Programmazione OO

Java

Milestone

- Definire un algoritmo
- Strutture di controllo
- Bohm-Jacopini Theorem
- Programmazione procedurare vs OOP
- Tipi di Dati
- Ereditarieta’
- Polimorfismo
- Incapsulamento

Programmazione procedurale

- La **programmazione procedurale** consiste nel creare dei blocchi di codice, identificati da un nome e racchiusi da dei delimitatori, che variano a seconda del linguaggio di programmazione; questi sono detti anche **sottoprogrammi**. Questi blocchi possono assumere i nomi di procedure o funzioni, a seconda del linguaggio e dei loro ruoli all'interno del linguaggio stesso. Il nome deriva dal linguaggio COBOL, che è stato il primo ad utilizzare questo concetto.

Un po’ di storia

- Il linguaggio **Java** è un linguaggio di programmazione orientato agli oggetti, derivato dal C++ e creato da James Goslin e altri ingegneri di Sun Microsystems. Il gruppo iniziò a lavorare nel 1991, il linguaggio inizialmente si chiamava Oak. Il nome fu successivamente cambiato in Java a causa di un problema di copyright (il linguaggio di programmazione Oak esisteva già nel 1991). Java fu annunciato ufficialmente il 23 maggio 1995 a SunWorld. La piattaforma di programmazione Java è fondata sul linguaggio stesso, sulla Java Virtual Machine (JVM) e sulle API.

Definizioni

- La **programmazione orientata agli oggetti** (**OOP**, **O**bject **O**riented **P**rogramming) è un paradigma di programmazione, che prevede di raggruppare in un'unica entità (la classe) sia le strutture dati che le procedure che operano su di esse, creando per l'appunto un "oggetto" software dotato di proprietà (dati) e metodi (procedure) che operano sui dati dell'oggetto stesso.

QUINDI???

Caratteristiche OOP 1/2

- L'idea principale della programmazione ad oggetti consiste nel rendere il software la rappresentazione di entità reali o astratte ma ben definite (oggetti). Questi oggetti, come nella vita pratica, hanno **proprietà** rappresentate da valori, e qualità o meglio **metodi**: ciò che sanno fare questi oggetti. Si pensi ad una automobile: ha delle proprietà come il colore o il numero di porte, e dei metodi, per esempio può girare a destra o sinistra, andare avanti o indietro, accelerare, decelerare ecc. Riportando la programmazione in questi termini è facile capire come questo renda più facile la gestione di grandi progetti, migliorarne la qualità e la mantenibilità.

Caratteristiche OOP 2/2

- La seconda caratteristica e' l'indipendenza dalla piattaforma. Cio' significa che l'esecuzione di programmi scritti in Java deve avere un comportamento simile su hardware diverso. Si dovrebbe essere in grado di scrivere il programma una volta e farlo eseguire dovunque. Questo è possibile con la compilazione del codice di Java in un linguaggio intermedio **bytecode**, basato su istruzioni semplificate che ricalcano il linguaggio macchina. Esso viene eseguito da una virtual machine, cioè da un interprete: Java è quindi, in linea di massima, un linguaggio interpretato. Inoltre, vengono fornite librerie standardizzate per permettere l'accesso alle caratteristiche della macchina (come grafica e networking) in modo unificato

Classi

- Le classi sono uno strumento per costruire strutture dati che contengano non solo dati ma anche il codice per gestirli.Come tutti i costrutti che permettono di definire le strutture dati, una classe definisce un nuovo tipo di dato (ADT) .I membri di una classe sono dati, divisi in **attributi**, e **metodi** che operano su un oggetto.

Oggetti

- Un oggetto è una **istanza** di una classe. Un oggetto occupa memoria, la sua classe definisce come sono organizzati i dati in questa memoria.Ogni oggetto possiede tutti gli attributi definiti nella classe, ed essi hanno un valore, che può mutare durante l'esecuzione del programma come quello di qualsiasi variabile. Il paradigma OOP suggerisce un principio noto come information hiding che indica che si debba accedere agli attributi dell'istanza solo tramite metodi invocati su quello stesso oggetto.

