Introduzione a Java
1

Giovanni Pardini
pardinig@di.unipi.it
www.di.unipi.it/~pardinig

Dipartimento di Informatica
Università di Pisa

Sommario

Sommario

Introduzione a Java

Java è un linguaggio object-oriented, con alcune caratteristiche

- fortemente tipato *(strongly typed)*: ogni espressione ha un tipo, che il compilatore usa per controllare la correttezza delle operazioni eseguite
- tutti gli errori sono rilevati al momento in cui avvengono, non sono possibili errori non catturati (*unchecked error*)
  - non è consentito l'accesso diretto alla memoria (cioè niente puntatori `void*`)
- gestione automatica della memoria a heap, con garbage collection
  - la deallocazione non è gestita dal programmatore, ma è il supporto a run time che si occupa di liberare la memoria non più usata

Sintassi di base

La sintassi del frammento imperativo di Java è simile a quella del C/C++

- comandi (`if`, `while`, `...`)
- le dichiarazioni di metodi (analoghi alle funzioni) e variabili
- si usano le graffe (`{...}`) per indicare l'apertura e la chiusura dei blocchi
- il numero di spazi/tabulazioni/a capo non sono importanti

- Come in C, un metodo `main` è utilizzato come punto di partenza del programma
  - viene invocato dal supporto a run-time all'inizio dell'esecuzione

Hello World!

```
1 public class Hello {
2     // entry point del programma
3     public static void main(String[] args) {
4         System.out.println("Hello World!");
5     }
6 }
```

- dichiarazione della classe `Hello`
- un metodo, `main`
  - il punto di ingresso nel programma, invocato all'inizio dell'esecuzione
  - stampa sulla console la stringa *Hello World!*

Hello World!

```
⇒1  public class Hello {
 2      // entry point del programma
 3      public static void main(String[] args) {
 4          System.out.println("Hello World!");
 5      }
 6  }
```

Dichiarazione della classe `Hello`

Hello World!

```
1 public class Hello {
2     // entry point del programma
3     public static void main(String[] args) {
4         System.out.println("Hello World!");
5     }
6 }
```

Commento, ignorato dal compilatore.
Un'altra sintassi per i commenti è

```
/* commento bla bla */
```

Hello World!

```
1 public class Hello {
2     // entry point del programma
⇒3     public static void main(String[] args) {
4         System.out.println("Hello World!");
5     }
6 }
```

Dichiarazione del metodo `main`

- un parametro, `args`, di tipo `String[]` (array di stringhe)
- `args` contiene gli argomenti passati dalla riga di comando

Hello World!

```
1 public class Hello {
2     // entry point del programma
3     public static void main(String[] args) {
⇒4         System.out.println("Hello World!");
5     }
6 }
```

Stampa la stringa *Hello World!* sulla console

- `System`: classe standard di Java
- `out`: variabile associata alla classe `System`
  - questa variabile contiene sempre un riferimento ad un oggetto che rappresenta lo standard output
- `println`: metodo di `out` per stampare una stringa (sullo standard output)

Dichiarazione di variabili e assegnamento
Sintassi

Ogni variabile locale utilizzata in un programma deve essere dichiarata prima del suo uso

- deve esserne indicato il tipo, cioè l'insieme dei valori che potrà memorizzare

```
<tipo> <ide>;              // dichiarazione
<ide> = <espressione>;     // assegnamento
<tipo> <ide> = <espr>;     // dichiarazione + assegn.
```

Esempio

```
1 int x;
2 float phi = 1.6180339887;
3 String msg = "ciao"
```

- La dichiarazione crea una associazione nell'ambiente locale tra il nome della variabile e il valore
  - se non è inizializzata, viene associata ad un valore fittizio

Tipi

Esempio: il tipo `int`

- contiene i valori $\{\ldots, -2, -1, 0, 1, 2, 3, \ldots\}$
- e fornisce le operazioni
  - + somma
  - $*$ moltiplicazione
  - - sottrazione
  - / divisione intera
  - ognuna di queste operazioni può essere applicata ad una coppia di valori `int` e ritorna un singolo valore `int`

Dichiarazione di variabili e assegnamento
Esempio

```
1 int x = 10;
2 int y = 20;
3 x = y;
4 x = x * 3;
```

1. dichiarazione della variable `x` di tipo `int`
   - gli è assegnato il valore 10
2. dichiarazione della variable `y` di tipo `int`
   - gli è assegnato il valore 20
3. assegnamento alla variabile `x` del valore associato alla variabile `y`
   - x assume il valore 20
4. assegnamento alla variabile `x` del valore di `x * 3`
   - x assume il valore 60

Tipi in Java

I tipi in Java si distinguono tra:

- **Tipi primitivi**, come `int`, `float`, `boolean`, ecc.
  Sono i tipi predefiniti del linguaggio
  - `int`: numero intero (a 32 bit)
  - `boolean`: valori `true`/`false`
  - `char`: carattere Unicode
  - `float, double`: numeri in virgola mobile di diversa precisione
- **Tipi riferimento**
  Rappresentano riferimenti ad oggetti
  - Un oggetto può essere acceduto solo possedendo un riferimento, che lo identifica univocamente

Principali operazioni aritmetiche

- `+` somma
- `*` moltiplicazione
- `-` sottrazione
- `/` divisione
- `++` incremento (solo `int`)
- `--` decremento (solo `int`)
- `+=` incremento con valore
- `-=` decremento con valore

- Applicabili solo a tipi primitivi numerici (`int`, `float`, ecc.)

Attenzione: il significato di queste operazioni cambia a seconda dei tipi a cui sono applicati

Esempio

```
int a = 10;
int b = 3;

   a / b
/* div. intera */
```

```
float c = 10;
float d = 3;

   c / d
/* divisione in
   virgola mobile */
```

Operatori booleani

- !<exp>: not
- <exp> & <exp>: and
- <exp> || <exp>: or
- <exp> ^ <exp>: xor

Esempio

```
1 boolean x = false;
2 boolean b = true;
3 boolean c = (!x || b)
```

Operatori di confronto

- `<`, `<=`, `>=` `>` minore stretto, minore o uguale, maggiore o uguale, maggiore stretto
  - applicabili solo a tipi primitivi numerici (no `boolean`)
- `==` uguaglianza
- `!=` diseguaglianza

- Gli operatori di uguaglianza e diseguaglianza (`==`, `!=`) possono essere applicati a qualunque tipo di dati Java
  - per ora, ci interessa sapere che possono essere applicati a tutti i tipi primitivi (purché "compatibili")

Controllo dei tipi

Il compilatore controlla staticamente (a tempo di compilazione) che il programma sia ben tipato

- ogni uso di una variabile deve essere conforme al tipo con cui è dichiarata

Esempio

```
1 int x = 3;
2 int y = 5;
3 boolean b = x + y;     // errore di tipo
4 boolean c = (x > b);   // errore di tipo
```

- riga 3: l'espressione `x + y` ha tipo `int`, e l'assegnamento di un valore `int` ad una variabile di tipo `boolean` non è consentito
- riga 4: non è possibile confrontare un valore di tipo `int` con un valore di tipo `boolean`

Principali comandi

La sintassi dei principali comandi Java è la stessa del C

```
if (<condizione>) {
    <C1>
} else {
    <C2>
}
```

```
while (<condizione>) {
    <corpo>
}
```

```
do {
    <corpo>
} while (<condizione>);
```

```
for (<inizializzazione>; <condizione>; <incremento>) {
    <corpo>
}
```

- <condizione> è una espressione di tipo boolean

Blocchi e visibilità dei nomi

Un blocco è una porzione di codice racchiusa tra graffe

```
{  // blocco
   ...
}
```

La visibilità del nome di una variabile indica le parti di un programma in cui il nome può essere utilizzato per riferirsi alla variabile

Una variabile dichiarata all'interno di un blocco

- è visibile solo all'interno del blocco stesso, e dei sottoblocchi
- è visibile solo dal punto della dichiarazione in poi

