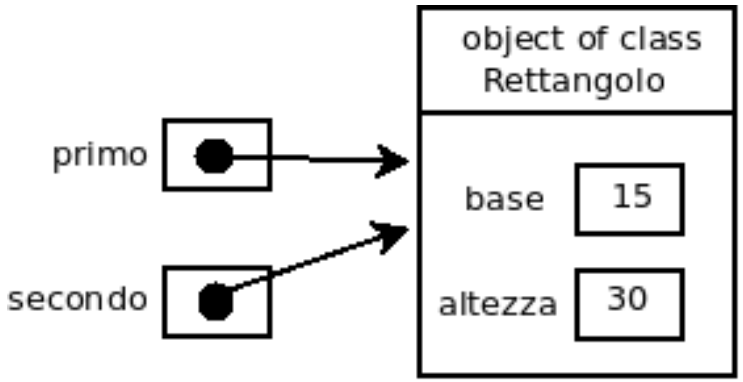
- L'assegnamento di `primo` a `secondo` copia il contenuto della variabile (il riferimento)

```
Rettangolo secondo = primo;
```



- La modifica di `secondo` modifica l'oggetto riferito anche da `primo`

```
secondo.base = 15;
```



Riferimenti (4)

Lo stesso discorso vale anche quando si passa un oggetto a un metodo come parametro

- Viene passato il riferimento
- Ogni modifica fatta all'oggetto all'interno del metodo non viene persa alla quando il metodo termina (il chiamante vedrà l'oggetto modificato)
- Lo stesso discorso vale per gli array (gli array sono in realtà oggetti!)

Riferimenti (5)

Una conseguenza del fatto che le variabili di tipo classe contengono riferimenti, è che l'opearatore di confronto == non si comporta (con gli oggetti) come uno si potrebbe aspettare...

Infatti `oggetto1 == oggetto2` vale `true` solo se `oggetto1` e `oggetto2` sono (riferimenti al) lo stesso oggetto.

Esempio:

```
Rettangolo r1 = new Rettangolo(10,12);
Rettangolo r2 = r1;
Rettangolo r3 = new Rettangolo(10,12);
```

Abbiamo che:

```
System.out.println(r1==r2); // stampa true
System.out.println(r1==r3); // stampa false
```

Riferimenti (6)

Una soluzione a questo problema può essere il metodo `equals`.

Tutti gli oggetti (capiremo perchè) dispongono di alcuni metodi di base

- Uno di questi è `equals`, e permette di confrontare due oggetti
- Lo abbiamo visto nelle stringhe

```
s1.equals(s2);
```

Nelle classi più comuni della Libreria Standard di Java, il metodo `equals` è implementato in modo da confrontare una per una tutte le variabili interne di una coppia di oggetti

Anche nelle proprie classi si può implementare tale metodo (vedremo...)

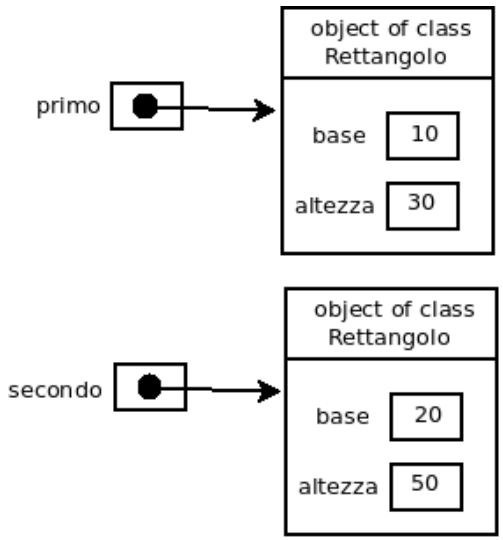
Garbage collection (1)

Un'altra conseguenza del fatto che le operazioni (lettura, assegnamento, copia, ...) su variabili di tipo classe lavorino su riferimenti è che si possono ottenere oggetti orfani (privi di riferimenti).

