UNIVERSITA' DEGLI STUDI DI ROMA

Breve Excursus su Evoluzione della Programmazione

Corso di
Linguaggi e Metodologie di Programmazione

Introduzione

Le slide presenti in questo documento hanno il solo scopo di riassumere i momenti fondamentali della evoluzione delle metodologie e dei linguaggi di programmazione

Si rimanda lo studente ad un più ampio studio sui testi segnalati per il corso.

Programmazione anni 60

Linguaggi di Programmazione già possedevano:

- astrazione rispetto ai registri del processore
- uso avanzato della memoria

Ma erano fortemente legati alle stesse modalità di organizzazione del flusso di operazioni.

Il 'salto incondizionato' (GO TO, o, semplicemente, GOTO) rappresentava l'unico meccanismo per gestire il flusso decisionale di un algoritmo

Critiche al GOTO

UNIVERSITA' DEGLI STUDI DI ROMA TorVergata Università

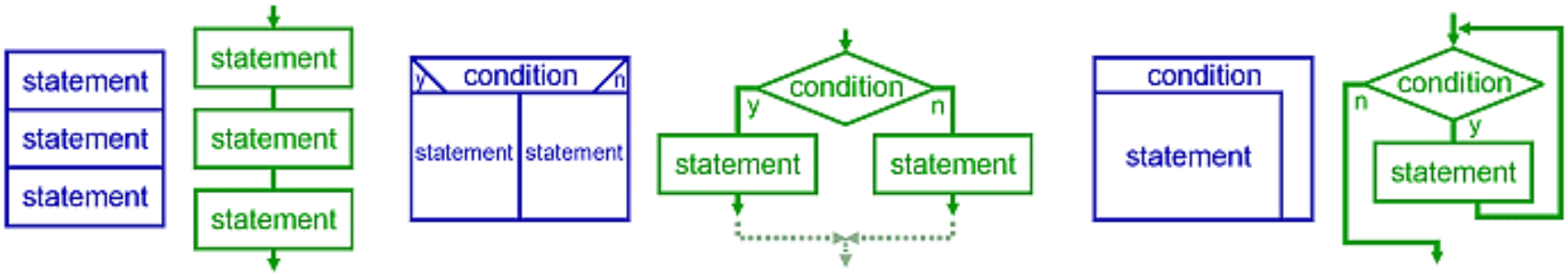
- Verso la fine degli anni 60, prime critiche alla programmazione tradizionale:
  - Edsger Dijkstra (March 1968). "Go To Statement Considered Harmful". Communications of the ACM 11 (3): 147–148. doi:10.1145/362929.362947.

    "unrestricted GOTO statements should be abolished from higher-level languages because they complicate the task of analyzing and verifying the correctness of programs"
  - Teorema di Böhm-Jacopini (1966): qualunque algoritmo può essere implementato utilizzando tre sole strutture, la *sequenza*, la *selezione* ed il *ciclo* (*iterazione*), da applicare ricorsivamente alla composizione di istruzioni elementari
  - Diffusione crescente di ALGOL (ALGOrithmic Language), in particolare nella sua versione ALGOL 60 (il linguaggio era cmq stato già inventato negli anni 50)

Programmazione Strutturata

Un programma strutturato è composto di tre combinazioni principali, applicate a blocchi primitivi di operazioni (statement)

- *Sequenza*: una esecuzione ordinata di statement
- *Selezione*: a seconda dello 'stato' del programma, uno statement viene 'selezionato' da un insieme di statement per essere eseguito
- *Ciclo, o iterazione*: uno statement è eseguito fino a che il programma non raggiunge un determinato stato

