Java: Basilari di Programmazione in Piccolo

Sommario: 20 Aprile, 2015

- Java Language Specification: Java SE 8 Edition
- Variabili, Fields e Scope degli identificatori di variabile
- Tipi Primitivi
- Espressioni, Statements, Blocchi
- Importanti default di Java: Object, 0-arity constructor, this, super.
- Rappresentazione Interna di Programma: Tabella delle Classi, Classi e AR dei metodi
- Invocazione di Metodo

Java Language Specification

- **Definizione Ufficiale**
  - Java Language Specification: Java SE 8 Edition
  - 15 febbraio 2015, pagg. 768
  - acquistabile (Amazon, Prentice Hall)
  - scaricabile da:
    - https://docs.oracle.com/javase/specs/jls/se7/html/
    - pagine del corso, sezione materiale.
- **Contiene**:
  - La definizione di ogni costrutto: Sintassi, Comportamento, motivazione e uso
  - La sintassi è fornita in una grammatica completa, riportata nel cap. 19.
- **Consultare**: in caso di dubbi

Fields e Scope

- **Variabili di Istanza (Non-Static Fields)**:
  - Definiscono lo stato degli oggetti della classe
  - Scope: Intera definizione di classe
- **Variabili di Classe (Static Fields)**:
  - Definiscono variabili statiche (degli oggetti) della classe
  - Scope: Intera definizione di classe

```
import java.io.*;
import java.util.*;

public class Ex {
    static int A;
/*Classe counter definisce un semplice contatore*/
    int A; //-- errore: A è gia definito
    int B;
 //metodi
public void met1 (int x) {
    int x; //-- errore: x è gia definito
    int y;
    y=A+x;
    {
        int y; //-- errore x è gia definito;
        int z = y+B;
    }
}
public int met2 (int A, int B) {
    met1(A+B+this.B);
    Ex.A++;
    return A+B;
}
}
```

Variabili, Parametri e Scope

- **Variabili locali**:
  - Definiscono variabili locali di un metodo o di blocco in-line
  - Scope: Intera definizione di metodo o blocco, rispettivamente
- **Parametri**:
  - Definiscono parametri di un metodo
  - Scope: Intera definizione di metodo

```
import java.io.*;
import java.util.*;

public class Ex {
    static int A;
/*Classe counter definisce un semplice contatore*/
    int A; //-- errore: A è gia definito
    int B;
 //metodi
public void met1 (int x) {
    int x; //-- errore: x è gia definito
    int y;
    y=A+x;
    {
        int y; //-- errore x è gia definito;
        int z = y+B;
    }
}
public int met2 (int A, int B) {
    met1(A+B+this.B);
    Ex.A++;
    return A+B;
}
}
```

Scope degli identificatori e Ricerca del binding

- In java tutti gli identificatori hanno scope statico
- Sono impedite collisioni di nomi
- Non abbiamo bisogno di meccanismo di Scoping Statico (per risolvere conflitti di scope)
- Gli Activation Record non hanno Catena Statica.

- Come troviamo il binding di un identificatore `A`?
  Sia AR il corrente record della valutazione del codice in cui occorre A. Cerco nell'ordine, nei seguenti Frame:
  - Frame di AR
  - Frame dell'oggetto, se AR è metodo di istanza,
  - Frame della classe.

Tipi Primitivi

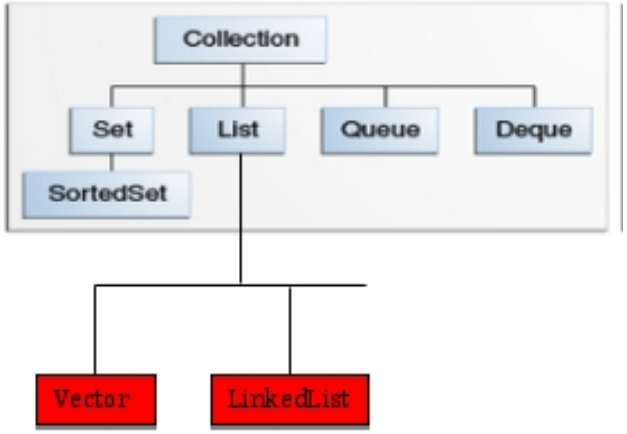
- Scalari: Sono implementati nella macchina ospite

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

- Usabili con gli operatori e nelle forme usuali
- Sono visti come (e convertiti in) oggetti (se necessario)

