Linguaggi Object Oriented: Principi e Introduzione a Java

Sommario: 24,26 Aprile, 2017

- Linguaggi OO: Rilevanza e Note Storiche
- Principi dei Linguaggi OO.
  - Struttura dei programmi: Classi e Oggetti in Java
  - Struttura dei programmi: Relazioni tra Classi in Java
  - Unità di Programmazione: Funzionalità Data-Centriche in Java
  - Effetti: Riuso di Codice in Java
  - Effetti: Evoluzione di ADT e Moduli
  - Effetti: Applicazioni Multi-Media, Multi-Contesto
- Java: Guida all'istallazione e Uso.

Linguaggi OO: Rilevanza e Note Storiche

- **1967. SIMULA**: Kristen Nygaard estende ALGOL60 con Classi, Oggetti, Ereditarità, Metodi, Coroutines.
  - Complicato per le applicazioni dell'epoca.
    Usato in Software prototipale e Ambito accademico.
- **1976. SmallTalk**: Alan Kay definisce il primo linguaggio incentrato sull'Oggetto come unica forma di valore.
  - Estremamente semplice.
    Limitato Manipolazione simbolica, AI, Ambito accademico.
- **1979. C++**: Brian Stroustrup estende C con Classi, e Sistema di tipi con polimorfismo.
  - Ripete l'esperienza di Nygaard.
    Oggi impiegato spesso al posto di C.

Linguaggi OO: Java

- Si considera che sia il 1° più popolare [1] Linguaggio di Programmazione
- **1991-1995. James Gosling, SUN**. Prima Definizione e Sviluppo: Java 1.0
  - Keypoints.
    - Sviluppo Software per Televisione via cavo, Interattiva
    - Multimedia e Concorrenza
    - Costruito intorno a C: C ridotto, esteso con tipi, per programmazione in piccolo.
    - Portabilità. Macchina Virtuale: Compila 1-Volta - Esegui Ovunque
- **1998. Java Community Process** (JCP). Comitato internazionale per la definizione e sviluppo di Java
- **2014. Ultima Release**: Java 8 (Oracle)
- Altre notizie: http://en.wikipedia.org/wiki/Java_%28programming_language%29

[1] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html: 20.846%, 2°C 13.905%

Principi: Struttura dei Programmi

- I programmi sono una collezione strutturata di funzionalità
  - Un programma in Java é una collezione di **Classi**, raccolte in:
    - **Package** a livello logico.

```
package FactPackage2;

import java.lang.*;
import java.util.*;

class IllegalArgument extends Exception{...}

class Factorial{...}

public class Main {...}
```

    - *directory* [2] *di sistema* a livello fisico.

```
Marco-Bellias-MacBook-Pro:FactPackage1 marcob$ cd .../FactPackage1
Marco-Bellias-MacBook-Pro:FactPackage1 marcob$ ls
Factorial.java      IllegalArgument.java    Main.java
Marco-Bellias-MacBook-Pro:FactPackage1 marcob$ javac Main.java
Marco-Bellias-MacBook-Pro:FactPackage1 marcob$ ls
Factorial.class     Factorial.java      IllegalArgument.class
    IllegalArgument.java    Main.class      Main.java
Marco-Bellias-MacBook-Pro:FactPackage1 marcob$
```

[2] Nel gergo Apple, la directory è chiamata *folder*

Struttura dei Programmi: Classi di Java

- Classi (e oggetti) = Meccanismi per **localizzare-Incapsulare** definizioni relative ad una funzionalità
- Una classe per una definizione della funzione *fattoriale* in Java

```
import java.lang.*;
import java.util.*;

class Factorial{
    public static int fact(int n) throws IllegalArgument{
        if (n<0) throw new IllegalArgument("Fact:"+n);
        if (n==0) return 1;
        else return n*fact(n-1);
    }
}
```

