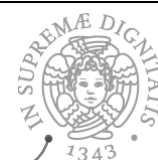




AA 2014-2015

PROGRAMMAZIONE 2

1



PRESENTAZIONI

👤 Gianluigi Ferrari

- Email giangi@di.unipi.it
- Web: www.di.unipi.it/~giangi

👤 Di cosa mi occupo (ricerca)

- Formal methods in Software Engineering,
 - ✓ Verification, Model Checking, Static Analysis of Programs
- Programming Language and Models for Concurrent and Distributed Systems
 - ✓ Service Oriented Computing
 - ✓ Cloud Computing
- Security
 - ✓ Language Based Security

2




UNIVERSITÀ DI PISA

PROGRAMMAZIONE 2

Cosa studiamo?
Due tematiche principali



Metodologie di Programmazione Object-Oriented



- ✎ tecniche per la programmazione orientata ad oggetti (in piccolo)
 - specifica, implementazione, dimostrazioni
 - Programmazione concorrente
- ✎ esemplificate utilizzando il linguaggio Java

4



- ✦ specifiche, implementazioni, dimostrazioni di “correttezza”
- ✦ le dimostrazioni sono tanto importanti quanto le implementazioni
- ✦ ogni meccanismo di astrazione ha associata una particolare sequenza di operazioni di specifica, implementazione e dimostrazione
 - che ci porterà ad utilizzare sottoinsiemi di costrutti Java “coerenti”
- ✦ non è compito di questo corso introdurre il linguaggio nella sua interezza
 - nè tanto meno le sue librerie (che vi imparate da soli, quando vi servono)

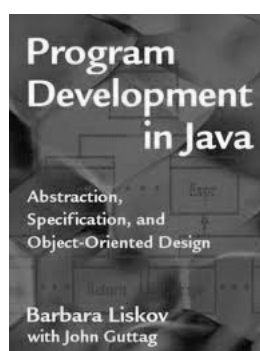
5

Materiale Didattico



Liskov, Guttag

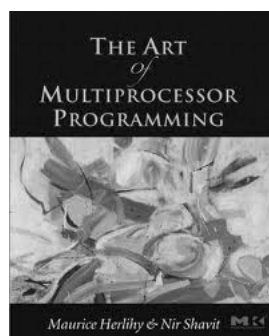
Program Development in
Java, Abstraction,
Specification, and
Object-Oriented Design



Materiale Didattico



M. Herlihy, N. Shavit
The Art of Multiprocessor
Programming



👉 Bruni, Corradini, Gervasi,
👉 Programmazione in Java,
👉 Seconda Edizione, Apogeo
2011



LINGUAGGI DI PROGRAMMAZIONE



- ✎ Studiare i principi che stanno alla base dei linguaggi di programmazione
- ✎ Essenziale per comprendere il progetto, la realizzazione e l'applicazione pratica dei linguaggi
- ✎ Non ci interessa rispondere alla domanda "Java e' meglio di C#"?

9

Linguaggi di Programmazione



- ✎ Paradigmi linguistici, costrutti
- ✎ Semantica operativa
- ✎ Implementazione, strutture a tempo di esecuzione
- ✎ Il nostro approccio: la descrizione dell'implementazione del linguaggio e' guidata dalla semantica formale:
 - Stretta relazione tra la semantica e la struttura del run time del linguaggio
 - Struttura del run-time simulata in Ocaml (a volte ritornano)
- ✎ Numerosi libri sull'argomento che sono utili da studiare per il nostro corso, metteremo a disposizione delle note.

