

# Perche sviluppare?

Why coding?

Il software domina il mondo

# Coding is fun

La programmazione come forma d'arte

# Coding is fun (1)

I seguenti punti sintetizzano perchè l'ingegneria del software può essere assimilata alle forme d'arte tradizionali e moderne.

- Un algoritmo può essere oggettivamente bello o brutto, ma anche **interpretabile soggettivamente**.
- La realizzazione di un software deriva da un'idea. Una soluzione software è **frutto dell'inventiva** o addirittura del genio di uno o più uomini (con profili professionali differenziati ma che per comodità chiameremo "programmatori").
- Non esiste una rappresentazione unica di un programma, ciascun programmatore **interpreta** e **realizza** il codice a **modo suo**.
- Ogni programmatore ha un **proprio stile**, che può variare nel tempo.
- Il buon programmatore unisce **tecnica**, **talento** e **creatività**. Questi aspetti, in diversa misura e rapporto, sono riscontrabili in tutti gli artisti.
- Un algoritmo software, come la musica, è anche figlio di una **ispirazione momentanea**. Uno stesso programmatore tipicamente realizza una procedura in maniera diversa a seconda del momento.

# Coding is fun (2)

- Anche nel software, come nella poesia e nella pittura, esistono le **correnti**, le **scuole** di **pensiero** e le mode.
- Il codice di un programma, se ben scritto, **ordinato** ed **indentato**, può essere paragonato allo svolgimento di una **poesia**, di un **poema**.
- Un software può essere **fonte** di **ispirazione** per un altro software.
- Un codice software è sempre posizionabile in un **periodo temporale** ben definito, per **forma** (es. linguaggio di programmazione), **tecnica** (es. procedurale, strutturata od object oriented) e **contenuto** (tipologia di soluzione indirizzata).
- Non tutti sono in grado di capire un algoritmo software. Non tutti possono comprenderlo allo stesso modo. Come per le altre forme d'arte, per poterlo apprezzare appieno servono le giuste conoscenze ed esperienze.
- Il programmatore sente come una **propria creazione** il software scritto, spesso ne è **geloso**.
- Come per i quadri, il programmatore ci tiene a **firmare** il proprio codice.

# Contenuti unità formativa

- Concetti ed elementi di **programmazione** di base (C#, Java)
- Metodi di **analisi** e **risoluzione** di problemi a livello software
- Concetti base di **progettazione** software e **rappresentazione** degli algoritmi

# Concetti ed elementi di programmazione di base

- Concetti generali:

- Algoritmi
- Linguaggi di programmazione
- Paradigmi di programmazione
- Processi di sviluppo software (software engineering)

- Costrutti software:

- Il linguaggio C#-Java, caratteristiche;
- L'ambiente installazione e configurazione;
- Struttura di un programma;
- Compilazione ed esecuzione di un programma;
- Utilizzo di IDE (Integrated Development Environment);
- Sintassi base;
- Istruzioni, variabili e di tipi di dati;
- Tipi di dato primitivi;
- Operatori aritmetici, di confronto e logici;
- Espressioni con operatori, annidamento di istruzioni;
- Costrutti iterativi (while, for ecc.);
- Controllo del flusso;
- Concetti di oggetto e programmazione OOP;
- Tipi di classe, metodi e proprietà;
- Le Collections, tipi di strutture dati e metodi (ArrayList, ecc.);

# Metodi di analisi e risoluzione di problemi a livello software

## Analisi:

- Esplicitazione dei requisiti
- Analisi dei requisiti
- Analisi del dominio
- Analisi di fattibilità
- Analisi dei costi



# Concetti base di progettazione software e rappresentazione degli algoritmi

## Progettazione:

- Progetto architetturale
- Progetto di dettaglio

## Rappresentazione di algoritmi:

- Diagrammi di flusso
- Pseudo codice
- UML (Unified Modeling Language)

