AA 2014-2015

PROGRAMMAZIONE II – Corso B
Introduzione al corso

- Andrea Corradini
  - Email: andrea@di.unipi.it
  - Web: www.di.unipi.it/~andrea

INFORMAZIONI GENERALI

- Pagina web del corso:

  http://www.di.unipi.it/~andrea/Didattica/PR2-B-14/

- Orario di ricevimento
  - **Martedì ore 15-18**
- Modalità di esame
  - scritto + **progetto** + orale
  - 2 prove in itinere (sostituiscono scritto)
- Un corso in evoluzione...

CONTENUTI DEL CORSO

- Metodologie di Programmazione Object-Oriented
  - Da programmazione “value-oriented” (OCaml) a “object-oriented” (Java)
  - Meccanismi di astrazione
  - Programmazione “by contract”
- Paradigmi di programmazione: aspetti implementativi
  - Linguaggio didattico
  - Costrutti dei vari paradigmi
  - Strutture a tempo di esecuzione
  - Implementazione in OCaml basata su semantica operazionale
- Dappertutto: enfasi su **semantica**, non su **sintassi**

http://www.di.unipi.it/~andrea/Didattica/PR2-B-14/

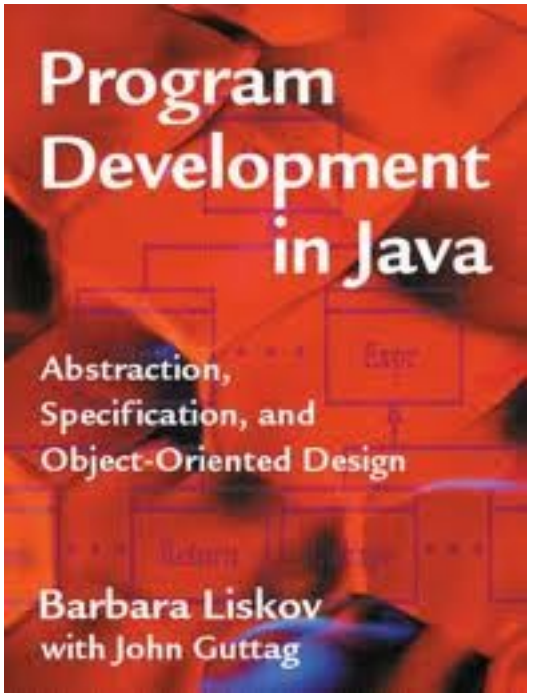
METODOLOGIE DI PROGRAMMAZIONE OBJECT-ORIENTED

- Useremo Java come esempio
- Non introdurremo il linguaggio nella sua interezza
  - né tanto meno le sue librerie
- Enfasi su specifiche, implementazioni, dimostrazioni di “correttezza”
- 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”
- Le dimostrazioni sono tanto importanti quanto le implementazioni
- Programmazione concorrente (verso fine corso)

Materiale Didattico: Tecniche di programmazione OO

Liskov, Guttag

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



http://www.di.unipi.it/~andrea/Didattica/PR2-B-14/

Materiale Didattico: OCaml e Java

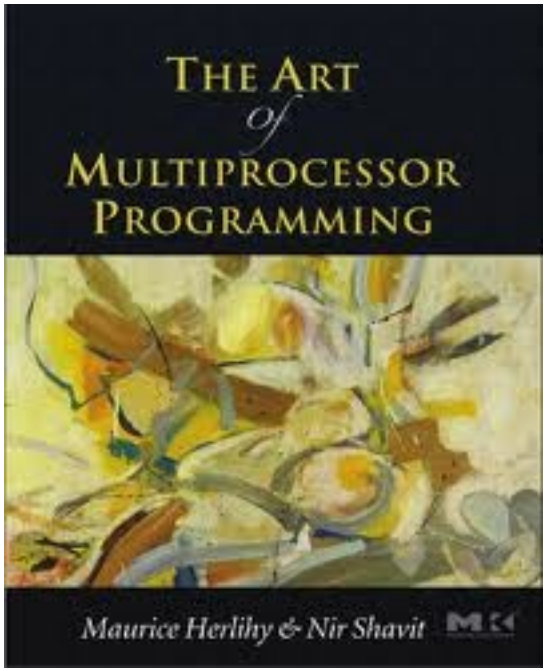
- OCaml:
  - Steve Zdancewiz, Benjamin C. Pierce, Stephanie Weirich
  - Programming Languages and Techniques
  - Lecture Notes for CIS 120
- Roberto Bruni, Andrea Corradini, Vincenzo Gervasi
  - Programmazione in Java,
  - Seconda Edizione, Apogeo 2011
- *Disclaimer*: solo se interessa approfondire il linguaggio. Molti testi equivalenti...

http://www.di.unipi.it/~andrea/Didattica/PR2-B-14/

Materiale Didattico: Programmazione Concorrente

M. Herlihy, N. Shavit

The Art of Multiprocessor Programming



http://www.di.unipi.it/~andrea/Didattica/PR2-B-14/

PARADIGMI 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 a domande come "Java è meglio di C#"?