10

FONDAMENTI UN VALORE



- ✎ Evitare discussioni da osteria
- ✎ Evitare malfuzionamenti
- ✎ Numerosi esempi: Post sul blog ufficiale di Microsoft Azure:
 - *Alle 17:45 ora del Pacifico del 28 febbraio 2012 Microsoft ha rilevato un problema che affliggeva i servizi Windows Azure in diverse regioni. Il problema è stato analizzato rapidamente ed è stato attribuito a un bug software. Sebbene le origini effettive siano oggetto di indagine, il problema sembra fosse causato da un calcolo del tempo errato nell'anno bisestile".*
- ✎ Teoria aiuta il progetto e la realizzazione dei linguaggi
- ✎ Esempio: implementazioni efficienti possono ottenere se la generazione del codice eseguibile e' ritardata fino a che non sono disponibili dati del run-time

11

Materiale Didattico



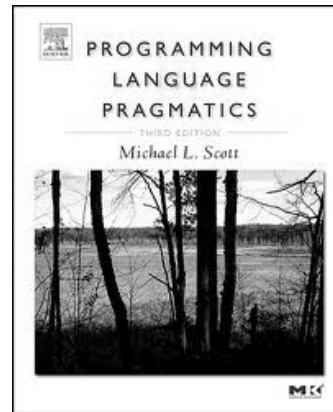
- ✎ M. Gabbrielli & S. Martini, Linguaggi di programmazione – Principi e paradigmi, McGraw-Hill 2006.



Materiale Didattico



Michael Scott
Programming Language
Pragmatics, Third
Edition



Materiale Didattico



Peter Sestoft
Programming Language
Concepts, Springer
2012

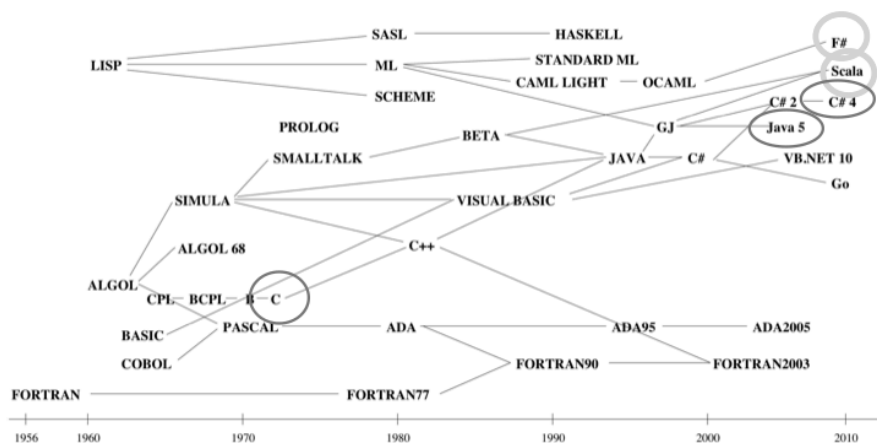


Linguaggi e astrazione



- i linguaggi di programmazione ad alto livello moderni sono il più potente strumento di astrazione messo a disposizione dei programmatori
- i linguaggi si sono evoluti trasformando in costrutti linguistici (e realizzando una volta per tutte nell'implementazione del linguaggio)
 - tecniche e metodologie sviluppate nell'ambito della programmazione, degli algoritmi, dell'ingegneria del software e dei sistemi operativi
 - in certi casi perfino in settori di applicazioni (basi di dati, intelligenza artificiale, simulazione, etc.)
- di fondamentale importanza è stata l'introduzione nei linguaggi di vari *meccanismi di astrazione*, che permettono di estendere il linguaggio (con nuove operazioni, nuovi tipi di dato, etc.) semplicemente scrivendo dei programmi nel linguaggio stesso

15



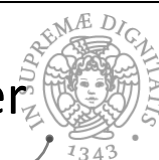
16

Tanti Linguaggi. Perché?



- ✎ Prendiamo il migliore e basta!!!
 - Come vedrete a **Calcolabilità e Complessità**, i linguaggi di programmazione sono tutti Turing Equivalenti: stessa potenza espressiva
- ✎ I migliori sono tanti
 - Visione Oracle-Sun: Java
 - Visione Microsoft: C#
 - Visione dello sviluppatore Web: JavaScript
- ✎ Tante motivazioni: alcuni linguaggi meglio si adattano a un particolare contesto
 - PROLOG: AI

A day life of a web programmer



- ✎ Develop a web site
 - Separare presentazione, stile e funzionalità
- ✎ Client side programming
 - Javascript (funzionalità), HTML (presentazione), CSS (stile)
- ✎ Server side programming
 - CGI scripts
 - Scripting (PHP, Pearl, Ruby ..)
 - Java
 - Database access (SQL)
 - XML per web services