Tipi Strutturati

- Statici: Array del C/C^{++}
- Dinamici: Di uso comune le sottoclassi dell'interfaccia Collection<T>



- Nei nostri esercizi useremo:
  - Vector<T>: con accesso diretto
  - LinkedList<T>: con accesso sequenziale
- Facili definirne di nuovi
  (in specifiche classi, librerie .jar, API delle proprie applicazioni)

Espressioni, Statements, Blocchi

- Espressioni: Quelle di C/C^{++} sui tipi primitivi e Array Statici
  + Quelle sui nuovi Tipi (classi) + **new** + **invocazione**

- Statements e Blocchi: include la struttura C/C^{++} a parte:
  - identificatori di variabili
  - alcuni costrutti di controllo (iteratori di collezione, gestione eccezioni, synchronized block e concorrenza)
  - sistema dei tipi (sottotipi, e polimorfismo)

Example

Un programma C su valori primitivi e array (e senza statement goto), può essere incapsulato in una classe dove le dichiarazione delle variabili si mantengono (o sono rinominate in caso di collisione), quelle di procedura diventano metodi statici, le invocazioni di procedura diventano invocazioni di metodi, le espressioni si mantengono.

Example

Un programma C su valori primitivi e array (e senza statement goto), può essere incapsulato in una classe dove le dichiarazione delle variabili si mantengono (o sono rinominate in caso di collisione), quelle di procedura diventano metodi statici, le invocazioni di procedura diventano invocazioni di metodi, le espressioni si mantengono.

```
import java.io.*;
import java.util.*;

public class stuck {
    static int taxCalculation(int x){
        return x;}
    public static void main(String [] argv){
        char a = 'e';
        System.out.println("Totale da pagare in euro: e="+a+'='+taxCalculation(10));
    }
}

/* --- Banale rifrasamento in Java del programma C
 #include <stdio.h>
 #include <stdlib.h>

 int taxCalculation(int x){
    return x;}
 int taxCalculation(int x){
    return x;}
 int main (int argc, char *argv[]){
    char a = 'e';
    printf("Totale da pagare in euro: e=%2d\n", a+'='+taxCalculation(10));
    return 0;
 } // stampa: Totale da pagare in euro: e=172

 --- Ma ora ha un comportamento corretto
 Marco-Bellias-MacBook-Pro:stuck marcob$ javac stuck.java
 Marco-Bellias-MacBook-Pro:stuck marcob$ java stuck
 Totale da pagare in euro: e=e=10
 Marco-Bellias-MacBook-Pro:stuck marcob$
 }
 */
```

Rappresentazione Interna del Programma
Tabella delle Classi, Classi, Metodi

Una volta analizzato, un programma perde la sua struttura esterna per mostrare la struttura delle classi definite. Ciò è ottenuto dalla seguente struttura che è residente in Memoria Statica per l'intera durata dell'esecuzione del programma sulla JVM

- **Tabella delle Classi**. Relazione (Nome della classe,Accesso al Descrittore di Classe)
- **Descrittore di Classe**. Contiene: Accesso alla Superclasse e Frame degli identificatori di Field e Metodi
- **AR di Metodo**. Template per costruzione di AR. Contiene: struttura Frame per identificatori di Field e Metodi, in aggiunta a CD, PC, RA, Intermedi [1]

[1] CatenaDinamica, ProgramCounter al codice, ReturnAddress del valore, valori intermedi.

Classe di un Programma A

- **AR di Metodo**. Contiene: struttura Frame per identificatori di Field e Metodi, in aggiunta a CD, PC, RA, Intermedi
  - Quando invocato, il template del metodo è copiato in un AR, completato opportunamente e posto sullo Stack.
  - Metodi Statici: Frame contiene identificatore della classe
  - Metodi Istanza: Frame contiene una costante "this" con accesso all'oggetto su cui è invocato il metodo.
  - Gli altri campi sono gli analoghi di quelli delle usuali procedure dei Linguaggi di Programmazione.

```
public class CounterM {
static int currentCounter = 0;
static final int max = 100;
int cValue;
int cMax;
static int alive () {
    return currentCounter;}
CounterM(int init){
    cValue = init;
    cMax = init+max;}
int get(){return cValue;}
void reset(int init){
    cValue = init;
    cMax = init+max;}
void inc (int resetValue){
    if (cMax>cValue) cValue++;
    else reset(resetValue);}
}
```

Importanti default di Java

- **Object**. Tutte le classi (eccetto la classe Object) hanno una superclasse.