COSA VEDREMO?

- Paradigmi linguistici, costrutti
- Semantica operazionale
- Implementazione, strutture a tempo di esecuzione (*runtime*)
- Il nostro approccio: la descrizione dell'implememementazione del linguaggio è guidata dalla semantica formale:
  - Stretta relazione tra la semantica e la struttura del runtime del linguaggio
- Numerosi libri sull'argomento che sono utili da studiare per il nostro corso
- Metteremo a disposizione delle note

FONDAMENTI: UN VALORE

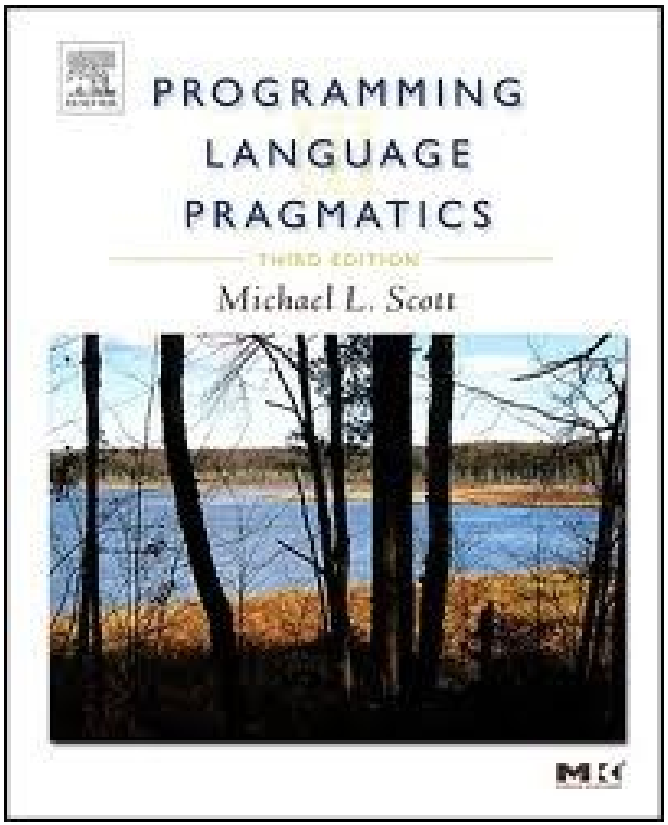
- Evitare ambiguità
- Evitare malfuzionamenti
- Numerosi esempi: Post sul blog ufficiale di Microsoft Azure:
  - *Alle 17:45 ora del Pacifico del 28 febbraio 2012 Microsoft ha rilevato un problema che affligeva i servizi Windows Azure in diverse regioni. Il problema è stato analizzato rapidamente ed 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"*.
- La teoria aiuta il progetto e la realizzazione dei linguaggi