# Iterativi e incrementali



Ambito di indagine

Hardware

Software

# Software

## Software di base:

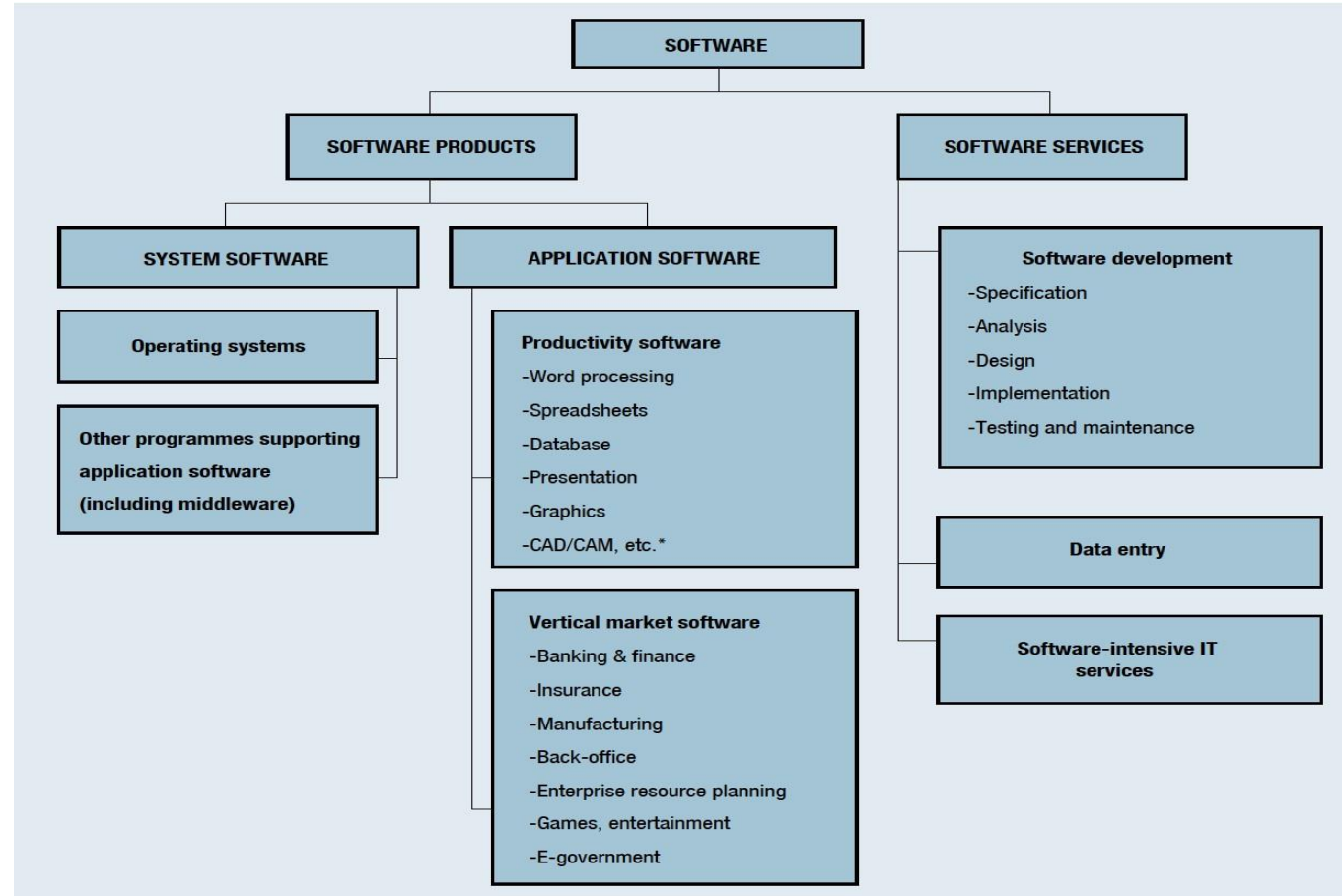
- Dedicato alla gestione dell'elaboratore
- Esempio: Sistema Operativo (Windows, Linux, etc)

## Software applicativo:

- Dedicato alla realizzazione di specifiche applicative
- Esempio: programmi per scrittura, gestione aziendale, navigazione su internet, ecc

# Software

## Tipi di software



# Ciclo di vita del software

- Analisi
  - Esplicitazione dei requisiti
  - Analisi dei requisiti
  - Analisi del dominio
  - Analisi di fattibilità
  - Analisi dei costi
- Progettazione
  - Progetto architetturale
  - Progetto di dettaglio
- Programmazione
  - Costrutti ed esempi
- Ispezione e Debugging
- Collaudo
- Deployment
- Manutenzione

# Risolvere un problema

Dato un problema, l'individuazione delle procedure necessarie alla sua soluzione richiede le seguenti fasi:

- descrizione dei requisiti che la soluzione dovrà soddisfare per essere considerata corretta (specifiche);
- procedimento con cui si individua o si inventa una soluzione (progetto);
- tecnica che consente di risolvere il problema (soluzione).