Una nota personale



- ✎ Ho iniziato a programmare con ALgolW e LISP
- ✎ Ho utilizzato PROLOG per progettare la base di conoscenza di un sistema esperto
- ✎ Ho utilizzato C, C++, Ocaml, Java in diversi progetti di ricerca
- ✎ Ora utilizzo Java e Ocaml per l'insegnamento e Java nei progetti di ricerca

Fibonacci



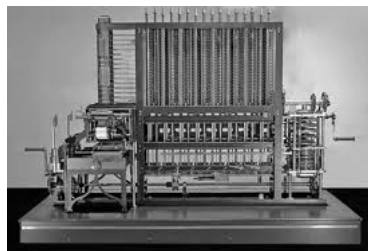
- ✎ Curiosita'. Navigate sul web
 - <http://www.scriptol.com/programming/fibonacci.php>
- ✎ Come si esprime il programma che calcola i numeri di fibonacci nei principali linguaggi di programmazione

Il primo programmatore



Babbage Analytical
Engine (1830, 1840)

Programmare: dati e
operazioni



E la prima hacker



Ada Lovelace (figlia di
Byron)



Un po' di storia dei linguaggi

i linguaggi di programmazione nascono con la macchina di Von Neumann (macchina a programma memorizzato)

- i programmi sono un particolare tipo di dato rappresentato nella memoria della macchina
- la macchina possiede un interprete capace di eseguire il programma memorizzato, e quindi di implementare un qualunque algoritmo descrivibile nel "linguaggio macchina"
- un qualunque linguaggio macchina dotato di semplici operazioni primitive per effettuare la scelta e per iterare (o simili) è Turing-equivalente, cioè può descrivere tutti gli algoritmi

i linguaggi hanno tutti lo stesso potere espressivo, ma la caratteristica distintiva importante è il "quanto costa esprimere"

- direttamente legato al "livello di astrazione" fornito dal linguaggio

23

➤ dai linguaggi macchina ai linguaggi Assembler

- nomi simbolici per operazioni e dati

➤ (anni 50) FORTRAN e COBOL (sempreverdi)

- notazioni ad alto livello orientate rispettivamente al calcolo scientifico (numerico) ed alla gestione dati (anche su memoria secondaria)
- astrazione procedurale (sottoprogrammi, ma con caratteristiche molto simili ai costrutti forniti dai linguaggi macchina)
- nuove operazioni e strutture dati (per esempio, gli arrays in FORTRAN, e i records in COBOL)
- nulla di significativamente diverso dai linguaggi macchina

24

I favolosi anni '60: LISP e ALGOL'60



- ✓ risultati teorici a monte
 - ✓ formalizzazione degli aspetti sintattici
 - ✓ primi risultati semantici basati sul lambda-calcolo
- ✓ caratteristiche comuni
 - ✓ introduzione dell'ambiente
 - ✓ vera astrazione procedurale con ricorsione
- ✓ ALGOL'60
 - ✓ primo linguaggio imperativo veramente ad alto livello
 - ✓ scoping statico e gestione dinamica della memoria a stack
- ✓ LISP (sempreverde)
 - ✓ primo linguaggio funzionale, direttamente ispirato al lambda-calcolo
 - ✓ scoping dinamico, strutture dati dinamiche, gestione dinamica della memoria a heap con garbage collector

25

- ALGOL'60, prototipo dei linguaggi imperativi
- LISP, prototipo dei linguaggi logici e funzionali
- Analizzando i due linguaggi ci accordiamo che originano concetti simili non a caso basati sulla teoria
 - La gestione dell'ambiente tramite lo stack
- gli approcci restano diversi e danno origine a due filoni
 - il filone imperativo (esempio C)
 - il filone funzionale (esempio Ocaml)



26

La fine degli anni '60



- PL/I: il primo tentativo di linguaggio “totalitario” (targato IBM)
 - tentativo di sintesi fra LISP, ALGOL'60 e COBOL
 - fallito per mancanza di una visione semantica unitaria
- SIMULA'67: nasce la classe
 - estensione di ALGOL'60 orientato alla simulazione discreta
 - quasi sconosciuto, riscoperto 15 anni dopo