Vincoli (controllati staticamente dal compilatore)

- ogni variabile deve essere inizializzata prima del suo utilizzo
- non è possibile ridefinire una variabile già definita

Blocchi
Esempio 1

```
1  public static int foo(int x, int y) {
2
3    int ret = -1;
4    if (x > 0) {
5      z = 3;  // non corretto: z non ancora dichiarata
6      int z = x + y;
7      ret = ret + z;  // ok
8      p = 5;
9    }
10   return ret - z; // non corretto: z non visibile
11 }
```

Blocchi
Esempio 2

```
1  public static int foo(int x, int y) {
2
3    int q;
4    q = q + 1;  // non corretto: q non inizializzata
5
6    int p;
7    if (x > 0) {
8      p = 5 + x;
9    }
10   return p;  // non corretto:
11              // p non sempre inizializzata
12 }
```

Blocchi
Esempio 3

```
1  public static int foo(int x, int y) {
2
3    int x;  // non corretto: x gia' dichiarata
4    int p = 5;
5    if (x > 0) {
6      int p = 10;  // non corretto: p gia' dichiarata
7
8    }
9    return p;
10 }
```

Sommario

Linguaggio a Oggetti

Tipo = collezione di elementi + operazioni

Classe rappresenta un tipo di dato,

Oggetti gli elementi appartenenti ad una classe

Metodi le operazioni che possono essere eseguite sugli oggetti della classe

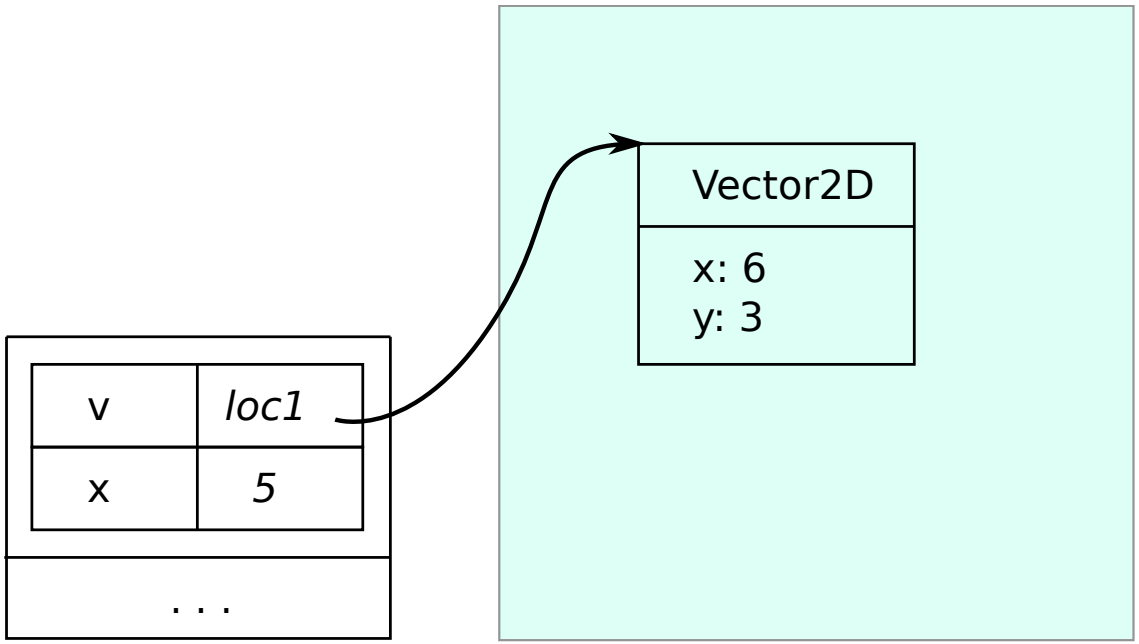
- Un particolare metodo, chiamato costruttore permette di inizializzare lo stato interno dell'oggetto al momento della creazione
- Una classe definisce:
  - variabili d'istanza = lo stato interno degli oggetti
  - metodi d'istanza = operazioni

Heap

Diversamente dai tipi primitivi, gli oggetti vengono allocati nello heap

- le variabili d'istanza sono associate all'oggetto nello heap
- ogni oggetto nello heap possiede un riferimento
- è possibile accedere ad un oggetto solo possedendone un riferimento

Heap: esempio




```
int x = 5;
Vector2D v = new Vector2D(6, 3); // creazione oggetto
```

- La classe `Vector2D` rappresenta vettori *(in senso matematico)* su un piano 2D, identificati dal valore delle coordinate x e y
  - x, y sono variabili d'istanza

Creazione di un oggetto

Per creare un oggetto si usa il costrutto new

```
new Classe (<parametri>)
```

che restituisce un riferimento ad un nuovo oggetto della classe specificata

L'esecuzione del costrutto new:

1. viene creato un nuovo oggetto nello heap
2. viene invocato un opportuno costruttore, per inizializzare lo stato dell'oggetto
3. viene restituito il riferimento all'oggetto

Esempio

```
1  Vector2D v = new Vector2D (6, 3);
```

Metodi d'istanza

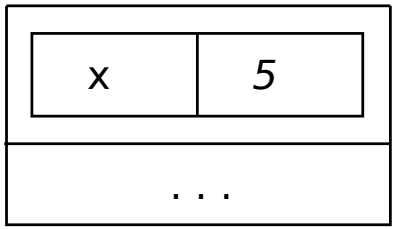
I metodi di istanza operano su oggetti della classe, e si invocano come:

```
<riferimento oggetto> . <metodo> (<parametri>)
```

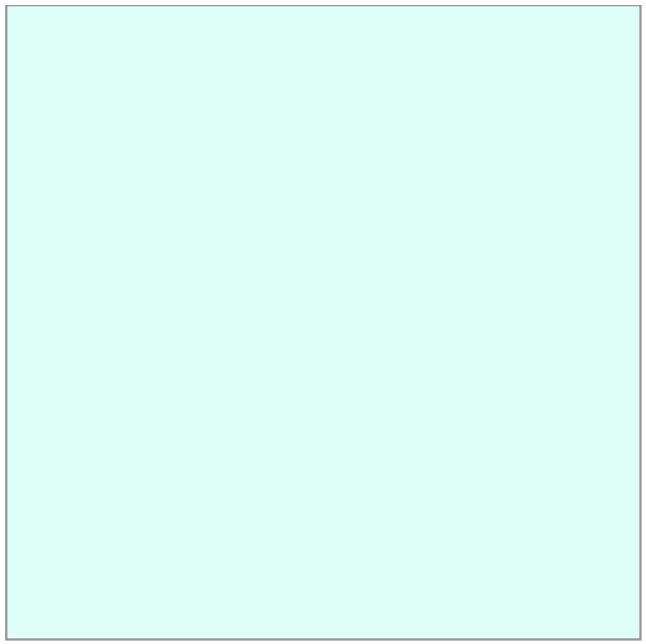
Esempio

```
1 Vector2D v = new Vector2D (6, 3);
2 int xval = v.getX();  // legge il valore della x
3 v.setX(5);            // modifica il valore della x
4 xval = v.getX();      // vale 5
```