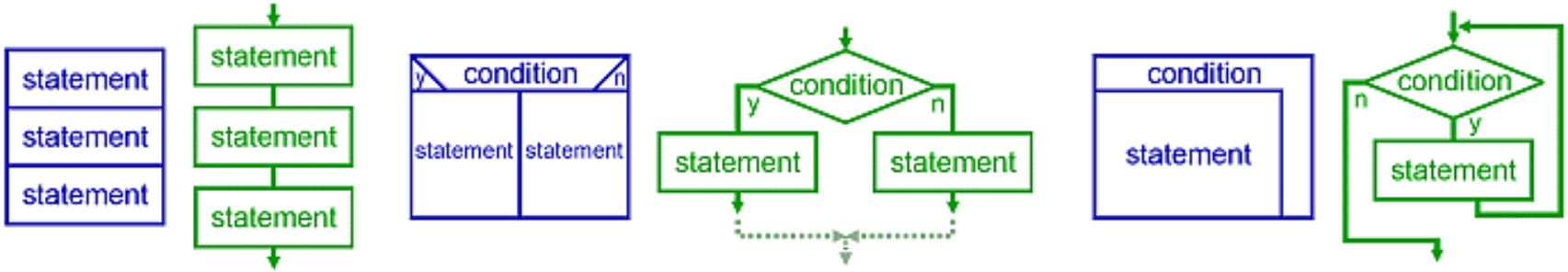


In blu, le rappresentazioni originariamente utilizzate per rappresentare i tre elementi fondamentali della programmazione strutturata: la sequenza, la selezione, la iterazione

Programmazione Strutturata

I seguenti requisiti devono essere rispettati da un linguaggio di programmazione strutturato

- *Completezza*: le tre strutture della programmazione strutturata devono avere almeno una rappresentanza sintattica nel linguaggio (e opzionalmente, delle variazioni)
- *Singolo punto di Ingresso e Uscita*: in ogni struttura di controllo si devono poter identificare un singolo punto di ingresso e un singolo punto di uscita. Ciò è necessario per la successiva:
- *Componibilità*: ogni struttura di controllo deve poter essere considerata come un macro-statement, di modo da poter essere usata, ricorsivamente, come istruzione in altre strutture di controllo



In blu, le rappresentazioni originariamente utilizzate per rappresentare i tre elementi fondamentali della programmazione strutturata: la sequenza, la selezione, la iterazione

Programmazione Procedurale

Consiste nella possibilità di agevolare la leggibilità di un programma (e la sua manutenibilità) tramite la definizione di blocchi di codice, racchiusi da delimitatori e usualmente identificati da un nome.

Tali blocchi assumo il nome di:

- *Subroutine* (in italiano sottoprogrammi, anche se nessuno usa tale nome!)
- *Procedure*
- *Funzioni*
- *Metodi*

tali nomi implicano differenze dipendenti dal contesto del modello di linguaggio usato

Principi Fondamentali dell'OOP (Object Oriented Programming)

- **Encapsulation**
  - Nascondere dettagli implementativi, permettendo di raggiungere solo alcune informazioni tramite *accessors* e *mutators*.
- **Abstraction**
  - Direttamente correlata alla precedente
  - La Abstraction ha a che fare con la definizione di modelli, viste etc.. per gli oggetti che si vogliono rappresentare
  - …mentre la encapsulation garantisce che le implementazioni di tali modelli rimangano opache ai loro utilizzatori
  - "*An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of object and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.*" — G. Booch, Object-Oriented Design With Applications, Benjamin/Cummings, MenloPark, California, 1991.

Principi Fondamentali dell'OOP (Object Oriented Programming)

- **Inheritance**
  - La *ereditarietà* permette di raccogliere a fattor comune caratteristiche condivise da oggetti simili, definendo dei modelli che rappresentino tali caratteristiche, e specializzando tali modelli per insiemi di oggetti via via più ristretti, ma accomunati da una similarità maggiore (ossia, più caratteristiche comuni).
  - Esempio: la *classe* delle autovetture accomuna tutti gli oggetti con 3 o 4 ruote dotate di un qualche sistema di propulsione e che permettano a delle persone di spostarsi. La classe delle citycar può essere definita come una specializzazione della precedente (quindi *eredita* dalla precedente) perché comprende tutte le sue caratteristiche, ed ha in più caratteristiche *aggiuntive*, o *restrizioni* su quelle già definite (e.g. avere dimensioni ridotte, o bassi consumi etc..)
- **Polymorphism** *(la capacità di assumere diverse forme)*
  - Attenzione, queste "forme" sono i **tipi**. Proprio grazie all'ereditarietà e in virtù della differenza tra tipo reale di un oggetto (i.e. la classe con la quale un oggetto è stato creato) e tipo dichiarato in un riferimento (il tipo di una variabile che punta ad un oggetto) gli oggetti possono presentarsi, attraverso le variabili che puntano ad essi, con tipizzazioni diverse
  - Esempio:
    Persona p = new Impiegato("mario", "rossi", "mat_12345678"). // p sarà visto dal compilatore come una Persona, sebbene l'oggetto sia un Impiegato
  - Polimorfismo dei metodi
    - *Overriding* (run-time polymorphism) // tipo reale scoperto a runtime tramite late binding!
    - *Overloading* (compile-time polymorphism) // scelta del metodo effettuata dal compilatore, sulla base dei parametri