Nell'informatica, la **soluzione** è espressa tramite un **algoritmo**.

# Algoritmo

Dato un **problema** e un **esecutore**, l'algoritmo:

- è una **successione finita** di passi elementari (direttive);
- eseguibile **senza ambiguità** dall'esecutore;
- risolve il **problema** dato.



# Caratteristiche di un algoritmo

Ogni algoritmo possiede le seguenti caratteristiche:

- **sequenzialità:**  
viene eseguito un passo dopo l'altro secondo un ordine specificato (**flusso di esecuzione**);
- **univocità:**  
i passi elementari devono poter essere eseguiti in modo **univoco** dall'esecutore, e quindi devono essere **descritti** in una forma eseguibile per l'esecutore.

# Elementi di un algoritmo

- **Dati:** le entità su cui opera l'algoritmo vengono generalmente chiamate dati, e comprendono sia i **dati iniziali** del problema che i **risultati intermedi**.
- **Operazioni:** gli interventi che si possono effettuare sugli oggetti, cioè sui dati sono calcoli, confronti, assegnamenti ed operazioni di I/O.
- **Flusso di controllo:** indica le possibili evoluzioni dell'esecuzione delle operazioni, cioè le possibili successioni dei passi dell'algoritmo.

# Descrizione di un algoritmo

Un algoritmo adatto per un esecutore automatico deve essere descritto tramite un **formalismo** (linguaggio) rigoroso e non ambiguo costituito da:

- **vocabolario**: insieme di elementi per la descrizione di oggetti, operazioni e flusso di controllo;
- **sintassi**: insieme di regole per la composizione degli elementi del linguaggio in frasi eseguibili e costrutti di controllo;
- **semantica**: insieme di regole per l'interpretazione degli elementi e delle istruzioni sintatticamente corrette.

# Linguaggio di programmazione

In un linguaggio di programmazione:

- gli **oggetti** vengono descritti tramite **nomi simbolici** (detti anche identificatori);
- le **operazioni** vengono descritte tramite **operatori**, **funzioni** e **procedure**;
- il **flusso di controllo** viene descritto tramite opportuni **costrutti di controllo**.

# Flusso di controllo vs. flusso di esecuzione

Il **flusso di controllo** è differente dal **flusso di esecuzione**.

Il flusso di controllo descrive tutte le **possibili** successioni di **operazioni** che possono essere realizzate dal programma nella sua esecuzione.

Il flusso di esecuzione è la **sequenza** di operazioni percorsa durante una **particolare esecuzione** del programma.

# Paradigmi di programmazione (1)

Un linguaggio di programmazione è un **metodo di codifica** che prevede una serie di istruzioni interpretabili ed eseguibili dai computer. I processori (CPU) dei computer sono in grado di elaborare solo istruzioni in codice macchina, cioè codice binario composto da 0 e 1 che come si può facilmente intuire è molto **difficile da interpretare** per un uomo.

# Paradigmi di programmazione (2)

Sono nati pertanto i diversi linguaggi di programmazione con i quali si può progettare e scrivere **in modo più semplice** un codice sorgente che, una volta compilato, fornisce il programma binario effettivamente eseguibile dall'elaboratore.

Esistono molte tecniche e metodologie di programmazione che si possono raggruppare in tre macro paradigmi:

- **Procedurale** (*Procedural Programming*),
- **Strutturata** (*Structured Programming*),
- **Orientata agli Oggetti** (*OOP Object Oriented Programming*).

# Programmazione Procedurale (1)

Questo stile di programmazione può ricondotto all'utilizzo di processori un po' vecchioti con un **approccio sequenziale** all'esecuzione dei comandi.

Con i linguaggi che prevedono una codifica procedurale, le **istruzioni** vengono eseguite in serie **una dopo l'altra**, per la **lettura** e la **modifica** di una memoria condivisa (memoria RAM, disco rigido) e nella gestione delle periferiche (tastiera, monitor, stampante, ecc.).