Ad esempio:

- supponiamo di creare due oggetti di tipo `Rettangolo`

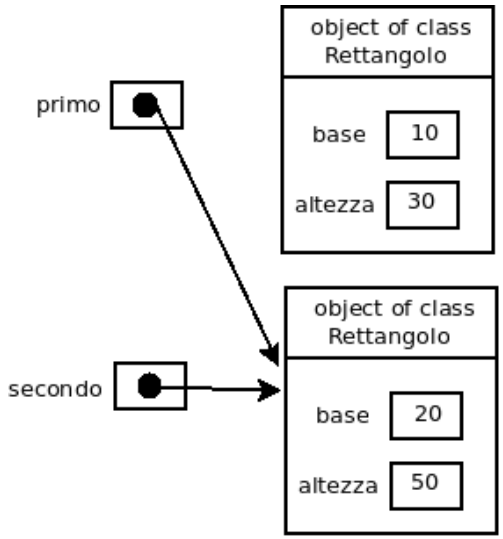
```
Rettangolo primo = new Rettangolo(10,30);
Rettangolo secondo = new Rettangolo(20,50);
```



Garbage collection (2)

- ora assegnamo `secondo` a `primo`

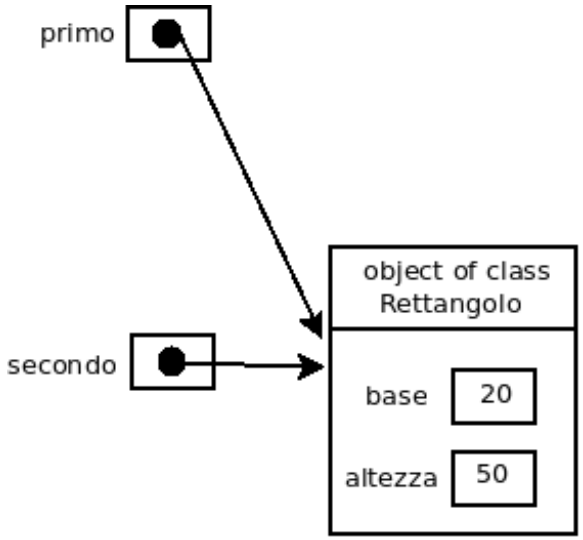
```
primo = secondo;
```



- come è possibile accedere al vecchio valore di `primo` (il rettangolo di dimensioni 10 e 30) ???
- Il vecchio oggetto è rimasto orfano... (nessun riferimento ad esso)

Garbage collection (3)

- Il vecchio oggetto non è più utilizzabile!!! (è garbage, spazzatura)
- Il linguaggio Java (come molti linguaggi moderni) prevede un meccanismo di rimozione degli oggetti privi di riferimenti detto Garbage Collector
- Il garbage collector viene eseguito periodicamente dalla Java Virtual Machine. Interrompe per un attimo l'esecuzione del programma e pulisce la memoria dagli oggetti privi di riferimenti



Sommario

Un esempio complesso (1)

Complichiamo l'esempio del conto corrente

- Realizziamo un programma di gestione di una banca

Vogliamo realizzare un programma che consente di compiere operazioni sui conti correnti di una banca tramite terminale.

- La banca potrà essere dotata di più terminali (uno per ogni sportello)
- Chi accede a un terminale deve autenticarsi (inserire username e password)
- Una volta autenticato un menù deve consentire di eseguire le varie operazioni sui conti
- Le operazioni sui conti da considerare sono simili a quelle già viste, ma assumendo due tipologie di clienti (e tassi): `"family"` e `"business"`

Un esempio complesso (2)