Materiale Didattico

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

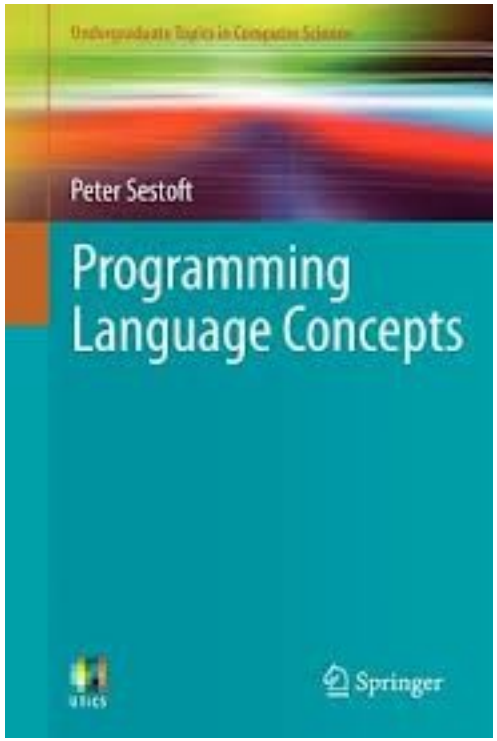
Materiale Didattico

- Michael Scott Programming Language Pragmatics, Third Edition

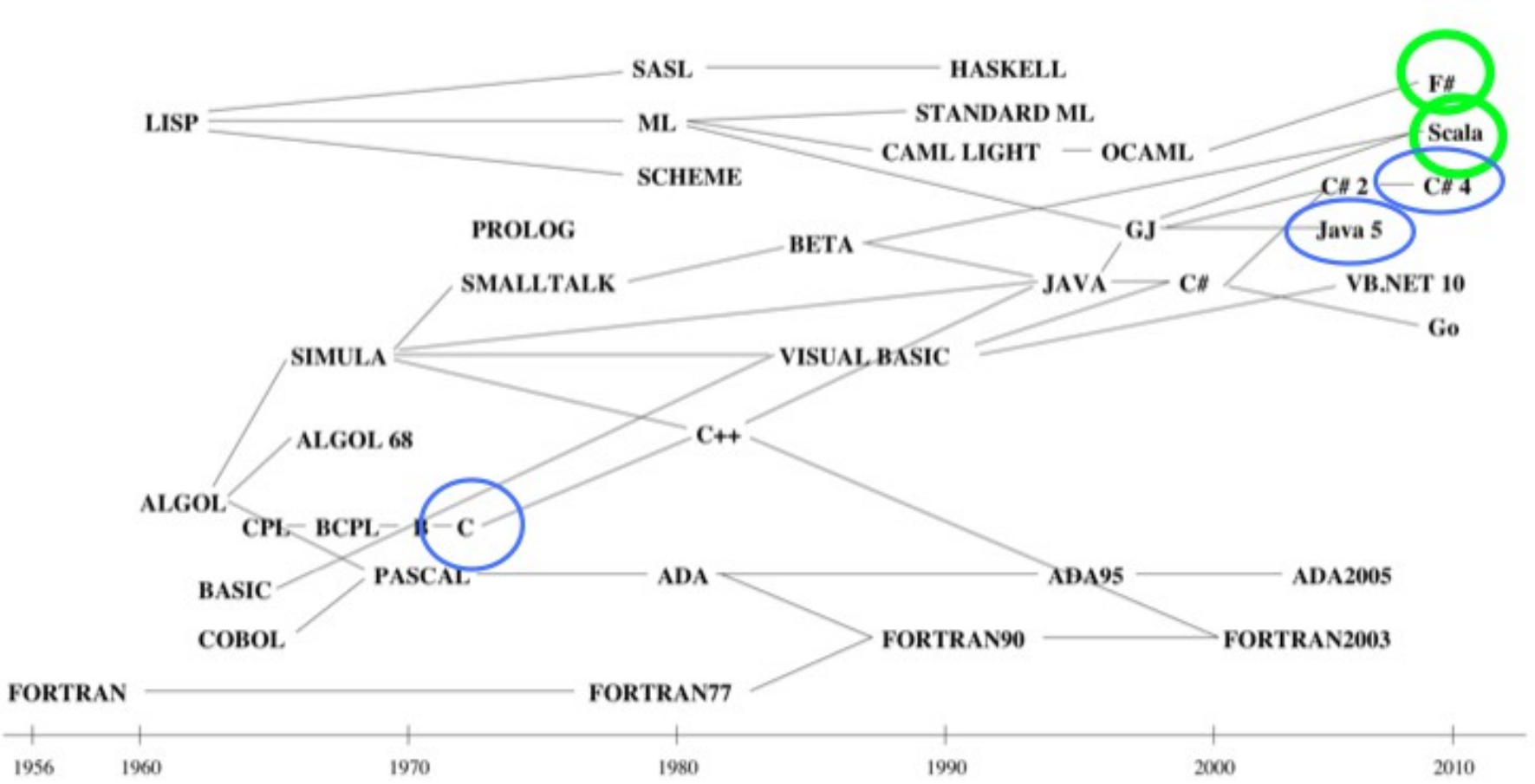


Materiale Didattico

Peter Sestoft
Programming
Language Concepts,
Springer 2012



LINGUAGGI DI PROGRAMMAZIONE: EVOLUZIONE



PERCHÉ TANTI LINGUAGGI?