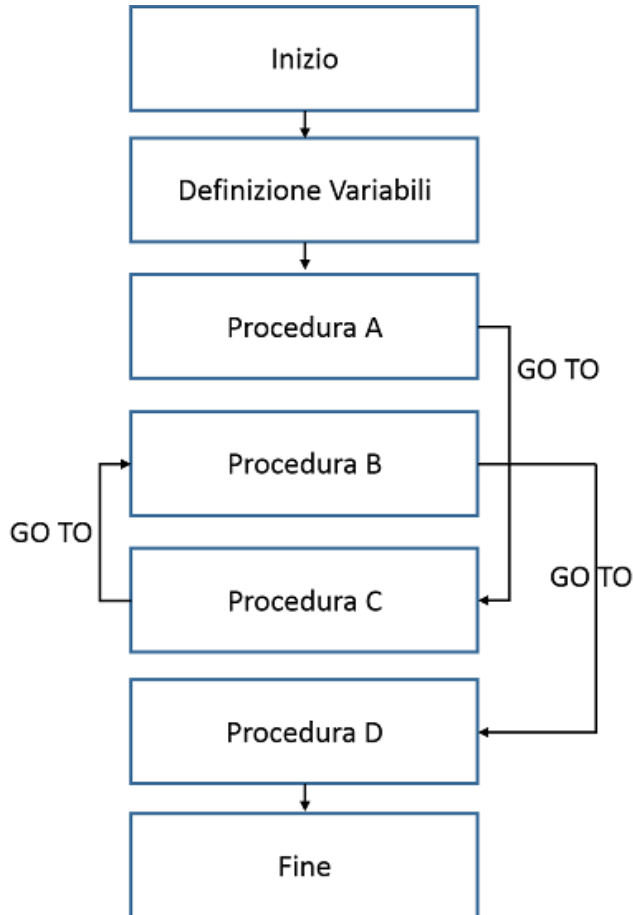
Nei programmi procedurali le istruzioni sono organizzate in **blocchi funzionali**, procedure progettate per uno scopo specifico, anche denominate subroutine o function a seconda del linguaggio e del ruolo all'interno del programma stesso, che possono essere raggiunte in **cascata** oppure utilizzando il **comando go to** da più punti del codice.

Il codice viene scritto ed eseguito dall'elaboratore passo dopo passo dall'inizio alla fine (**top-down**) quindi è molto facile da seguire soprattutto nei programmi più semplici.

Esempi di linguaggi con approccio procedurale sono: Cobol, Basic e Fortran.



# Programmazione Procedurale (2)



# Programmazione Strutturata (1)

La programmazione strutturata, evoluzione di quella procedurale, pur continuando ad avere un approccio **top-down** enfatizza maggiormente la **separazione** dei dati trattati rispetto alle operazioni del programma stesso. L'esecuzione delle istruzioni non è più fatta obbligatoriamente in ordine sequenziale ma è gestita con un **flusso logico** e condizionata con loop, sequenze e decisioni (if, switch) **basate sul valore dei dati**.