  `... class CounterM {...`

  sta per:

  `... class CounterM extends Object {...`

- **constructors**. Ogni costruttore ha come prima operazione la costruzione dell'oggetto della superclasse.

  Sia `A` il nome di una classe, e `A(p1,...,pk){s1;...;sn;}` la dichiarazione di un k-arity constructor.

  Allora se s1 è diverso da `super()`, il costruttore sta per:

  `A(p1,...,pk){super();s1;...;sn;}`

- **0-arity constructor**. Tutte le classi con metodi di istanza per oggetti hanno un costruttore di arità 0.

  Sia `A` il nome di una classe con metodi di istanza ma priva di un tale costruttore.

  Allora tale classe è considerata estesa con il costruttore 0-arity:

  `A(){super()};`

- **this**. Nelle dichiarazioni di un oggetto l'uso dell'identificatore di self reference this può essere omesso.

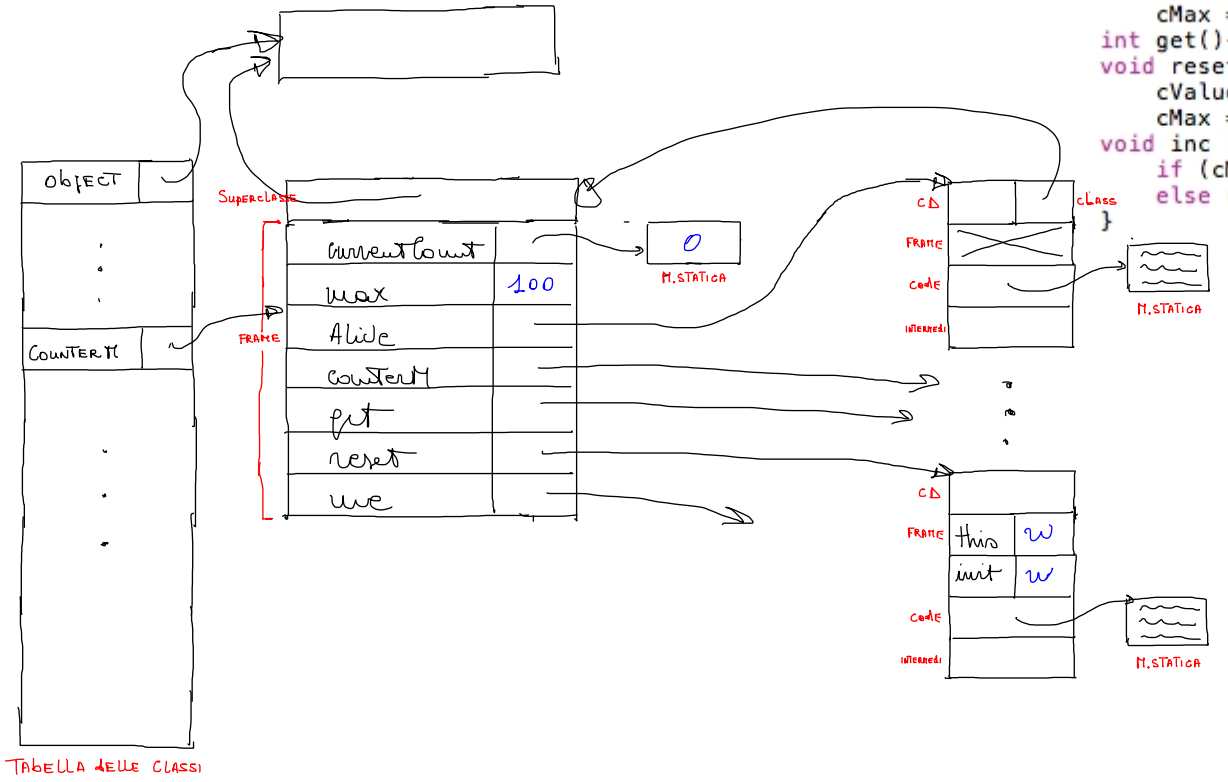
  `cValue, cMax, reset(resetValue)`

  usati nella classe CounterM, della slide precedente, stanno per:

  `this.cValue, this.cMax, this.reset(resetValue)`

Rappresentazione Interna di A e della sua classe

```
public class CounterM {
static int currentCounter = 0;
static final int max = 100;
int cValue;
int cMax;
static int alive () {
    return currentCounter;}
CounterM(int init){
    cValue = init;
    cMax = init+max;}
int get(){return cValue;}
void reset(int init){
    cValue = init;
    cMax = init+max;}
void inc (int resetValue){
    if (cMax>cValue) cValue++;
    else reset(resetValue);}
}
```