Metodi

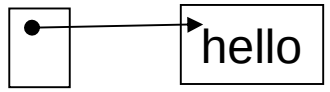
- Indicare un sottoprogramma associato in modo esclusivo ad una classe (**metodo statico**) oppure ad un oggetto (**metodo di istanza**). Solitamente consiste in una sequenza di istruzioni scritte per eseguire una determinata azione, eventualmente personalizzata da un set di parametri e in grado di restituire al programma chiamante un *valore di ritorno* (o di *output*) di un determinato tipo.
- public int calcolaSomma(int x, int y)

Tipi di dati primitivi in Java

- Interi
  - byte (8 bit)
  - Short (16 bit)
  - Int (32 bit)
  - long (64 bit)
- Reali
  - float (32 bit)
  - double (64 bit)
- Boolean (1 bit)
- Char (16 bit) NOTA: I caratteri in JAVA sono Unicode, quindi 8 bit non sono sufficienti!
- Le variabili vengono memorizzare in memoria (Stack).

Tipi di dato astratto in Java (ADT)

- Le CLASSI
  - Es String s = “hello”;



- Gli oggeti vengono memorizzare in memoria (Heap).

Creazione/Operatori (tipi primitivi)

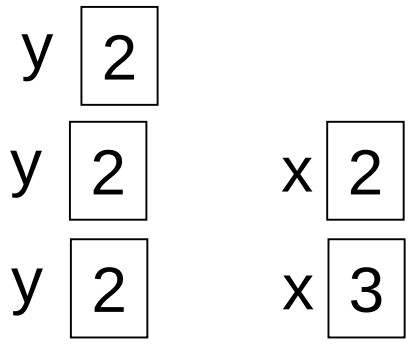
- Creazione int x = 3;
  - Addizione +
  - Sottrazione –
  - Moltiplicazione *
  - Divisione /
  - Resto della divisione intera % (modulo)

Creazione/Operatori (tipi ADT)

- Creazione StringBufferer s = new StringBuffer (“hello”);
  - Tipo oggetto: StringBufferer
  - Nome oggetto: s
  - Costruttore: new
  - Parametro del costruttore: hello
- Modifica di un oggetto s.append(“!”);
  - Metodo invocato: append
  - Operatore per invocare il metodo: .

Tipi primitivi COPIA PER VALORE

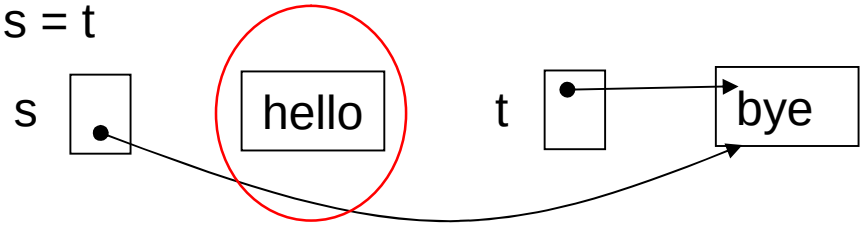
- y = 2;
- x = y;
- x = 3;



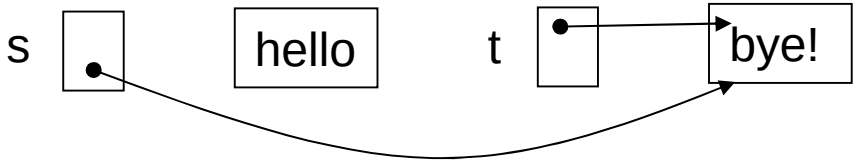
Tipi derivati COPIA PER RIFERIEMNTO

- StringBufferer s = new StringBuffer (“hello”);
- StringBufferer t = new StringBuffer (“bye”);