Metodi d'istanza: esempio



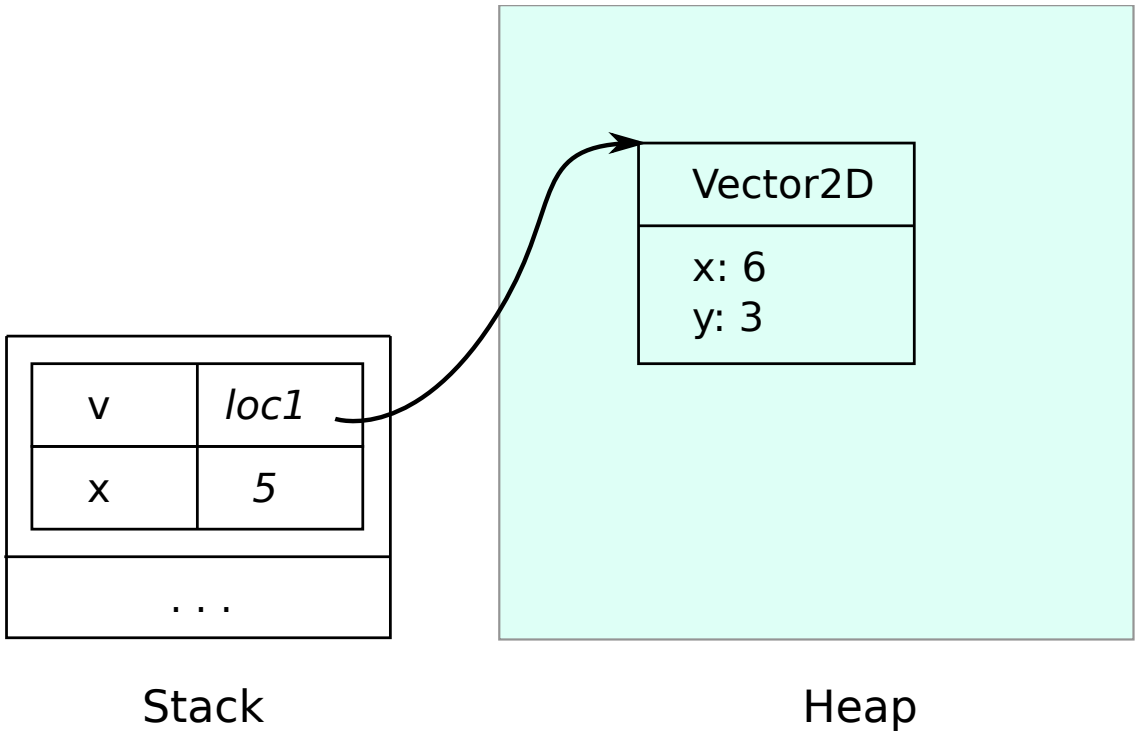
Stack



Heap

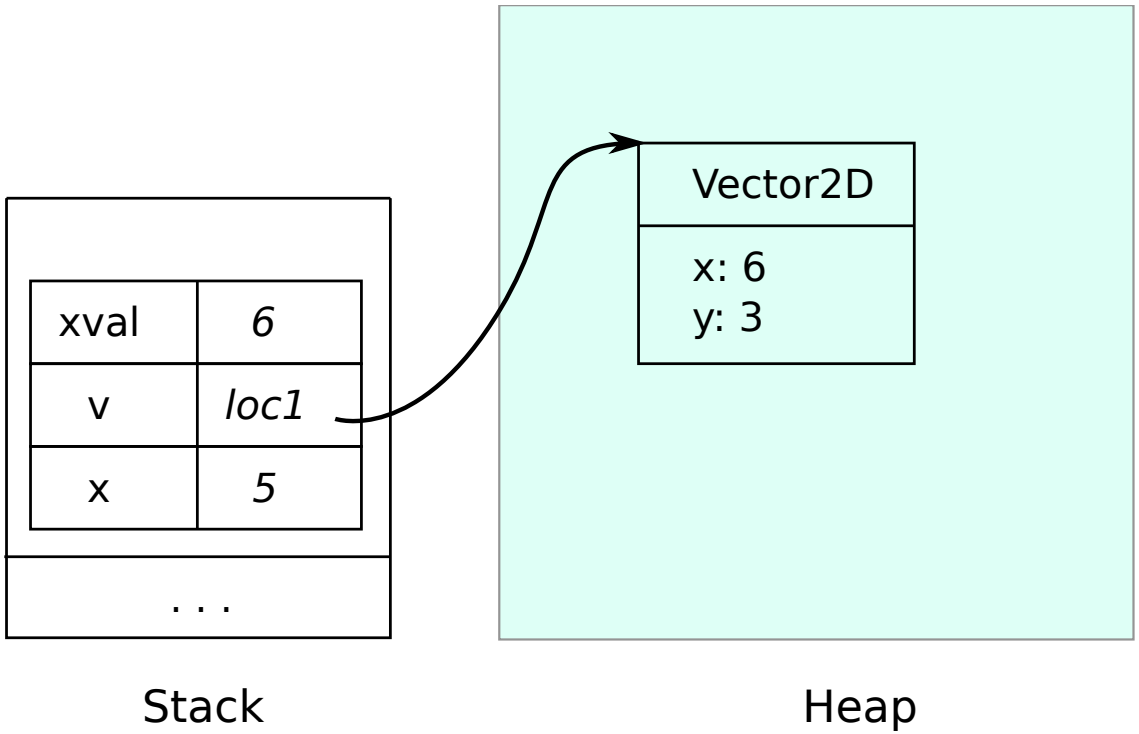
```
⇒1  int x = 5;
  2  Vector2D v = new Vector2D (6, 3);
  3  int xval = v.getX();  // legge il valore della x
  4  v.setX(5);            // modifica il valore della x
  5  xval = v.getX();  // vale 5
```

Metodi d'istanza: esempio



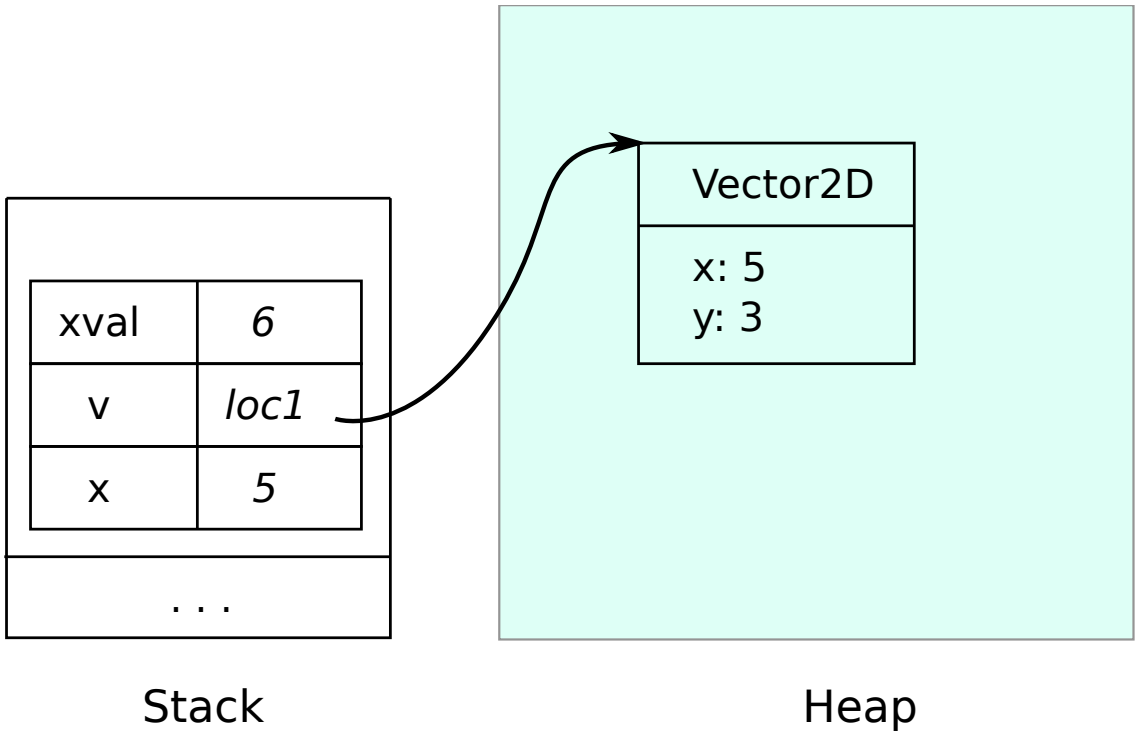

```
1  int x = 5;
⇒2 Vector2D v = new Vector2D (6, 3);
3  int xval = v.getX();  // legge il valore della x
4  v.setX(5);            // modifica il valore della x
5  xval = v.getX();      // vale 5
```

