

Linguaggi Object Oriented: Principi e Introduzione a Java

Sommario: 13 Aprile, 2015

- 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: Riutilizzo di Codice in Java
 - Effetti: Evoluzione di ADT e Moduli
 - Effetti: Applicazioni Multi-Media, Multi-Contesto
- Java: Guida all'installazione e Uso.

Linguaggi OO: Rilevanza e Note Storiche

- **1967. SIMULA:** Kristen Nygaard estende ALGOL60 con Classi, Oggetti, Ereditarietà, 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 2° 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>

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$
```

²Nel gergo Apple, la directory è chiamata *folder*  4/18

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 IllegalArgumentException{
        if (n<0) throw new IllegalArgumentException("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 sola entità utilizzabile per il calcolo del fattoriale
 - L'entità è introdotta attraverso un **metodo**
 - Metodi = astrazioni dei controllo³ ma in Java, non sono considerate Unità di Programmazione

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

class Factorial{
    public static int fact(int n) throws IllegalArgumentException{
        if (n<0) throw new IllegalArgumentException("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.

³come 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 IllegalArgumentException{
        if (n<0) throw new IllegalArgumentException("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 IllegalArgumentException{
        int fact5 = Factorial.fact(5);
        System.out.println("fact of 5 is " + fact5);
    }
}

/*
Marco-Bellias-MacBook-Pro:FactPackage1 marcob$ ls
Factorial.class      IllegalArgumentException.class  Main.class
Factorial.java      IllegalArgumentException.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:

⁴Ogni Classe è (definisce) un tipo che si aggiunge ai tipi predefiniti  8/18

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++;}
}
```

- 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());
    }
}
```

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

Java: Guida all'installazione e Uso

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

- **Download e Installazione.**

- (a) Connettersi al sito <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- (b) selezionare la versione più avanzata per la propria macchina
- (c) installare 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 vedere⁶

<https://docs.oracle.com/javase/tutorial/essential/io/pathClass.html>

- **Uso.**

(a) Eding del programma:

classi del programma hanno nome con suffisso ".java", sono editate con un text editor, poste in una directory XX appositamente creata e posta nella directory PT.

(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.   18/18