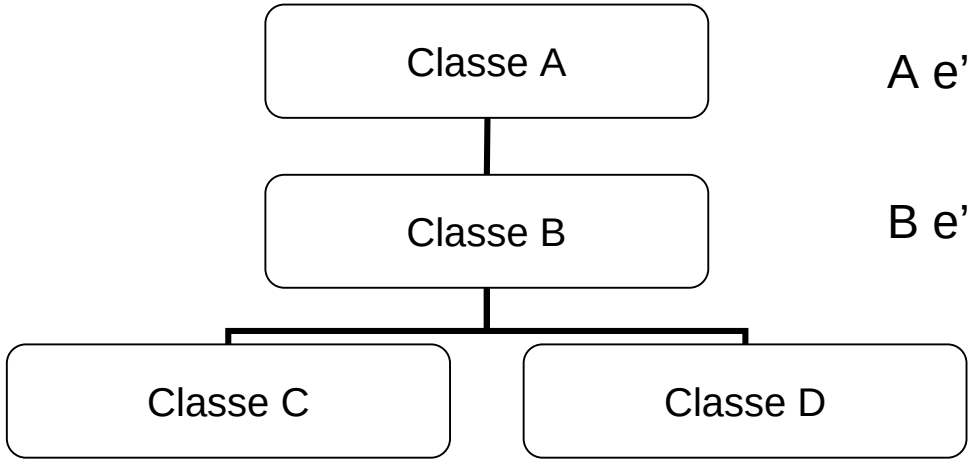
- s = t



- Oggetto perduto, GARBAGE COLLECTION
- s.append(“!”);



Ereditarieta’



A e’ la clase base di B quando B estende A

B e’ la sottoclase base di A

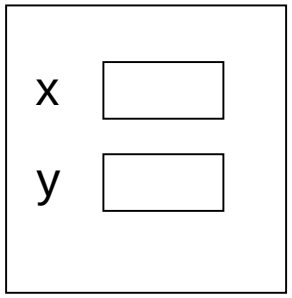
Una classe ha dei campi ed esporta dei metodi, la classe derivata aggiunge nuovi campi e nuovi metodi ma mantiene quelli pre-esistenti.

Esempio di ereditarieta’

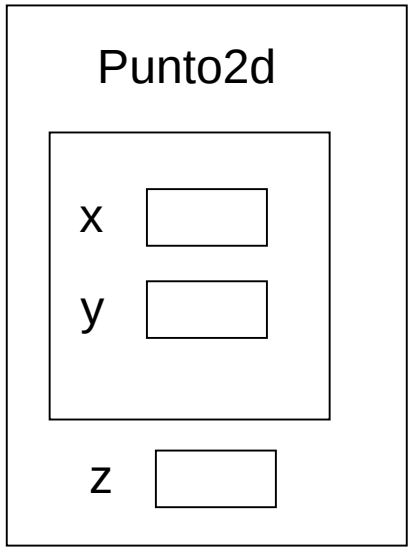
```
class Punto2d{
        int x=0;
        int y=0;
}
```

```
class Punto3d extend Punto2d{
        int z=0;
}
```

Punto2d



Punto3d



Incapsulamento

- L'incapsulamento è la proprietà per cui un oggetto contiene ("incapsula") al suo interno gli attributi (dati) e i metodi (procedure) che accedono ai dati stessi. Lo scopo principale dell'incapsulamento è appunto dare accesso ai dati incapsulati solo attraverso i metodi definiti, nell'interfaccia, come accessibili dall'esterno. Gestito in maniera intelligente, l'incapsulamento permette di vedere l'oggetto come una black-box, cioè una scatola nera di cui, attraverso l'Interfaccia sappiamo cosa fa e come interagisce con l'esterno ma non come lo fa . Il vantaggi principali portati dall'incapsulamento sono: **robustezza**, **indipendenza** e l'estrema **riusabilità** degli oggetti creati. (API di JAVA)