Metodi d'istanza: esempio



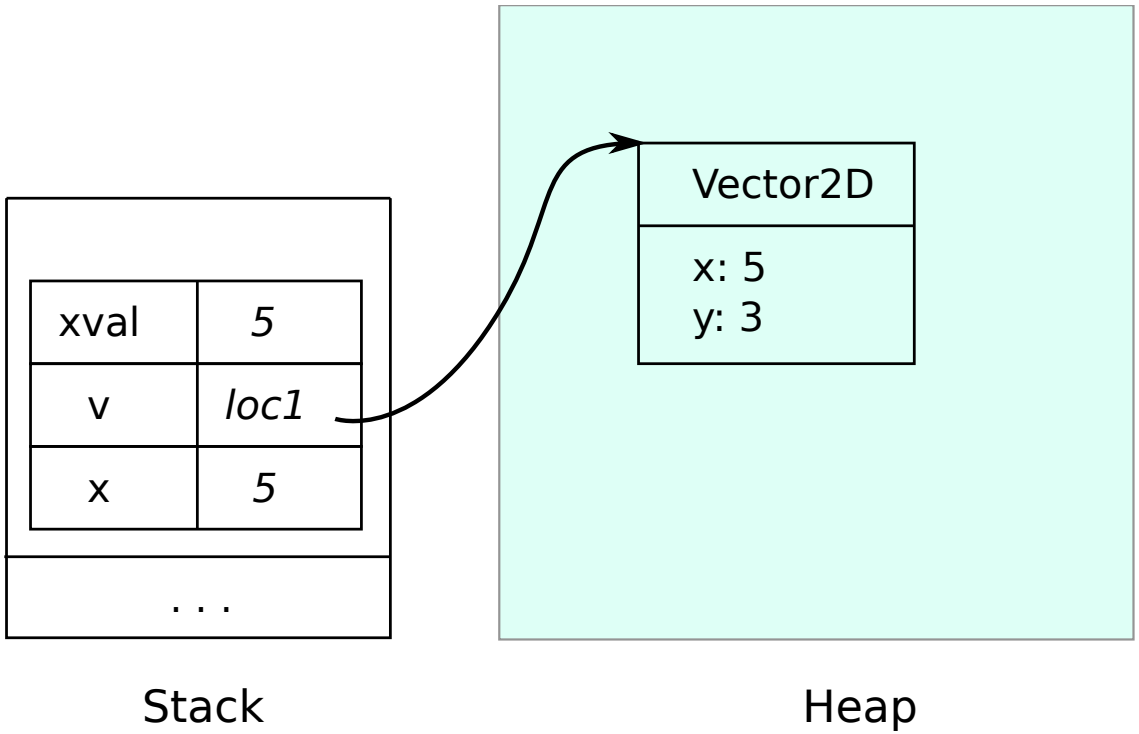

```
1 int x = 5;
2 Vector2D v = new Vector2D (6, 3);
⇒3 int xval = v.getX();  // legge il valore della x
4 v.setX(5);             // modifica il valore della x
5 xval = v.getX();       // vale 5
```

Metodi d'istanza: esempio




```
1  int x = 5;
2  Vector2D v = new Vector2D (6, 3);
3  int xval = v.getX();  // legge il valore della x
⇒4  v.setX(5);          // modifica il valore della x
5  xval = v.getX();  // vale 5
```

Metodi d'istanza: esempio




```
1  int x = 5;
2  Vector2D v = new Vector2D (6, 3);
3  int xval = v.getX();  // legge il valore della x
4  v.setX(5);            // modifica il valore della x
⇒5 xval = v.getX();  // vale 5
```

Definizione di classe

Una definizione di classe contiene le dichiarazioni di

- variabili d'istanza: lo stato dell'oggetto
- metodi d'istanza: le operazioni eseguibili sull'oggetto

```
1  public class <NomeClasse> {
2      // variabili d'istanza
3      <Tipo1> <Nome1>;
4      <Tipo2> <Nome2>;
5      ...
6
7      // metodi d'istanza
8      <MethodDecl1>;
9      <MethodDecl2>;
10     ...
11
12     // altro (variabili e metodi statici)
13 }
```

Dichiarazione di variabili e metodi d'istanza

Una variabile d'istanza può essere dichiarata come

```
[<visibilità>] <Tipo> <Nome>;
```

Esempio

```
1 public class Vector2D {
2     public int x;
3     public int y;
4 }
```

- La classe `Vector2D` contiene due variabli d'istanza
  - `x` di tipo `int` rappresenta il valore della coordinata x
  - `y` di tipo `int` rappresenta il valore della coordinata y

Accesso alle variabili d'istanza

Per accedere alle variabili d'istanza, si può utilizzare una sintassi simile a quella per i metodi:

```
<oggetto> . <variabile>
```

Esempio

```
1 Vector2D v = new Vector2D (6, 3);
2 int xval = v.x;  // legge il valore della x
3 v.x = 5;         // modifica il valore della x
4 xval = v.x;      // vale 5
```

- Ci aspettiamo lo stesso comportamento del codice visto prima

Dichiarazione di un metodo (1)

La dichiarazione di un metodo in Java è simile a quella di una funzione in C

```
[<visibilità>] <tipo ritorno> <nome metodo> (<parametri>) {
    <corpo>
}
```

Esempio

```
1 int max(int a, int b) {
2     if (a > b) {
3         return a;
4     } else {
5         return b;
6     }
7 }
```

- Funzione che prende due valori `int` e ritorna un valore `int`

Dichiarazione di un metodo (2)

```
[<visibilità>] <tipo ritorno> <nome metodo> (<parametri>) {
    <corpo>
}
```

- `<tipo di ritorno>` indica il tipo del risultato che viene restituito dal metodo
  - la keyword speciale `void` indica che il metodo non restituisce alcun risultato
- i parametri formali sono formati da una lista `(<tipo1> <nome1>, <tipo2> <nome2>,...)`, che può essere vuota
  - al momento della chiamata del metodo, i valori passati per i parametri vengono legati ai rispettivi nomi, in modo da essere usabili dentro il corpo del metodo
- le variabili dichiarate nel corpo del metodo sono dette variabili locali al metodo
  - non sono visibili all'esterno del metodo

Dichiarazione di un metodo (3)

```
[<visibilità>] <tipo ritorno> <nome metodo> (<parametri>) {
    <corpo>
}
```

- all'interno del `<corpo>` si usa un comando

  ```
  return <expr>;
  ```

  per terminare l'esecuzione del metodo, restituendo il valore (ottenuto dalla valutazione di) `<expr>`
  - ogni cammino di esecuzione dentro il metodo (che non termina lanciando una eccezione) deve ritornare un valore
- per metodi con tipo di ritorno `void`, un comando

  ```
  return;
  ```

  può essere usato per terminare l'esecuzione del metodo in punti arbitrari al suo interno

Dichiarazione di un metodo (4)

In un metodo d'istanza si può accedere a tutte le variabili d'istanza

Esempio

```
1  public class Vector2D {
2      public int x;
3      public int y;
4
5      // legge il valore della variabile d'istanza x
6      public int getX() {
7          return x;
8      }
9
10     // assegna il valore nx alla variabile d'istanza x
11     public void setX(int nx) {
12         x = nx;
13     }
14 }
```

- Il nome x nei metodi si riferisce implicitamente alla variabile d'istanza

Costruttore

Ogni classe definisce sempre almeno un costruttore, per inizializzare le variabili d'istanza

Un costruttore è dichiarato in modo simile ad un metodo d'istanza:

```
[<visibilità>] NomeClasse (<parametri formali>) {
    <corpo>
}
```

- non è specificato il tipo di ritorno
  - non deve ritornare alcunché, solo inizializzare l'oggetto
- il nome corrisponde al nome della classe

- Se nessun costruttore è dichiarato, il linguaggio crea implicitamente un costruttore senza parametri che inizializza tutte le variabili d'istanza al loro valore nullo (dipende dal tipo)