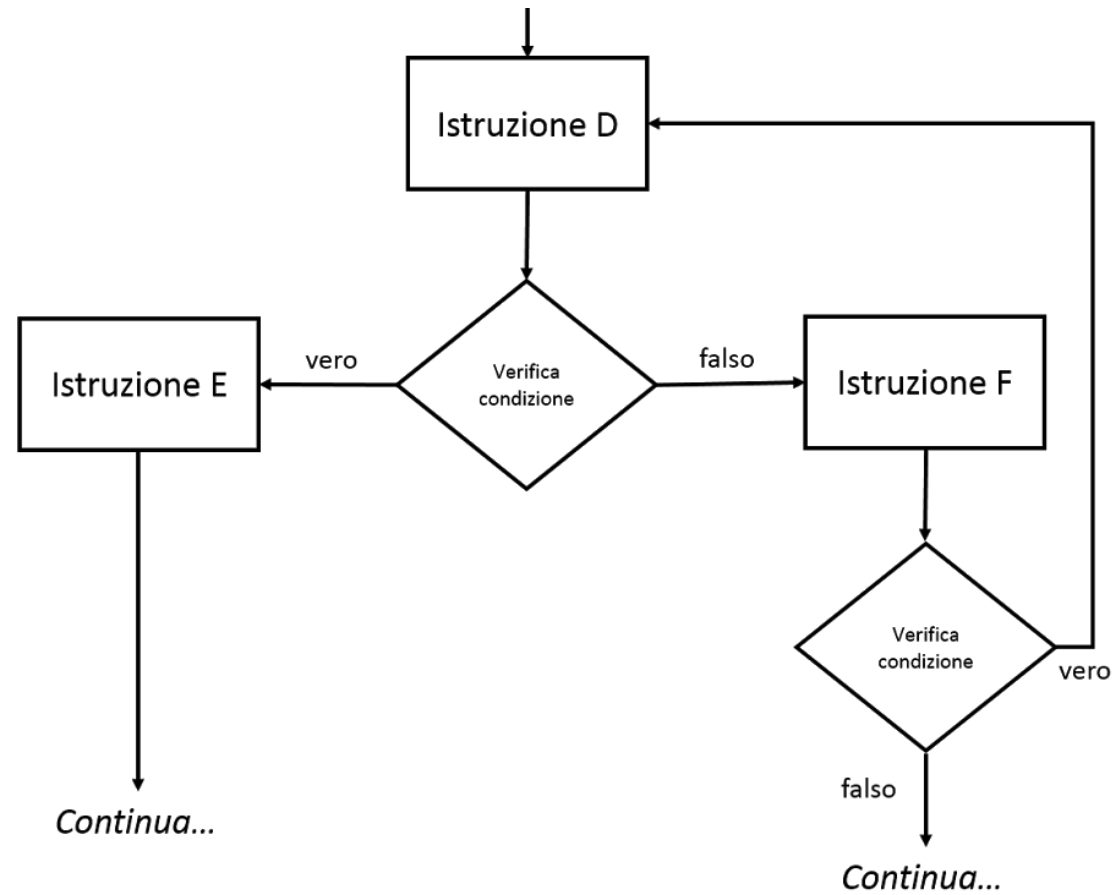
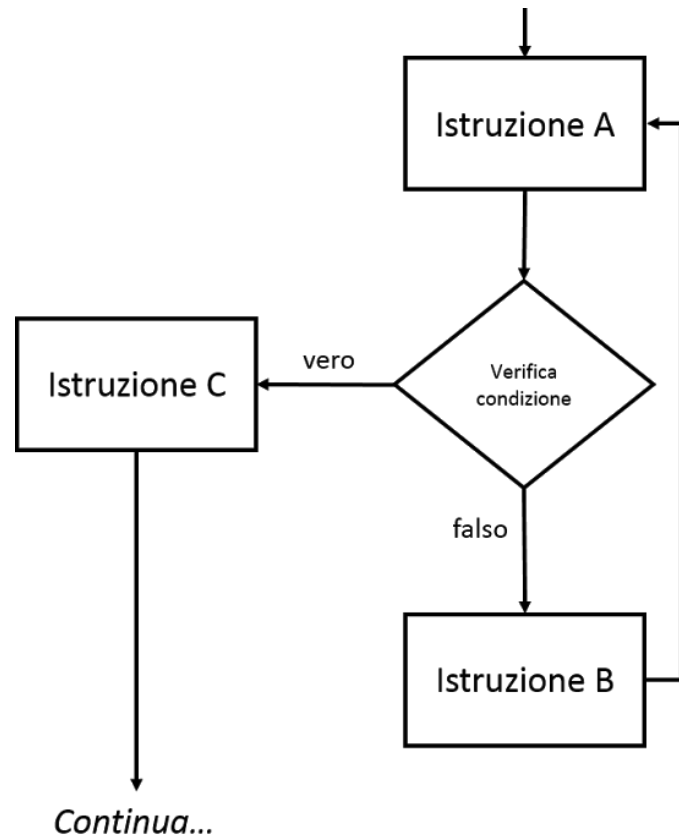
La progettazione e la codifica strutturata di un software non si limita ad un approccio sequenziale dall'alto verso il basso ma si concentra sulla **scomposizione** del problema in **sotto-parti** (funzioni, moduli e librerie esterne) con la conseguente **semplificazione** del processo principale;

Il codice viene strutturato in singoli moduli che implementano specifiche funzionalità.