Polimorfismo

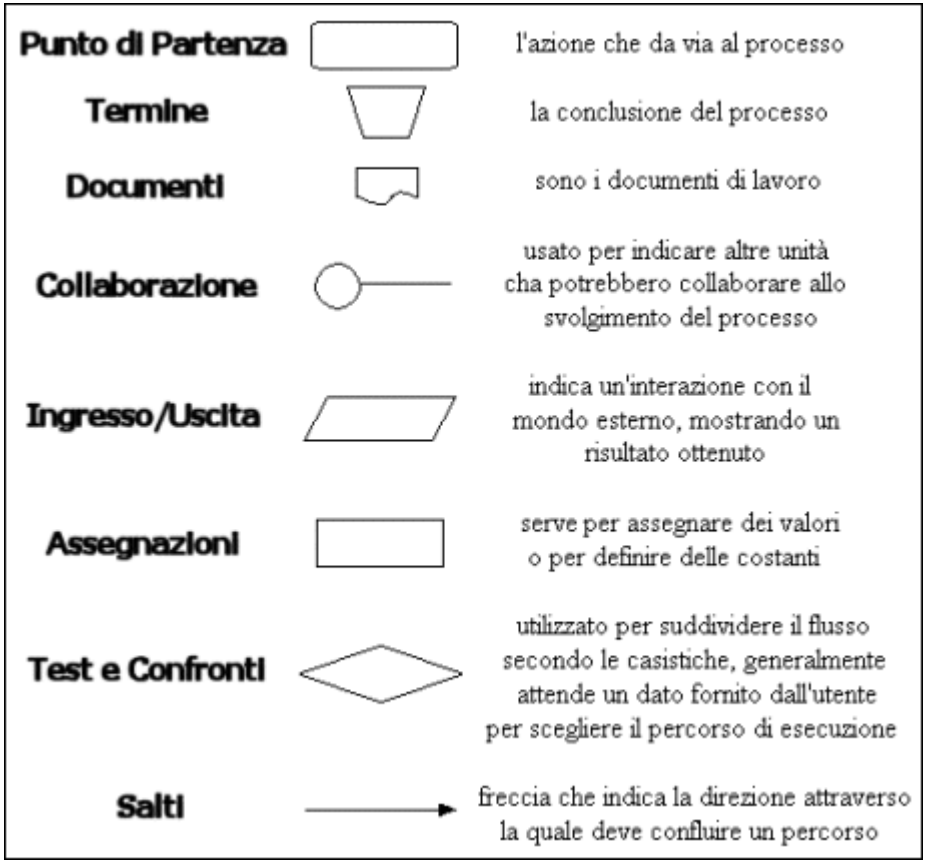
- La possibilità che le classi derivate implementino in modo differente i metodi e le proprietà dei propri antenati rende possibile che gli oggetti appartenenti a delle sottoclassi di una stessa classe rispondano diversamente alle stesse istruzioni.
- Esempio: in una gerarchia in cui le classi Cane e Gatto discendono dalla SuperClasse Animale potremmo avere il metodo mangia() che restituisce la stringa "osso" se eseguito sulla classe Cane e "pesce" se eseguito sulla classe Gatto.
- I metodi che vengono ridefiniti in una sottoclasse sono detti "**polimorfi**", in quanto lo stesso metodo si comporta diversamente a seconda del tipo di oggetto su cui è invocato.

Sintassi

- Dichiarazione/Inizializzazione int x = 0;
- Assegnamento x = x + 1;
- Confronto if(x==2) x = x +1;

