Java: Basilari di Programmazione in Piccolo

Sommario: 19 Aprile, 2018

- Java Language Specification: Java SE 8 Edition
- Variabili, Fields e Scope degli Identificatori
- Tipi Primitivi
- Espressioni, Statements, Blocchi

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;
    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 y 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;
    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 y 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:
  - Ma non è richiesto un Meccanismo per la gestione
- Sono impedite collisioni variabili locali
  - No variable shadowing nei metodi.
- Conflitti tra field e variabili locali o parametri risolti
  - Uso di this: `A+B+this.B`
- Gli Activation Record non richiedono Catena Statica.

- Come troviamo il binding di un identificatore `A`?
  Sia AR il corrente activation 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

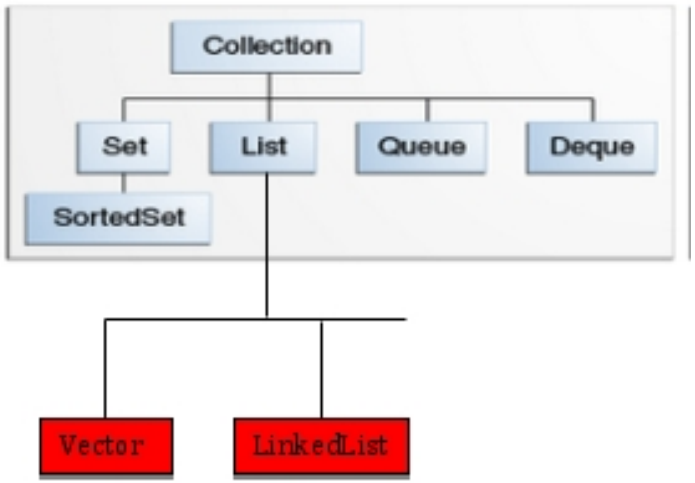
- Scalari: Sono implementati nella macchina ospite

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 racchiuso 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 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, Oggetti, 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**. Coppie (Nome, Descrittore);
- **Descrittore di Classe**. Contiene: Accesso a Superclasse, ...i;
- **Oggetto**. Contiene: Accesso a Super Oggetto, a Classe, ...;
- **Descrittore di Metodo**. Template per AR. Contiene: ...;
- **AR**. Controllo Esecuzione. Contiene: Istanza Template ...