Vengono utilizzate le strutture di controllo per estrarre in modo deterministico, in base ai dati trattati in quel contesto, l'ordine esatto in cui i set di istruzioni devono essere eseguiti (un codice strutturato **non prevede** l'utilizzo dell'istruzione "goto" ).

*Esempi di linguaggi con approccio strutturato sono: C, Pascal, PL/I e Ada.*

# Programmazione Strutturata (2)



# Spaghetti code

Spaghetti code è un termine dispregiativo per il **codice sorgente** di quei programmi che hanno una **struttura di controllo** del flusso complessa e/o incomprensibile, con uso esagerato ed errato di **go to**, **eccezioni** e altri costrutti di branching (diramazione del controllo) non strutturati.

Il suo nome deriva dal fatto che questi tipi di codice tendono a assomigliare a un **piatto di spaghetti**, ovvero un mucchio di fili intrecciati ed annodati.

# Procedurale vs Strutturato

Il seguente programma è un esempio banale di spaghetti code in BASIC

```
10 dim i
20 i = 0
30 i = i + 1
40 if i <= 10 then goto 70
50 print "Programma terminato."
60 end
70 print i & " al quadrato = " & i * i
80 goto 30
```

Esempio di codice equivalente scritto con uno stile di programmazione strutturato:

```
function square(i)
    square = i * i
end function
dim i
for i = 1 to 10
    print i & " al quadrato = " & square(i)
next print "Programma terminato."
```

Le istruzioni di «**GOTO**» introducono una dipendenza dai numeri di riga del programma, e come il flusso di esecuzione salta in maniera **imprevedibile** da una zona all'altra. In pratica, nei programmi reali le occorrenze di **spaghetti code** sono ben più complesse e possono aumentare notevolmente i costi di **manutenzione** di un programma.

# Programmazione ad Oggetti (1)

I linguaggi **Object Oriented** attualmente sono i più **largamente usati** ed i più **potenti** in relazione alle architetture hardware e software nei quali vengono utilizzati. I programmi sono scritti come una **collezione** di **oggetti auto-consistenti** che comunicano tra di loro.

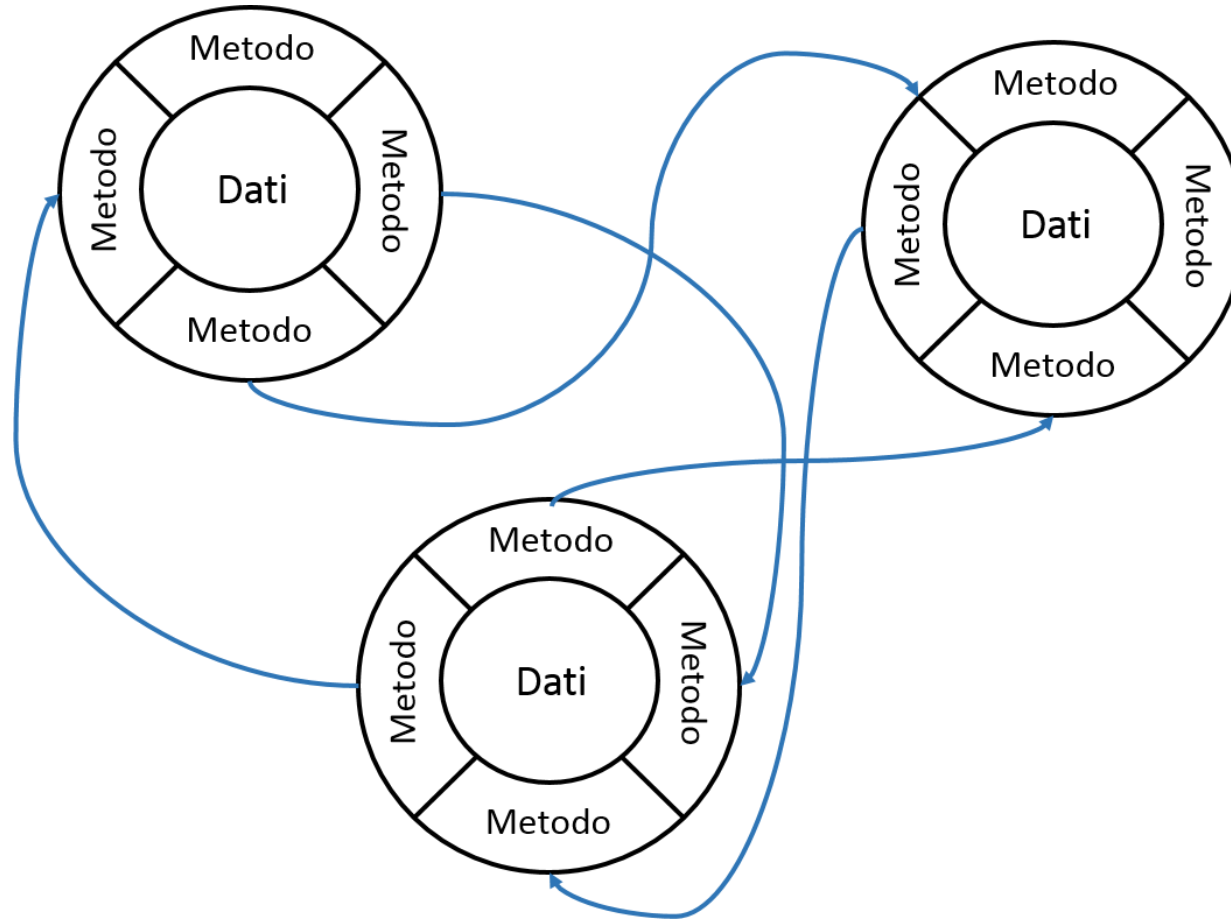
Si basano sulle **classi**, cioè istanze di oggetti software nei quali rivestono maggiore importanza i **dati** piuttosto che il flusso logico.