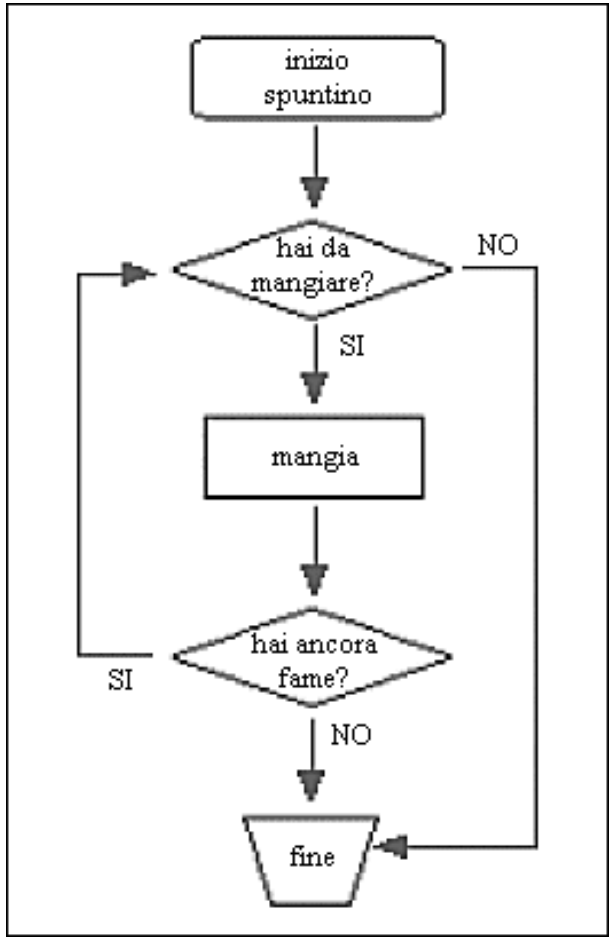
Diagrammi di flusso

- sono dei disegni che rappresentano graficamente un nostro ragionamento rappresentandolo come algoritmo, permettendo una comprensione immediata del funzionamento del nostro percorso logico, mantenendo un controllo accurato sulla funzionalità e la casistica del ragionamento.



Esempio

- Inizia lo spuntino
- Hai qualcosa da mangiare?
- Mangia
- Hai ancora fame?
- Fine dello spuntino



Algoritmo

- Insieme di istruzioni elementari (univocamente interpretabili) che, eseguite in un ordine stabilito, permettono la soluzione di un problema in un numero finito di passi , ossia un metodo per la soluzione di un problema adatto a essere implementato sotto forma di programma.
- Proprieta':
  - La sequenza di istruzioni deve essere **finita**
  - Essa deve portare ad un **risultato**
  - Le istruzioni devono essere **eseguibili** materialmente
  - Le istruzioni non devono essere **ambigue**

Paradigma di programmazione

- è un insieme di strumenti concettuali forniti da un linguaggio di programmazione per la stesura di programmi, e definisce/determina il modo in cui il programmatore concepisce e percepisce il programma. Un programma object-oriented, per esempio, è costituito da oggetti che interagiscono fra loro, mentre nella programmazione procedurale il programma è una composizione di funzioni.

Programmazione strutturata

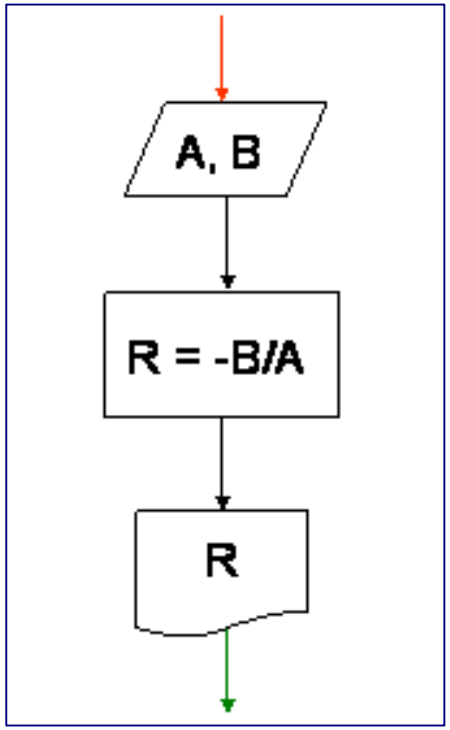
- La **programmazione strutturata** è un paradigma di programmazione emerso fra gli anni '60 e gli anni '70 nel contesto della programmazione procedurale. I concetti introdotti dalla programmazione strutturata sono alla base di numerosi altri paradigmi procedurali successivi, non ultimo quello *object-oriented*.
- Le strutture di controllo ammesse dai linguaggi strutturati, e le regole sintattiche e semantiche del loro uso, possono variare nei dettagli specifici; tuttavia, devono essere rispettati un insieme di requisiti fondamentali:
  - **completezza**. Un linguaggio strutturato deve fornire la sequenza, almeno una struttura di tipo *alternativa*, e almeno una struttura di tipo *iterazione*;
  - **singolo punto di ingresso e di uscita**. Idealmente, ogni struttura di controllo, completa delle istruzioni *controllate*, deve poter essere considerata come una *singola macro-istruzione* dal punto di vista del controllo complessivo, con un ben identificato punto di ingresso e un ben identificato punto di uscita.
  - **componibilità**. Ogni struttura di controllo può avere fra le sue istruzioni controllate, ricorsivamente, altre strutture di controllo (senza limiti).