- Una classe per **oggetti** di tipo *contatore* in Java

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

public class AbstractCounter {
/* Classe counter definisce una classe per oggetti
   che si comportano come un contatore astratto
*/
    private int c;
//metodi
    public AbstractCounter(){c=0;}
    public void reset () {c=0;}
    public int get(){return c;}
    public void inc (){c++;}
}
```

Classi di Java introducono *entità* per il programma

- Un numero finito di *entità statiche*.
  - Una classe che introduce una solo entità utilizzabile per il calcolo del fattoriale
  - L'entità è introdotta attraverso un **metodo**
  - Metodi = astrazioni dei controllo[3] ma in Java, non sono considerate Unità di Programmazione

```
import java.lang.*;
import java.util.*;

class Factorial{
    public static int fact(int n) throws IllegalArgument{
        if (n<0) throw new IllegalArgument("Fact:"+n);
        if (n==0) return 1;
        else return n*fact(n-1);
    }
}
```

  - Queste entità possono essere usate come sotto per modellare la programmazione di linguaggi non OO.

[3] come le procedure e le funzioni dei linguaggi imperativi

Classi di Java introducono *entità* statiche per il programma

- Un numero finito di *entità statiche*.

```
import java.lang.*;
import java.util.*;

class Factorial{
    public static int fact(int n) throws IllegalArgument{
        if (n<0) throw new IllegalArgument("Fact:"+n);
        if (n==0) return 1;
        else return n*fact(n-1);
    }
}
```

  - Queste entità possono essere usate come sotto per modellare la programmazione di linguaggi non OO.

```
import java.lang.*;
import java.util.*;

public class Main {
//OVERVIEW: testing
public static void main(String[] args)throws IllegalArgument{
    int fact5 = Factorial.fact(5);
    System.out.println("fact of 5 is " + fact5);
}
}

/*
 Marco-Bellias-MacBook-Pro:FactPackage1 marcob$ ls
 Factorial.class         IllegalArgument.class   Main.class
 Factorial.java      IllegalArgument.java    Main.java
 Marco-Bellias-MacBook-Pro:FactPackage1 marcob$ java Main
 fact of 5 is 120
 Marco-Bellias-MacBook-Pro:FactPackage1 marcob$
*/
```

  - Non rappresentano il modo standard di uso delle classi di Java

Classi di Java introducono *entità* per il programma

- Un numero finito di *entità statiche*.
- Un numero infinito di *entità dinamiche*: **Oggetti**.
  - **Oggetti** = istanze di una classe dotate di una "copia" di tutti i **Campi e Metodi** non statici della classe
  - **Campi** = Componenti identificati da una dichiarazione che specifica **Nome** e **Tipo**[4] del campo.

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

public class AbstractCounter {
/* Classe counter definisce una classe per oggetti
   che si comportano come un contatore astratto
*/
    private int c;
//metodi
    public AbstractCounter(){c=0;}
    public void reset () {c=0;}
    public int get(){return c;}
    public void inc (){c++;}
}
```

  - La classe AbstractCounter definisce:

[4] Ogni Classe è (definisce) un tipo che si aggiunge ai tipi predefiniti

Classi di Java introducono Oggetti e Tipi per il programma

- Un numero finito di *entità statiche*.
- Un numero infinito di *entità dinamiche*: **Oggetti**.

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

public class AbstractCounter {
/* Classe counter definisce una classe per oggetti
   che si comportano come un contatore astratto
*/
    private int c;
//metodi
    public AbstractCounter(){c=0;}
    public void reset () {c=0;}
    public int get(){return c;}
    public void inc (){c++;}
}
```

  - La classe AbstractCounter definisce:
    - 1 campo di nome c di tipo "int" (e, privato = accessibile solo dai metodi della classe)
    - 1 costruttore per oggetti di tipo AbstractCounter
    - 3 metodi per operare sugli oggetti di tipo AbstractCounter

Classi di Java introducono Oggetti e Tipi per il programma

- Un numero finito di *entità statiche*.
- Un numero infinito di *entità dinamiche*: **Oggetti**.

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

public class AbstractCounter {
/* Classe counter definisce una classe per oggetti
   che si comportano come un contatore astratto
*/
    private int c;
//metodi
    public AbstractCounter(){c=0;}
    public void reset () {c=0;}
    public int get(){return c;}
    public void inc (){c++;}
}
```