Costruttore: esempio

```
1  public class Vector2D {
2      public int x;
3      public int y;
4
5      // costruttore
6      public Vector2D (int initx, int inity) {
7          x = initx;
8          y = inity;
9      }
10     ...
```

Vector2D: implementazione

```
1  public class Vector2D {
2    public int x;
3    public int y;
4
5    //costruttore
6    public Vector2D (int initx, int inity) {
7      x = initx;
8      y = inity;
9    }
10   // legge il valore della variabile d'istanza x
11   public int getX() {
12     return x;
13   }
14   // assegna il valore di nx alla variabile d'istanza x
15   public void setX(int nx) {
16     x = nx;
17   }
18 }
```

Operatore `this`

Dentro i metodi d'istanza è possibile utilizzare l'operatore `this` per ottenere un riferimento all'oggetto corrente

Esempio

```
public class Vector2D {
  public int x;
  public int y;

  // costruttore
  public Vector2D (int x, int y) {
    this.x = x;
    this.y = y;
  }
  ...
```

Nel corpo del costruttore:

- `x`, `y`: si riferiscono ai *parametri* del costruttore
- `this.x`, `this.y`: si riferiscono alle *variabili d'istanza* dell'oggetto

Riferimento null

Il valore speciale `null` può essere associato ad una variabile di tipo oggetto per indicare che la variabile non riferisce alcun oggetto

Dato che `null` non riferisce alcun oggetto, non è possibile invocarci metodi d'istanza né accedere alle variabili d'istanza (non esistono)

```
1  Vector2D v = null;
2  v.setX(10);          // errore a run-time
```

- Di default, tutte le sue variabili d'istanza di tipo oggetto sono inizializzate al valore `null`
  - eventualmente, il costruttore può modificarne il valore

Confronto tra oggetti

L'operatore di confronto == applicato a tipi oggetto, confronta i riferimenti e non gli oggetti stessi

Esempio

```
Vector2D v1 = new Vector2D(3,7);
Vector2D v2 = new Vector2D(3,7);
Vector2D v3 = v1;

boolean a = (v1 == v2);  // false
boolean b = (v1 == v3);  // true
```

- Per confrontare il contenuto degli oggetti si deve definire un metodo apposito

Confronto tra oggetti: esempio

Per confrontare il contenuto degli oggetti possiamo definire un metodo

```
1 public class Vector2D {
2   ...
3   public boolean confronta (Vector2D v1) {
4       return this.x == v1.x && this.y == v2.y;
5   }
```

Esempio

```
Vector2D v1 = new Vector2D(3,7);
Vector2D v2 = new Vector2D(3,7);
Vector2D v3 = v1;

boolean a = (v1 == v2);           // false
boolean b = (v1.confronta(v2));   // true
```

Metodi e variabili statici

Metodi e variabili statici sono associati ad una classe

- sono sempre accessibili, anche se nessun oggetto della classe è mai stato creato

Per dichiarare una variabile/metodo statico si aggiunge la keyword `static` alla dichiarazione

```
// variabile statica
static <tipo> <nome>;

// metodo statico
static <tipo ritorno> <nome> (<parametri>) {
  <corpo>
}
```

- I metodi/variabili statiche sono sempre dichiarati dentro una classe

Campi statici: esempio

```
1  public class Num {
2      // variabile statica
3      public static double phi = 1.6180339887;
4
5      // metodo statico
6      public static int gcd(int n, int d) {
7          while (n != d) {
8              if (n > d) n = n - d;
9              else d = d - n;
10         }
11         return n;
12     }
13 }
```

- Il metodo gdc calcola il massimo comune divisore tra due numeri interi

Accedere ai campi statici

Per riferirsi a un campo (variabile/metodo) statico è necessario specificare la classe a cui appartiene

```
// accesso ad una variabile statica
<nome classe> . <nome variabile>

// chiamata di metodo statico
<nome classe> . <nome metodo> ( <parametri attuali> )
```

Esempio: il metodo `gcd` può essere invocato come

```
1 int a = 6;
2 int b = 15;
3 int x = Num.gcd(a, b); // chiamata di metodo
4 double k = Num.phi;  // accesso alla variabile statica
```

Metodo `main`

Quando un programma viene lanciato, il supporto a run-time invoca il metodo statico `main` della classe specificata

```
1 public class <nome classe> {
2   public static void main(String[] args) {
3     <corpo>
4   }
5 }
```

Semantica della chiamata di metodo

Quando un programma viene lanciato, il supporto a run-time invoca il metodo `main`

Alla chiamata di un metodo, l'esecuzione del metodo corrente viene sospesa fino al completamento del metodo chiamato

- se il tipo di ritorno non è `void`, quando il metodo chiamato termina restituisce un valore al chiamante

- Durante l'esecuzione di un programma Java, in ogni istante è in esecuzione uno e un solo metodo

Passaggio dei parametri

Il passaggio dei parametri in Java avviene sempre per valore

- il valore del parametro attuale viene copiato nella variabile del parametro formale
- per i riferimenti ad oggetto, viene copiato il riferimento

Esempio: definizione

```
1 public Vector2D {
2   ...
3   public static void azzeraX(Vector2D v) {
4      v.x = 0;  // azzera la x dell'oggetto riferito da v
5   }
```

Esempio: uso

```
1  Vector2D v1 = new Vector2D(5, 3);
2  Vector2D.azzeraX(v1);
3  int x = v1.getX();  // vale 0
4  int y = v1.getY();  // vale 3
```

Invocare un metodo statico
Esempio completo

```
1  public class Num {   // dichiarazione di classe
2    public static int gcd(int n, int d) {
3      while (n != d) {
4        if (n > d) n = n - d;
5        else d = d - n;
6      }
7      return n;
8    }
9  }
10
11 public class Principale {  // dichiaraz. classe
12   public static void main(String[] args) {
13     int a = 6;
14     int b = 15;
15     int x = Num.gcd(a, b);  // chiamata metodo
16     System.out.println(x);
17   }
18 }
```

Campi statici

Per riferirsi ad un campo statico *dall'interno della stessa classe* non è necessario specificare il nome della classe stessa

```
1  public class Num {   // dichiarazione di classe
2    public static int gcd(int n, int d) {
3      while (n != d) {
4        if (n > d) n = n - d;
5        else d = d - n;
6      }
7      return n;
8    }
9
10   public static void prova(int x) {
11     int z = gcd(x, 123);  // chiamata metodo
12     System.out.println(z);
13   }
14 }
```

- Questo vale anche per i metodi d'istanza

Classe `String` (1)

La classe `String` è usata per rappresentare sequenze di caratteri

```
String msg = "Hello";
```

- La variabile `msg` riferisce un oggetto `String`

La classe `String` fornisce alcuni metodi d'istanza utili

- `void` *`length`*`()`
  restituisce la lunghezza della stringa (numero di caratteri)
- `char` *`charAt`*`(int index)`
  restituisce il carattere in posizione `index`

Classe `String` (2)

Per controllare se due stringhe sono uguali si usa il metodo d'istanza

```
boolean equals(s1)
```

che confronta la stringa corrente con `s1`

- ritorna `true` se e solo se le stringhe sono uguali

Concatenazione

L'operatore `+` può essere utilizzato anche per concatenare due stringhe

Esempio

```
1 String s1 = "abc";
2 String s2 = s1 + "def";
3 boolean b = s2.equals("abcdef"); // true
```

- Le stringhe Java non sono modificabili
  - Es.: l'operatore `+` ritorna un *altro* oggetto stringa

Sommario

File sorgenti

Un file sorgente Java <NomeClass>.java ha la seguente struttura

```
1 // import delle classi usate da librerie esterne
2 import <fully-qualified-class-name>;
3 ...
4
5 public class <NomeClasse> {
6   ...
7 }
```

- Ogni file deve contenere una sola definizione di classe, *il cui nome corrisponde a quello del file*

Compilazione ed esecuzione (1)