Una classe può essere definita come un **contenitore** che abbina i **dati** con le **operazioni/azioni** che possono essere eseguite su di essi. Le caratteristiche di ciascun oggetto (classe) vengono chiamate **proprietà** o attributi, mentre le azioni sono dette **metodi**.

Diversamente dai linguaggi precedentemente descritti, gli **OOP** hanno un approccio di progettazione **bottom-up**, per prima cosa non va definita la struttura del programma generale ma vanno **progettati** gli **oggetti**, le modalità con le quali gestiscono le proprie informazioni e le interfacce con cui interagiscono con gli altri oggetti.

*Esempi di linguaggi Object Oriented sono: C++, Java e C#.*

# Programmazione ad Oggetti (2)



# Linguaggi altre definizioni (1)

I linguaggi di programmazione sono classificabili in almeno quattro grandi categorie, parzialmente sovrapposte:

- **Imperativi**: le istruzioni descrivono in modo esplicito le modifiche del contenuto della memoria;

esempi: *Basic, Pascal, C, Fortran, Cobol*;

- **Funzionali**: il programma è costituito da una funzione che ha per argomenti altre sottofunzioni;

esempio: *Lisp*;



# Linguaggi altre definizioni (2)

- **Dichiarativi (o logici)**: il programma è costituito da una serie di asserzioni e di regole, e la sua esecuzione consiste in una dimostrazione di veridicità di un'asserzione, senza indicare il flusso di esecuzione;  
esempio: *Prolog*;
- **Orientati agli oggetti**: il programma è costituito da entità (oggetti) che comunicano tra loro scambiandosi messaggi e che godono delle proprietà di incapsulamento, ereditarietà e polimorfismo;  
esempi: *Ada, Smalltalk, Java*.

# Linguaggi di specifica

- un linguaggio di specifica è un linguaggio formale (o parzialmente tale) usato per descrivere un **sistema software** a un **livello di astrazione** superiore a quello dei **linguaggi di programmazione**.
- Un linguaggio di specifica può, a seconda dei casi, essere anche un **linguaggio di modellazione**. In questo caso, la descrizione del sistema può in qualche modo essere considerata un **modello**, cioè una rappresentazione semplificata del sistema stesso, che ne riproduce i tratti essenziali.

# Linguaggi di descrizione

I linguaggi semiformali hanno regole ben definite, ma permettono descrizioni in linguaggio naturale, non rigorose.

- Sono usati nella fase di sviluppo di algoritmi:
- **schema a blocchi** (diagramma di flusso): usa blocchi di varie forme geometriche connessi da archi orientati la cui direzione indica il flusso di controllo, mentre la forma dei blocchi indica la natura del blocco stesso (operazione, confronto, operazione di I/O);
- **pseudocodice**: usa strutture di controllo di un linguaggio di programmazione e operazioni descritte in linguaggio naturale.