TIPATI

- Gli oggetti sono valori e sono usati come sotto

```
public class ACUse {
/* Classe ACUse mostra un uso degli oggetti di tipi AbstractCounter */
//metodi
    public static void main (String [] args) {
        AbstractCounter myCounterA, myCounterB; //due variabili di tipo...
        myCounterB = new AbstractCounter();  //crea un oggetto di tipo ... e lo assegna a...
        myCounterB.inc();
        myCounterA = myCounterB;
        myCounterA.inc();
        System.out.println("myCounterA segna "+myCounterA.get());
        System.out.println("myCounterB segna "+myCounterB.get());
    }
}
```

tipo

CREAZIONE di OGGETTO

- che rappresenta il modo standard di uso delle classi di Java

Gli Oggetti sono valori con modello "value reference"

- Cosa viene stampato?.

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

public class AbstractCounter {
/* Classe counter definisce una classe per oggetti
   che si comportano come un contatore astratto
*/
    private int c;
//metodi
    public AbstractCounter(){c=0;}
    public void reset () {c=0;}
    public int get(){return c;}
    public void inc (){c++;}
}
```

```
public class ACUse {
/* Classe ACUse mostra un uso degli oggetti di tipi AbstractCounter */
//metodi
    public static void main (String [] args) {
        AbstractCounter myCounterA, myCounterB; //due variabili di tipo...
        myCounterB = new AbstractCounter();  //crea un oggetto di tipo ... e lo assegna a...
        myCounterB.inc();
        myCounterA = myCounterB;
        myCounterA.inc();
        System.out.println("myCounterA segna "+myCounterA.get());
        System.out.println("myCounterB segna "+myCounterB.get());
    }
}
```

- Gli Oggetti in Java, sono:

  valori reference = il valore coincide con un reference alla struttura di memoria occupata.

Struttura dei Programmi: Relazioni tra Classi

I programmi sono una collezione strutturata di Classi (con propria funzionalità)

- La struttura della collezione è definita dalla **Relazione tra Classi**
- In Java la relazione è esplicita e ha due forme.
  - **extends**. Classe-Sottoclasse. Ereditarità singola
  - **implements**. Interfaccia-Sottoclasse. Ereditarità multipla
- La struttura è fondamentale per realizzare funzionalità con comportamento via, via, più complicato
- Parte centrale della programmazione OO in Java.

Principi: Funzionalità Data-Centric

I programmi sono una collezione strutturata di Classi (con propria funzionalità)

- La funzionalità è espressa da 1 o più Classi (in Java) correlate.
- Le Classi usualmente introducono Oggetti (con un Tipo).
- La funzionalità allora è espressa in forme simili a quella sotto

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

public class AbstractCounter {
/* Classe counter definisce una classe per oggetti
   che si comportano come un contatore astratto
*/
    private int c;
//metodi
    public AbstractCounter(){c=0;}
    public void reset () {c=0;}
    public int get(){return c;}
    public void inc (){c++;}
}
```

- **Data-Centric** dove vediamo una classe di valori importati per il problema da risolvere e le operazioni che possiamo usare su essi per risolvere il problema o una sua parte (metodologie di programmazione Data-Centric)

Principi: Riuso di Codice

Quando la realizzazione di un sistema A è stata completata e la sua esecuzione è a regime, la scrittura del codice di A entra in una nuova fase: La manutenzione.

- È una fase di programmazione molto impegnativa e costosa.
- **Riuso** significa contenere il codice che deve essere prodotto e gli errori conseguenti
- Nell'esempio sotto, A è esteso con una nuova classe di contatori che riusano il codice dei contatori già presenti, ma

```
public class NewCounter extends AbstractCounter{
/* Classe NewCounter per contatori AbstractCounter con operazione
   alive che indica quanti NewCounter sono stati prodotti.
*/
    private static int counterCount = 0;
    public static int alive(){return counterCount;}
//metodi
    public NewCounter(){counterCount++;}
}
```

- sono in grado di dire quanti contatori sono in uso.