Invocazione di Metodo (di istanza)

Richiede un AR analogamente all'invocazione di procedura ricorsiva. Ma ha struttura: $\mathbf{E_0}.\mathbf{Name}(\mathbf{E_1}, ..., \mathbf{En})$
dove:

- $\mathbf{E_0}$ è un'espressione che calcola l'oggetto su cui invocare il metodo;
- **Name** deve essere il nome di un metodo dell'oggetto di $\mathbf{E_0}$
- $\mathbf{E_i}$ sono le espressioni che calcolano gli `n` argomenti dell'invocazione;
- La trasmissione degli argomenti è sempre e solo per valore; [2]

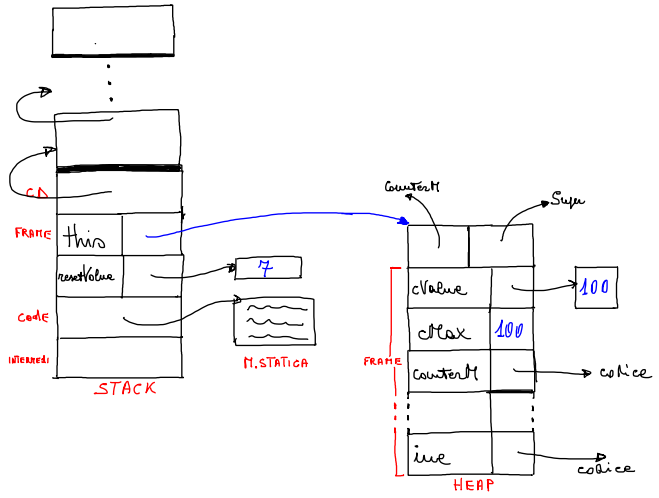
Consideriamo l'espressione `reset(resetValue)` che ...

[2] Ricordare che gli oggetti hanno valori con modello per reference

Invocazione di Metodo (di istanza)/1

- Consideriamo l'espressione `reset(resetValue)` che occorre nel corpo del metodo `inc`.
- A destra un possibile stato dello Stack di controllo e della Memoria prima dell'invocazione del metodo.
- Notare l'oggetto di tipo counterM, nello heap.

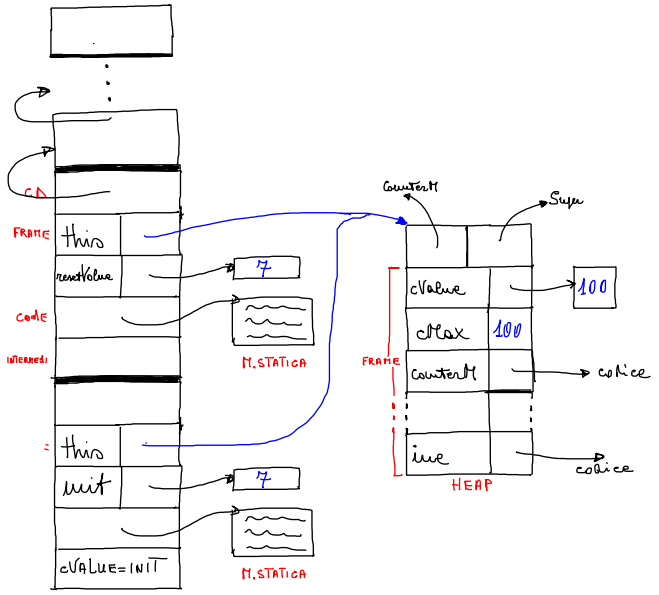
```
public class CounterM {
static int currentCounter = 0;
static final int max = 100;
int cValue;
int cMax;
static int alive () {
    return currentCounter;}
CounterM(int init){
    cValue = init;
    cMax = init+max;}
int get(){return cValue;}
void reset(int init){
    cValue = init;
    cMax = init+max;}
void inc (int resetValue){
    if (cMax>cValue) cValue++;
    else reset(resetValue);}
}
```



Invocazione di Metodo (di istanza)/2

- A destra lo stato dello Stack di controllo e della Memoria durante l'invocazione del metodo.