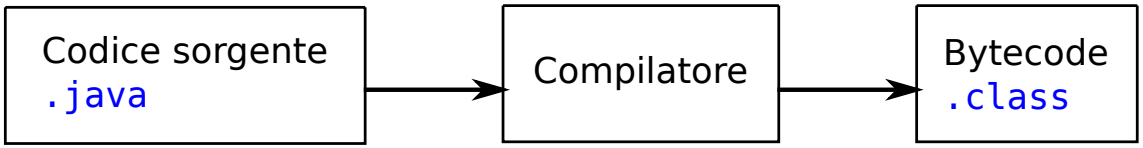
I programmi Java sono compilati in un linguaggio intermedio chiamato bytecode, da eseguire su una macchina virtuale

- Java Virtual Machine (JVM)

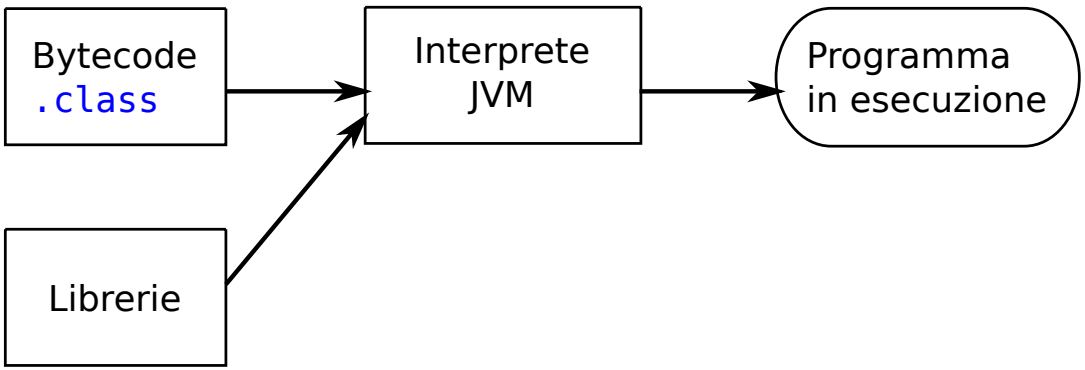
- I programmi Java (compilati in bytecode) sono portabili su molte architetture diverse, senza necessità di modifiche al codice sorgente
  - purché sia disponibile un interprete per la JVM sull'architettura considerata

Compilazione ed esecuzione (2)

Compilazione



Esecuzione

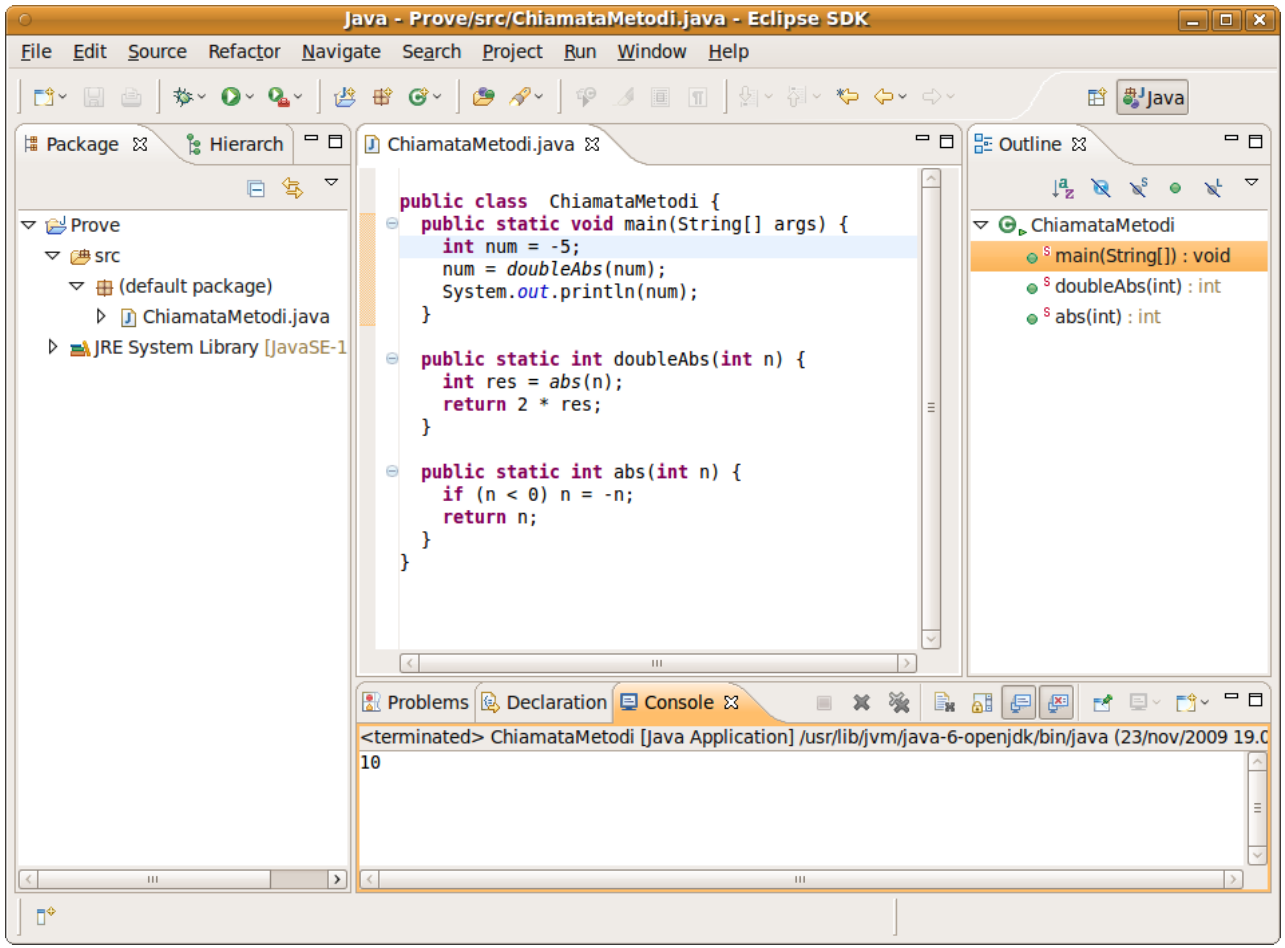


- la compilazione di un file sorgente `.java` produce un file `.class`
- il file `.class` contiene la traduzione del programma in bytecode il linguaggio intermedio di Java
- l'interprete esegue il bytecode, simulando l'esecuzione della macchina virtuale Java (*Java Virtual Machine, JVM*)

Ambiente di sviluppo

Per sviluppare in Java si può utilizzare l'*ambiente di sviluppo* open-source **Eclipse** (*www.eclipse.org*)

Eclipse



Sommario I

Sommario II

Sommario

Comando if

```
1 if (<cond>) {
2     <C1>
3 } else {
4     <C2>
5 }
```

Se la condizione <cond> è *vera* viene eseguito <C1>, altrimenti viene eseguito <C2>

- <cond> deve essere di tipo `boolean`
- il ramo `else` può essere omesso se vuoto

Esempio

```
1 int a;
2 if (k < 0) {
3     a = -k;
4 } else {
5     a = k;
6 }
```

Comando while

```
1 while (<cond>) {
2     <corpo>
3 }
```

Prima di ogni iterazione viene valutata la guardia <cond>, quindi:

- se <cond> è vera, viene eseguito il corpo del ciclo, quindi si passa alla successiva iterazione
- se <cond> è falsa, si esce dal ciclo

- Il corpo del ciclo è eseguito fintanto che la guardia rimane vera.