Bohm-Jacopini Theorem

- Enunciato nel 1966 dagli informatici Corrado Böhm e Giuseppe Jacopini, afferma che qualunque algoritmo può essere implementato utilizzando tre sole strutture, la **sequenza**, la **selezione** ed il **ciclo**, da applicare ricorsivamente alla composizione di istruzioni elementari
- Le strutture di controllo del flusso dati fondamentali sono:
  - Sequenza
  - Selezione
  - Iterazione

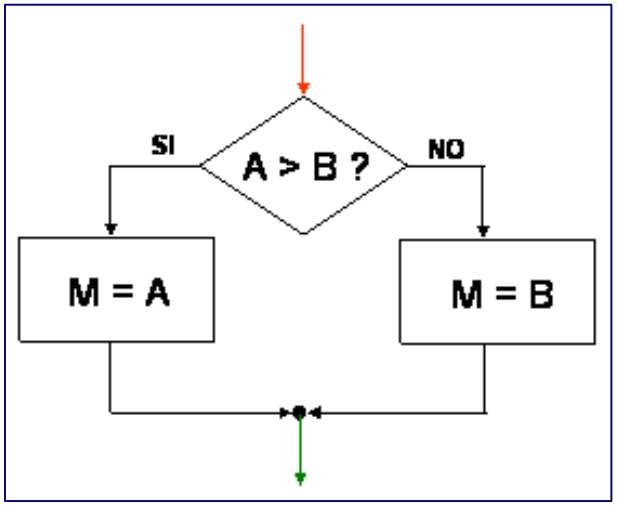
Sequenza

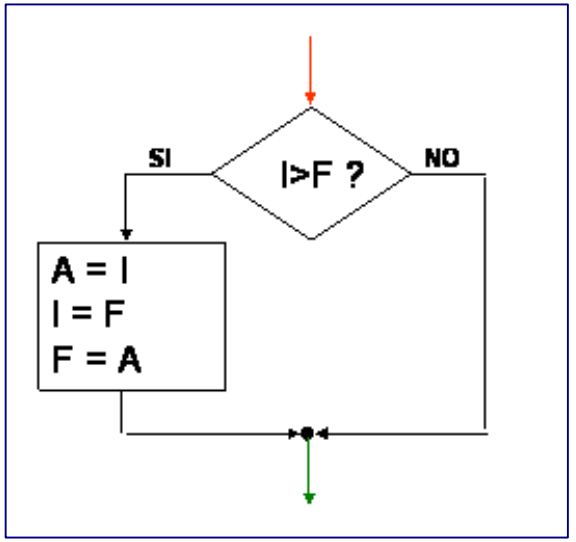
- E’ la normale elencazione di istruzioni perché vengano eseguite una di seguito all'altra nell'ordine in cui sono state scritte dal programmatore.



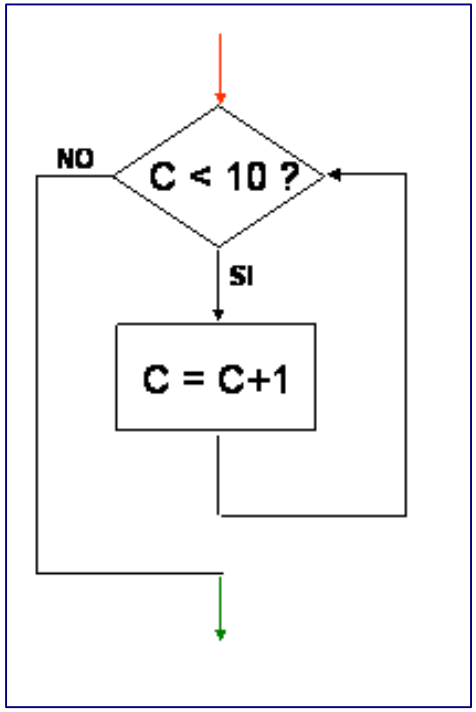
Selezione

- E’ la scelta fra due percorsi da seguire successivamente, che dipende da una condizione che può essere vera o falsa