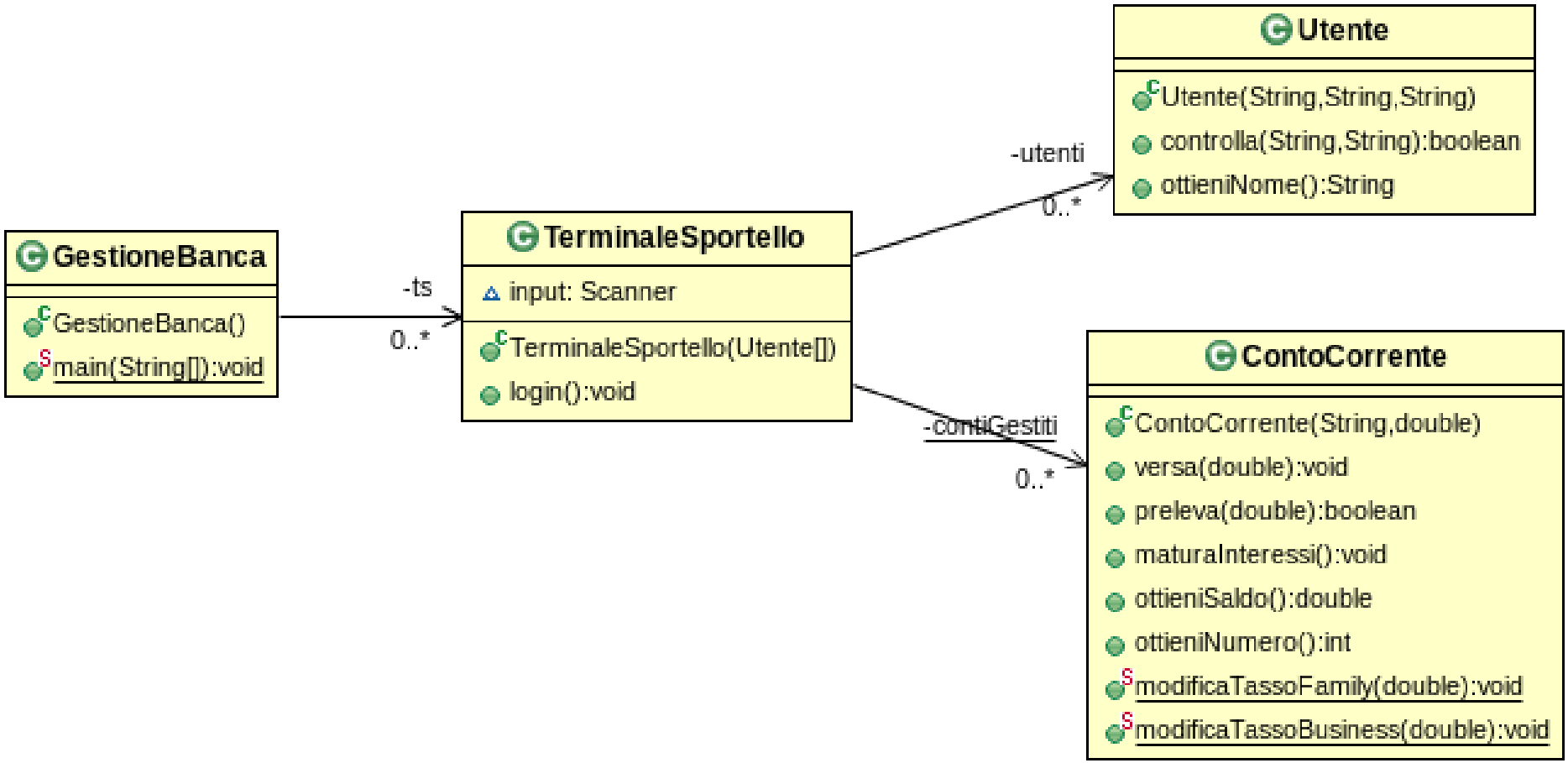
Il programma consiste di 4 classi:

- `GestioneBanca` che contiene il `main`
- `TerminaleSportello` che gestisce un terminale della banca
- `Utente` che contiene le credenziali di accesso di un utente
- `ContoCorrente` estensione della classe vista a lezione

Il `main` inizializzerà le credenziali degli utenti e consentirà di accedere a uno dei terminali (a scelta).

Il terminale consentirà di fare login (utilizzando gli oggetti `Utente`) e consentirà di eseguire le operazioni sui conti correnti (utilizzando gli oggetti `ContoCorrente`.

Un esempio complesso (3)



Un esempio complesso (4)

Per casa:

- provate di leggere il codice del programma (disponibile sulla pagina web del corso) e di capirne il funzionamento