Esempio

```
1 int k = 0;
2 while (k < 5) {
3     System.out.println(k);
4     k++;
5 }
```

Comando do-while

```
1 do {
2     <corpo>
3 } while (<cond>);
```

Dopo ogni iterazione (esecuzione del corpo del ciclo) viene valutata la guardia <cond>, quindi:

- se <cond> è vera, si esegue un'altra iterazione
- se <cond> è falsa, si esce dal ciclo

Esempio

```
1 int k = 0;
2 do {
3     System.out.println(k);
4     k++;
5 } while (k < 5);
```

Comando `for`
Sintassi

```
1 for (<init>; <cond>; <incr>) {
2     ...
3 }
```

- `<init>`: comando di inizializzazione del ciclo
- `<cond>`: guardia del ciclo
- `<incr>`: comando di incremento

Comando `for`
Esempio

```
1 for (<init>; <cond>; <incr>) {
2     ...
3 }
```

Esempio

```
1 int totale = 0;
2 for (int i = 0; i < 10; i++) {
3     totale = totale + i;
4 }
```

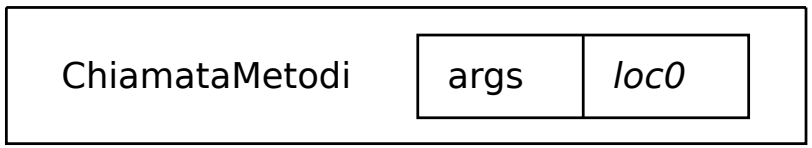
- A differenza del C, in Java è possibile (e consigliato) dichiarare la variabile `i` direttamente dentro il comando `for`
  - la `i` è visibile solo all'interno del ciclo

Semantica della chiamata di metodo: esempio

Metodi statici

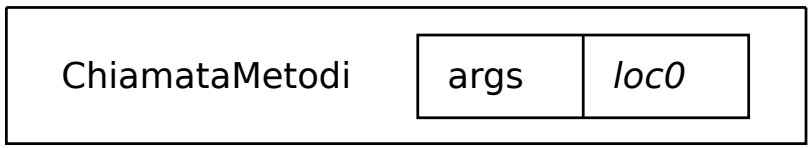
```
1  public class  ChiamataMetodi {
2    public static void main(String[] args) {
3      int num = -5;
4      num = doubleAbs(num);
5      ...
6    }
7
8    public static int doubleAbs(int n) {
9      int res = abs(n);
10     return 2 * res;
11   }
12
13   public static int abs(int n) {
14     if (n < 0) n = -n;
15     return n;
16   }
17 }
```

Evoluzione dello stato: lo stack (1)



- `ChiamataMetodi` è il nome della classe
- `args` punta ad un array nello heap
  - *loc*0 è una locazione nello heap

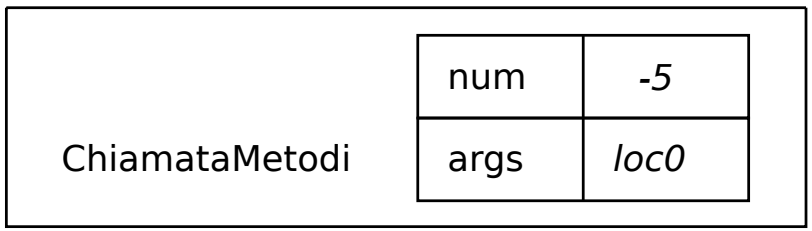
Evoluzione dello stato: lo stack (1)



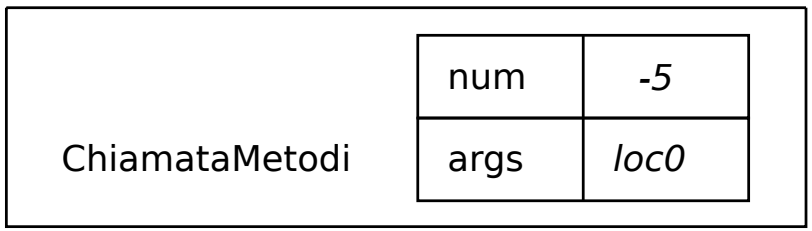
- `ChiamataMetodi` è il nome della classe
- `args` punta ad un array nello heap
  - *loc*0 è una locazione nello heap

- Esecuzione di `int num = -5`

Evoluzione dello stato: lo stack (2)

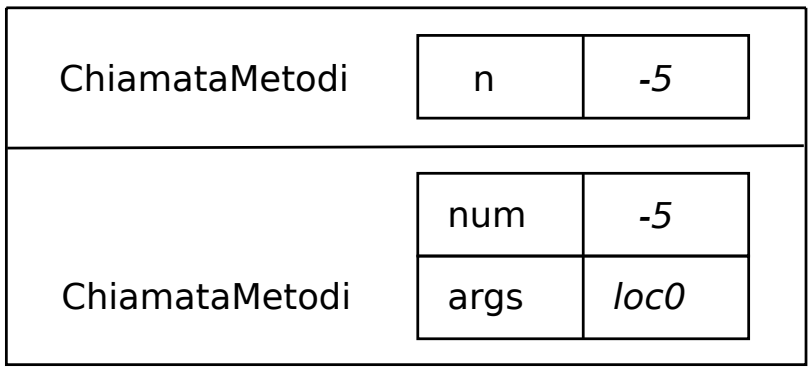


Evoluzione dello stato: lo stack (2)

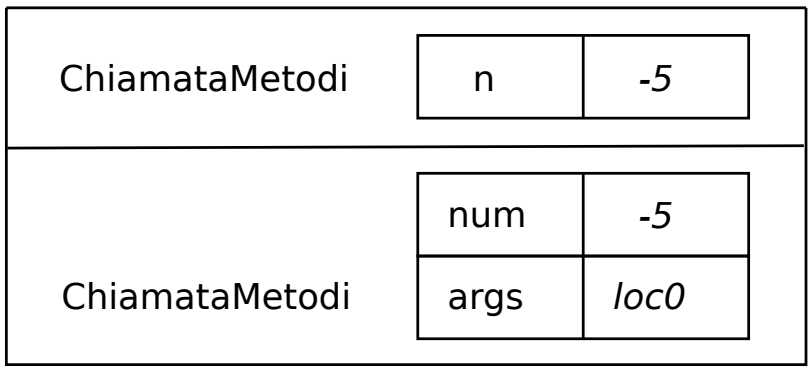


- Chiamata `doubleAbs(num)`

Evoluzione dello stato: lo stack (3)

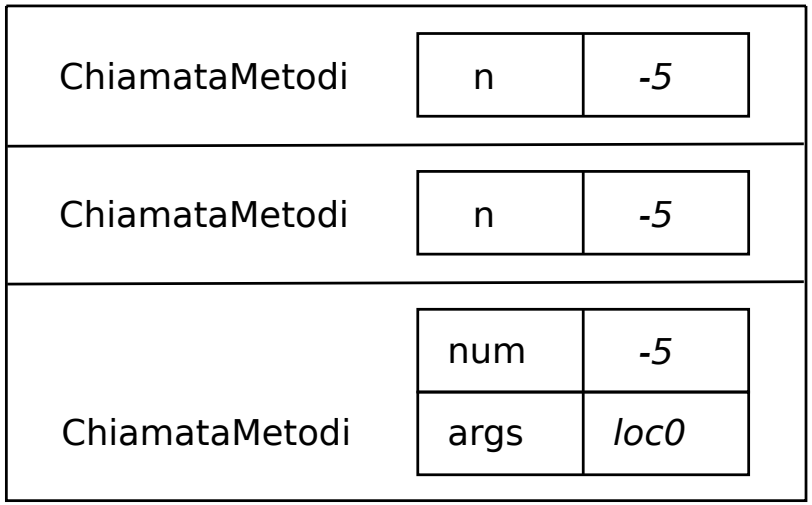


Evoluzione dello stato: lo stack (3)