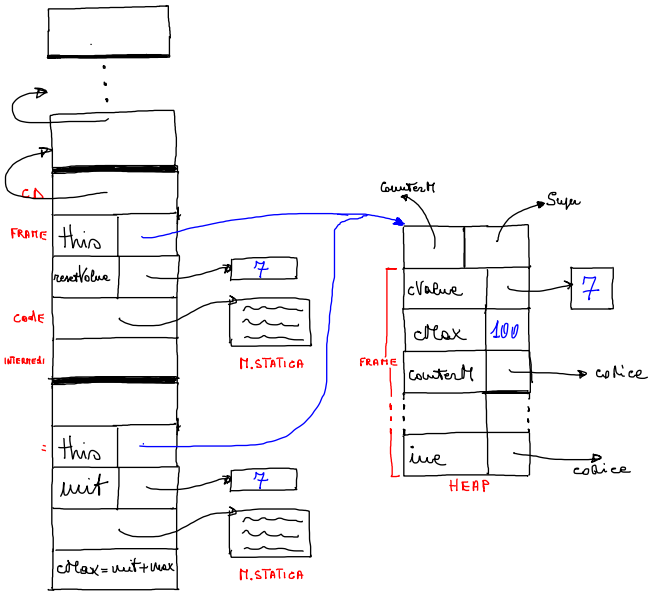
```
public class CounterM {
static int currentCounter = 0;
static final int max = 100;
int cValue;
int cMax;
static int alive () {
    return currentCounter;}
CounterM(int init){
    cValue = init;
    cMax = init+max;}
int get(){return cValue;}
void reset(int init){
    cValue = init;
    cMax = init+max;}
void inc (int resetValue){
    if (cMax>cValue) cValue++;
    else reset(resetValue);}
}
```



Invocazione di Metodo (di istanza)/3

- A destra lo stato dello Stack di controllo e della Memoria durante la valutazione del corpo del metodo invocato.

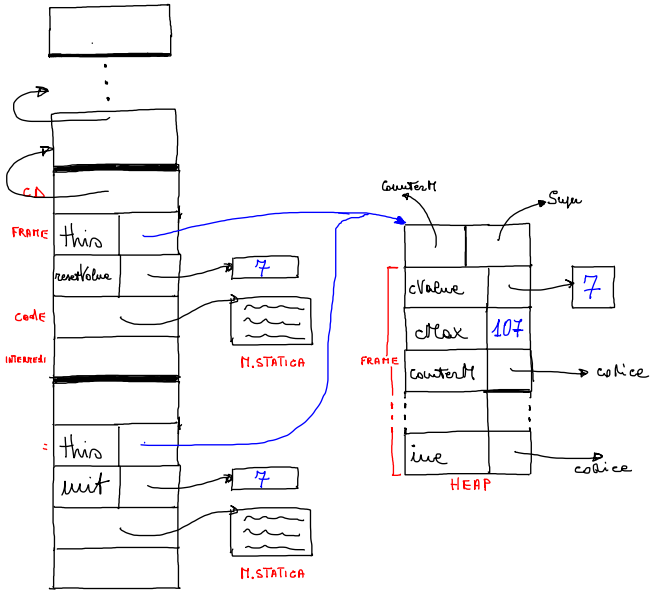
```
public class CounterM {
static int currentCounter = 0;
static final int max = 100;
int cValue;
int cMax;
static int alive () {
    return currentCounter;}
CounterM(int init){
    cValue = init;
    cMax = init+max;}
int get(){return cValue;}
void reset(int init){
    cValue = init;
    cMax = init+max;}
void inc (int resetValue){
    if (cMax>cValue) cValue++;
    else reset(resetValue);}
}
```



Invocazione di Metodo (di istanza)/4

- A destra lo stato dello Stack di controllo e della Memoria ancora valutazione del corpo del metodo invocato.

```
public class CounterM {
static int currentCounter = 0;
static final int max = 100;
int cValue;
int cMax;
static int alive () {
    return currentCounter;}
CounterM(int init){
    cValue = init;
    cMax = init+max;}
int get(){return cValue;}
void reset(int init){
    cValue = init;
    cMax = init+max;}
void inc (int resetValue){
    if (cMax>cValue) cValue++;
    else reset(resetValue);}
}
```



Invocazione di Metodo (di istanza)/5

- A destra lo stato dello Stack di controllo e della Memoria al termine dell'invocazione del metodo.

```
public class CounterM {
static int currentCounter = 0;
static final int max = 100;
int cValue;
int cMax;
static int alive () {
    return currentCounter;}
CounterM(int init){
    cValue = init;
    cMax = init+max;}
int get(){return cValue;}
void reset(int init){
    cValue = init;
    cMax = init+max;}
void inc (int resetValue){
    if (cMax>cValue) cValue++;
    else reset(resetValue);}
}
```