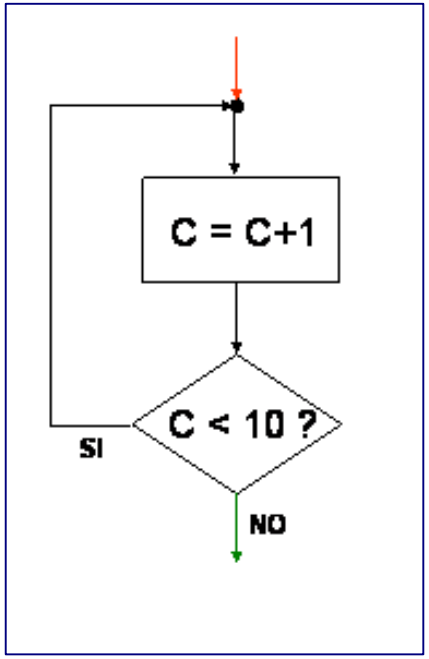




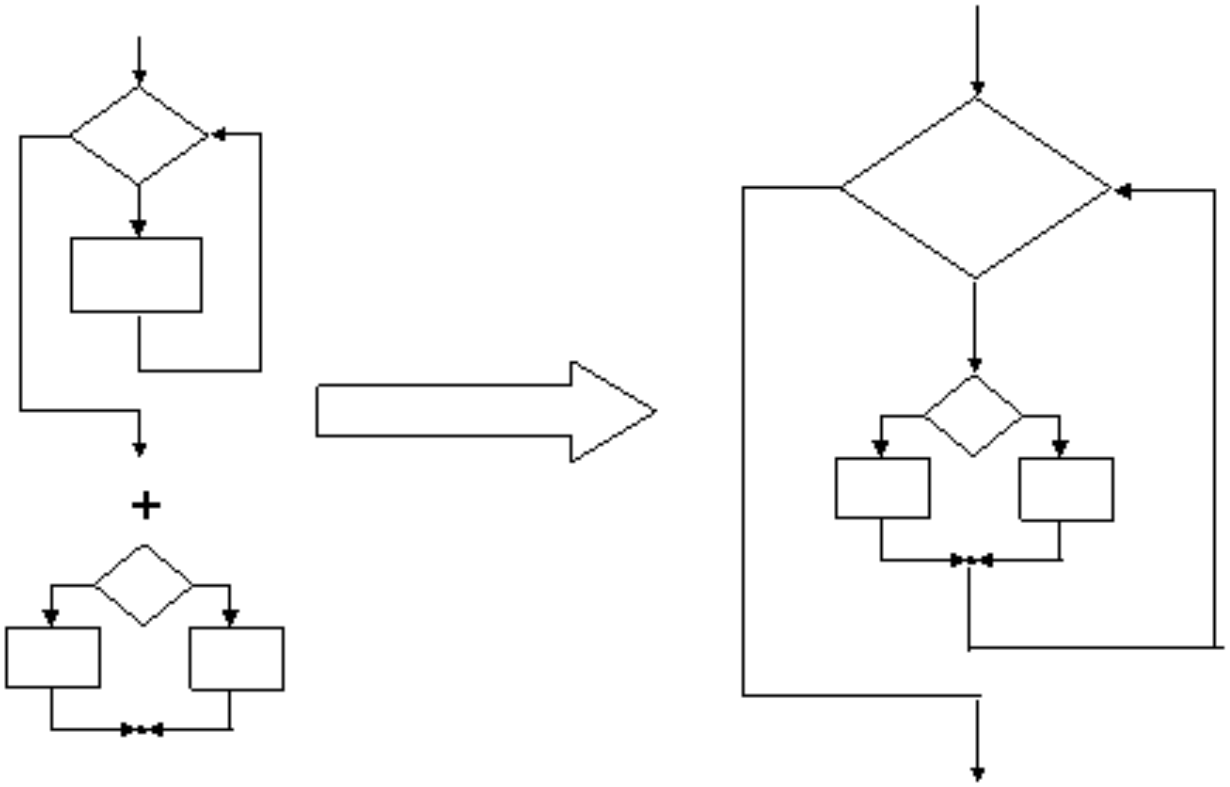
Iterazione



- E’ un blocco di istruzioni che vengono ripetutamente eseguite fino a che una certa condizione cambia di stato



Componibilita’ delle strutture



viene *iterativamente* ripetuta una *alternativa*.