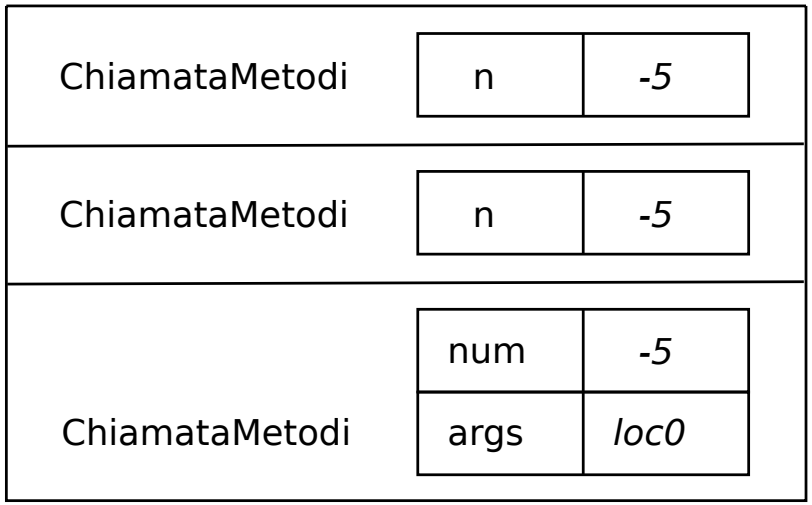


- Chiamata `abs(n)`

Evoluzione dello stato: lo stack (4)

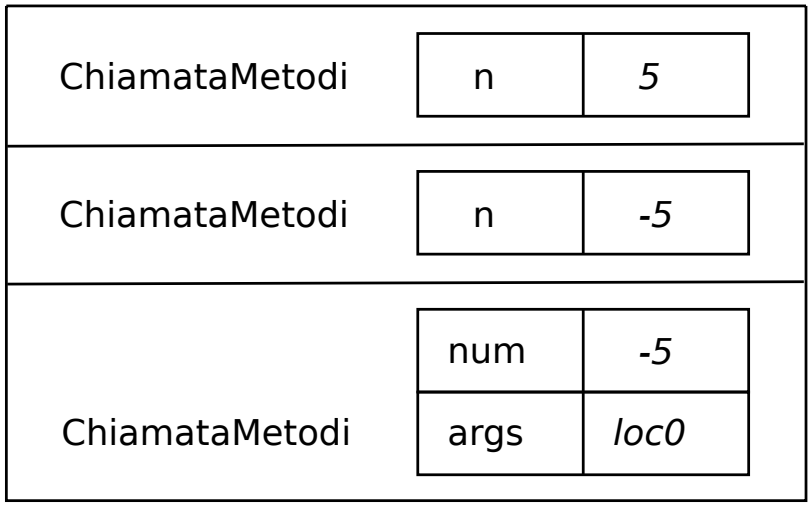


Evoluzione dello stato: lo stack (4)

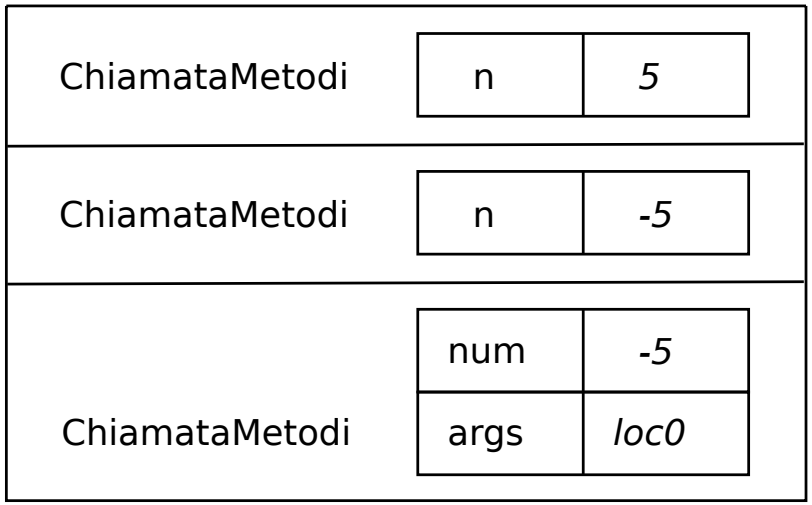


- Esecuzione di `if (n < 0) n = -n`

Evoluzione dello stato: lo stack (5)

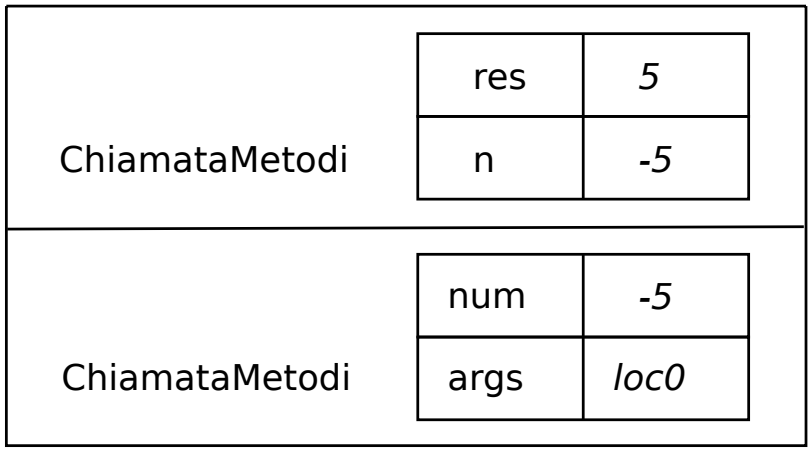


Evoluzione dello stato: lo stack (5)

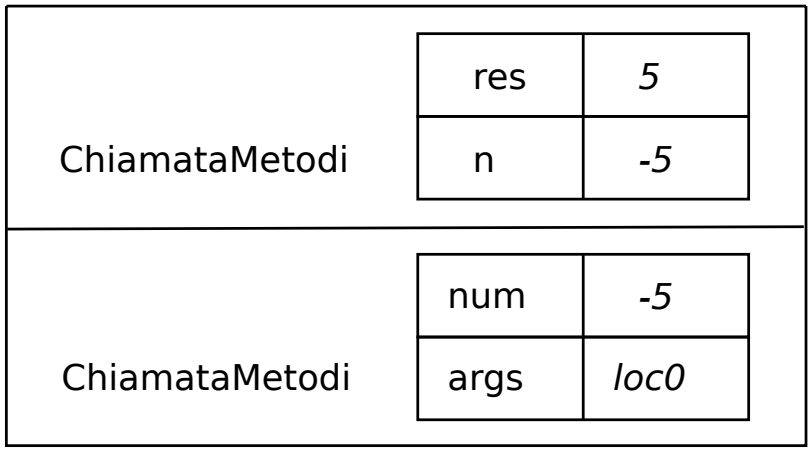


- Uscita da `abs()`
- Esecuzione di `int res = abs(n)`

Evoluzione dello stato: lo stack (6)

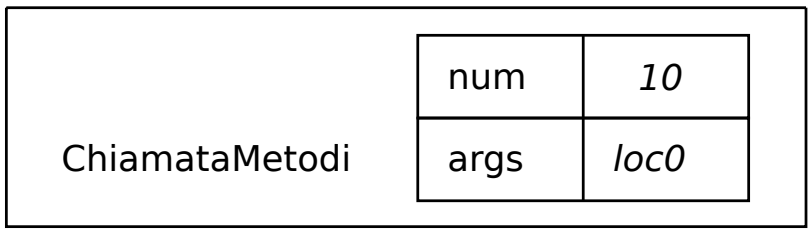


Evoluzione dello stato: lo stack (6)



- Uscita da `doubleAbs()`
- Esecuzione di `num = doubleAbs(num)`

Evoluzione dello stato: lo stack (7)



Fine

Compilazione: comandi

Per compilare una classe si usa il seguente comando:

```
javac <nomefile>.java
```

- se non ci sono errori, viene prodotto il file `<nomefile>.class`, contenente il bytecode della classe compilata
- eventuali altre classi riferite vengono compilate automaticamente (con limiti)

- È possibile indicare sulla linea di comando più file da compilare

```
javac <nomefile 1>.java ... <nomefile n>.java
```

Esecuzione: comandi

Per lanciare l'esecuzione di una classe (cioè il suo metodo statico `main`), si usa il seguente comando:

```
java <NomeClasse>
```

- **Attenzione**: è necessario specificare il nome delle classe, non il nome del file contenente la classe compilata!
  - l'interprete cerca il file compilato della classe nella directory corrente