- Sono tutti computazionalmente equivalenti:
  - Come vedrete a **Calcolabilità e Complessità**, i linguaggi di programmazione sono tutti “Turing equivalenti”
- Non ce n'è uno migliore: a ciascuno il suo...
  - Visione Oracle-Sun: Java
  - Visione Microsoft: C#
  - Visione dello sviluppatore Web: JavaScript
- Alcuni linguaggi sono preferibili in determinati contesti applicativi
  - Esempio, PROLOG per Artificial Intelligence
- Ma anche in un singolo contesto...

A day in the life of a Web Developer

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

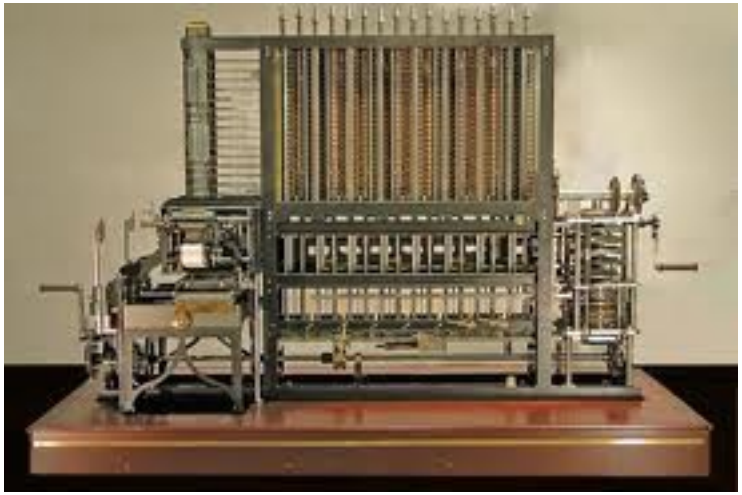
MA QUANTI SONO I LINGUAGGI DI PROGRAMMAZIONE?

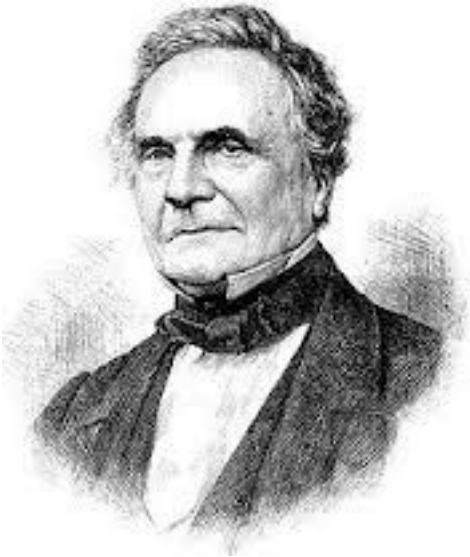
- n+1 !!!
- Curiosità: The “hello world!” Collection
  - http://www.roesler-ac.de/wolfram/hello.htm
- Meno linguaggi, ma più interessante:
  - http://www.scriptol.com/programming/fibonacci.php

Il primo progetto di computer

Babbage Analytical Engine (1830/40)

Macchina programmabile, con input, output e sistema di elaborazione dati





E la prima programmatrice

Ada Lovelace
(figlia di Lord 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

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 anche in particolari contesti applicativi (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

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

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 (gestito con stack)
  - 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

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

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

Il dopo PASCAL

- C = PASCAL + moduli + tipi astratti + eccezioni + semplice interfaccia per interagire con il sistema operativo
- ADA: 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)

La programmazione logica

- PROLOG
  - Implementazione di un frammento del calcolo dei predicati del primo ordine
    - Clausole Horn + risoluzione
  - 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

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"

JAVA

- Molte caratteristche dal filone imperativo
  - essenzialmente tutte quelle del C++
- Alcune caratteristiche dei linguaggi del filone logico-funzionale
  - Gestione della memoria con garbage collector
- Utiliza 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): compilazione in bytecode eseguibile sulla Java Virtual Machine (JVM)
  - Ne facilita la portabilità e lo rende particolarmente adatto ad essere integrato nelle applicazioni di rete

SCALA

- Integra caratteristiche object-oriented e funzionali
  - Compilazione produce codice intermedio Java, eseguibile sulla JVM

F#

- Linguaggio multi-paradigma (imperativo, funzionale, object oriented) basato su .NET
- Variante di ML largamente compatibile con OCaml
- ML spiegato al popolo