Principi: Riuso di Codice/2

- A è esteso con una nuova classe di contatori che riusano il codice dei contatori già presenti, ma hanno delle proprietà aggiuntive

```
public class NewCounter extends AbstractCounter{
/* Classe NewCounter per contatori AbstractCounter con operazione
   alive che indica quanti NewCounter sono stati prodotti.
*/
    private static int counterCount = 0;
    public static int alive(){return counterCount;}
//metodi
    public NewCounter(){counterCount++;}
}
```

- vecchie e nuove parti di A convivono perfettamente.

```
public class ACUse {
/* Classe ACUse mostra un uso degli oggetti di tipi AbstractCounter e NewCounter */
//metodi
    public static void main (String [] args) {
        AbstractCounter myCounterA, myCounterB; //due variabili di tipo...
        myCounterB = new AbstractCounter();  //crea un oggetto di tipo ... e lo assegna a...
        myCounterB.inc();
        myCounterA = myCounterB;
        myCounterA.inc();
        System.out.println("myCounterA segna "+myCounterA.get());
        System.out.println("myCounterB segna "+myCounterB.get());
        {//uso di NewCounter
            NewCounter x;
            for (int i=0; i+myCounterA.get()<10; i++) {//i due tipi di contatore usati insiemee New
                NewCounter y = new NewCounter();
                x=y;
                x.inc();
            }
        }
        System.out.println("sono stati usati "+NewCounter.alive()+" NewCounter");
    }
}
```

Principi: ADT, Moduli, Oggetti, Packages, API

- ADT. Non presenti in Java come meccanismi specifici, ma
  - Java ha meccanismi **Modificatori** che:
  - regolano la visibilità di Classi, Oggetti, campi e metodi e
  - forniscono Classi in grado di comportarsi come ADT

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

public class AbstractCounter {
/* Classe counter definisce una classe per oggetti
   che si comportano come un contatore astratto
*/
    private int c;
//metodi
    public AbstractCounter(){c=0;}
    public void reset () {c=0;}
    public int get(){return c;}
    public void inc (){c++;}
}
```

- Moduli. Classi e Oggetti in Java sono Unità di Programmazione[5], ma per la programmazione in grande
  - Java combina **Modificatori** con **Packages** e **Applicative Programming Interface**:

[5] queste unità, in Java, possono essere viste come moduli per programmazione in piccolo

Principi: Applicazioni Multi-Media, Multi-Contesto

- **Multi-Media**. La modularità e l'astrazione racchiuse negli Oggetti permette di **integrare strutture** di natura molto diversa tra loro quali suoni, immagini, video.
- **Multi-Contesto**. Il modello Data-Centric si presta a programmare applicazioni in contesti molto diversi quali commerciali, amministrativi, gestionali e scientifici.
- **Verbosità e Efficienza**. Sono i veri limiti dell'approccio OO.

Java: Guida all'istallazione e Uso

La nostra piattaforma (Linux, Mac, Windows, ...) deve avere istallato il Java Development Kit: jdk1.6.XX o versioni superiori

- **Download e Istallazione.**
  (a) Connettersi al sito http://www.oracle.com/technetwork/java/javaee/downloads/index.html
  (b) selezionare la versione più avanzata per la propria macchina
  (c) istallare il jdk scaricato, seguendo le indicazioni
- **Selezione Directories.**
  (a) Directory jdkXXX/bin dei comandi javac (compilatore) e java (JVM).
    - Individuare il cammino PTool alla directory "jdkXXX/bin"

  (b) Directory YYYY da usare per i propri programmi Java.
    - Creare una directory YYYY e individuare il cammino PProgram ad essa
- **Setting delle variabili di ambiente.**
  Variabili coinvolte:
  path = PTool
  classpath = PProgram
  Per le diverse piattaforme vedere[6]
  https://docs.oracle.com/javase/tutorial/essential/io/pathClass.html
- **Uso.**
  (a) Editing del programma:
    - Classi del programma hanno nome con suffisso ".java", sono editate con un text editor,
    - sono poste in una directory XX appositamente creata e posta nella directory PProgram.

  (b) Compilazione del programma: javac XX/c1.java ... XX/ck.java
    - eseguire in command prompt (Windows), Terminal (Mac)
    - produce k file ci.class contenenti il bytecode della corrispondente classe

  (c) Esecuzione: java XX/cm
    - esegue il codice del metodo main presente nella classe cm.class.

[6] per Mac, operare sempre da Terminal seguendo le indicazioni per Linux.