27

Evoluzione del filone imperativo



- risultati anni '70
 - metodologie di programmazione, tipi di dati astratti, modularità, classi e oggetti
 - programmazione di sistema in linguaggi ad alto livello: eccezioni e concorrenza
- PASCAL
 - estensione di ALGOL'60 con la definizione di tipi (non astratti), l'uso esplicito di puntatori e la gestione dinamica della memoria a heap (senza garbage collector)
 - semplice implementazione mista (vedi dopo) facilmente portabile

28

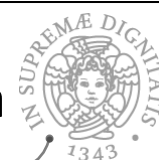
Il dopo PASCAL



- C= PASCAL + moduli + tipi astratti + eccezioni + semplice interfaccia per interagire con il sistema operativo
- ADA: il secondo tentativo di linguaggio “totalitario” (targato Dipartimento della Difesa U.S.A.)
 - come sopra + concorrenza + costrutti per la programmazione in tempo reale
 - progetto ambizioso, anche dal punto di vista semantico, con una grande enfasi sulla semantica statica (proprietà verificabili dal compilatore)
- C++ = C + classi e oggetti (allocati sulla heap, ancora senza garbage collector)

29

La programmazione logica



- PROLOG
 - implementazione di un frammento del calcolo dei predicati del primo ordine
 - strutture dati molto flessibili (termini) con calcolo effettuato dall’algoritmo di unificazione
 - computazioni non-deterministiche
 - gestione della memoria a heap con garbage collector
- CLP (Constraint Logic Programming)
 - PROLOG + calcolo su domini diversi (anche numerici) con opportuni algoritmi di soluzione di vincoli

30

La programmazione funzionale



- ML: implementazione del lambda-calcolo tipato
- definizione di nuovi tipi ricorsivi, i valori dei nuovi tipi sono termini, che possono essere visitati con un meccanismo di pattern matching (versione semplificata dell'unificazione)
 - scoping statico (a differenza di LISP)
 - semantica statica molto potente (inferenza e controllo dei tipi)
 - un programma "corretto" per la semantica statica quasi sempre va bene
 - gestione della memoria a heap con garbage collector
 - HASKELL= ML con regola di valutazione "lazy"

31



JAVA



- molte caratteristiche dal filone imperativo
 - essenzialmente tutte quelle del C++
- alcune caratteristiche dei linguaggi del filone logico-funzionale
 - gestione della memoria con garbage collector
- utilizza il meccanismo delle classi e dell'ereditarietà per ridurre il numero di meccanismi primitivi
 - quasi tutto viene realizzato con classi predefinite nelle librerie
- ha una implementazione mista (anch'essa tipica del filone logico)
 - che ne facilita la portabilità e lo rende particolarmente adatto ad essere integrato nelle applicazioni di rete

32

SCALA



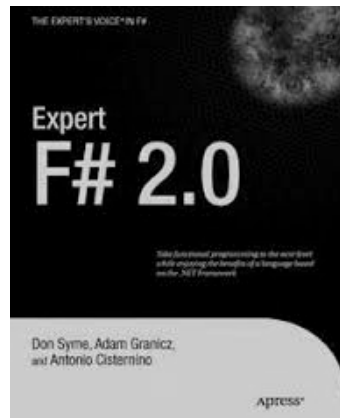
- Scala smoothly integrates features of object-oriented and functional languages



F#



- ML spiegato al popolo



Materiale didattico, esame, istruzioni per l'uso del corso



- il materiale didattico delle lezioni sarà disponibile sulla mia pagina web così come tutti i programmi Ocaml e Java che verranno discussi nelle esercitazioni
- esame = progetto + prova scritta + orale
 - ammissione all'orale con votazione $\geq 18/30$ nello scritto valutazione positiva del progetto.
 - 2 prove intermedie che possono rimpiazzare la prova scritta
- consigli
 - seguire il corso (e soprattutto le esercitazioni), mantenendosi al passo con lo studio
 - partecipare (attivamente) alle esercitazioni
 - sostenere le prove intermedie