# Sottoprogrammi

Programmi complessi vengono progettati seguendo uno schema gerarchico in cui il problema iniziale viene suddiviso in sottoproblemi e così via (metodologia **top-down**).

Questo approccio ha i seguenti vantaggi:

- chiarezza del programma principale;
- sintesi;
- efficienza.

# Sottoprogrammi vantaggi

- **Chiarezza**: i dettagli di basso livello sono descritti a parte nei sottoprogrammi, evidenziando la struttura di controllo generale;
- **Sintesi**: per i sottoproblemi presenti in più parti del problema principale, richiamando il sottoprogramma, si evitano di ripetere le stesse sequenze di operazioni;
- **Efficienza**: sottoproblemi di uso comune sono disponibili, già risolti da esperti programmatori, raccolti nelle cosiddette librerie.

# Programma eseguibile

Perché un algoritmo sia eseguibile da un calcolatore, esso deve essere tradotto in una sequenza di istruzioni codificate nel codice operativo caratteristico della CPU del calcolatore che lo eseguirà (**binario**).

# Traduzione in binario (1)

La traduzione in codice eseguibile è effettuata da un programma apposito, che può essere di due tipi:

- **interprete**: traduce solo le istruzioni del flusso di esecuzione;
  - la traduzione viene effettuata ad **ogni esecuzione**;
- **compilatore**: traduce l'intero programma;
  - la traduzione viene effettuata **una sola volta**.

# Traduzione in binario (2)


La generazione di un programma eseguibile consiste nella traduzione automatica

- di un algoritmo espresso in un linguaggio simbolico (**programma sorgente**)
- nello stesso algoritmo espresso in codice macchina (**programma eseguibile**).



# Traduzione in binario (3)

	Linguaggio di codifica	Il computer...
Basso livello	Linguaggio macchina	lo esegue direttamente
	Linguaggio assembly	necessita dell'assembler
	⋮	⋮
Alto livello	Linguaggio di programmazione	necessita di un traduttore



# Catena di programmazione

Le fasi che portano un algoritmo ad essere eseguito sono:

- **editing**: composizione del programma sorgente;
- **compiling**: traduzione in codice binario;
- **linking**: collegamento con i sottoprogrammi di libreria;
- **loading**: caricamento in memoria;
- **esecuzione**: esecuzione del programma.

# Editing

Generalmente un algoritmo viene codificato in un programma sorgente.

Lo strumento utilizzato per la scrittura è un apposito programma chiamato editor.

Questa fase ha termine con la produzione di un file di testo detto file **programma sorgente**.

# Compiling

La fase di traduzione, o **compilazione**, utilizza un programma (detto **compilatore**).

Esso elabora il **file sorgente**, riconoscendo i simboli, le parole e i costrutti del linguaggio e producendo:

- la **forma binaria** del codice macchina corrispondente (file programma oggetto);
- oppure, una segnalazione degli **errori sintattici**.

# Errore sintattico

Gli errori sintattici sono **violazioni** delle **regole** sintattiche del linguaggio di programmazione.

Essi rendono il programma non associabile ad un significato e quindi ne impediscono la traduzione.

Esempio:  $3 * (5 + / 2)$

# Linking

Nella fase di **linking** (collegamento), un programma (detto **linker**) collega il file oggetto con le funzioni di libreria.

Esso produce:

- un **programma eseguibile**;
- oppure, una segnalazione di errore nella citazione delle routine di libreria (linker error).

# Loading

Per poter essere eseguito, un file eseguibile deve essere caricato in **memoria** centrale.

Questa funzione viene svolta da un programma chiamato **loader** (caricatore), che individua una regione di memoria adeguata all'esecuzione del programma.

Se tale spazio in memoria non esiste, segnala un errore di caricamento per memoria insufficiente.

# Esecuzione

Il programma in esecuzione elabora i dati in ingresso e produce i risultati in uscita.

Possibili errori (non di tipo sintattico!):

- calcoli matematicamente impossibili (**run-time error**); – esempio: divisione per zero;
- operazioni fisicamente impossibili (**run-time error**); – esempio: memoria insufficiente;
- calcoli scorretti;
  - esempio: **overflow**;



