



Linguaggi: Sintassi e Semantica

Il linguaggio Java

A cura di A. Orlandini

Obiettivi

- ❑ **Distinguere la sintassi dalla semantica di un linguaggio**
- ❑ **Conoscere le convenzioni del meta-linguaggio BNF (EBNF)**
- ❑ **Riconoscere la sintassi e la semantica delle istruzioni strutturate del linguaggio Java**
- ❑ **Descrivere la risoluzione di un qualunque problema di programmazione attraverso le sole tre istruzioni strutturate (Teorema di Jacopini Böhm)**

Contenuti

- Definizione di un linguaggio**
- Sintassi e semantica**
- Linguaggi di programmazione e grammatiche**
- Meta-linguaggio BNF**
- Sintassi dei linguaggi di programmazione**
- Sintassi del linguaggio Java**
- Semantica del linguaggio Java**
- Sintassi, semantica ed errori**

Linguaggi naturali . . .

- ❑ Per definire un **linguaggio naturale** si parte dalla definizione di un **alfabeto**
 - in italiano ci sono 21 lettere, in inglese 26, . . .
- ❑ Con i caratteri dell'alfabeto possiamo formare un **insieme di sequenze**, dette **parole**
- ❑ Non tutte le sequenze sono parole del linguaggio naturale
- ❑ La **grammatica** del linguaggio fornisce le regole per decidere quali sequenze sono parole del linguaggio
 - parole **corrette grammaticalmente**

. . . Linguaggi naturali

□ Con le lettere dell'alfabeto italiano possiamo costruire alcune sequenze

- ad esempio **abcdef, ghil, rst** - che **non sono parole** della lingua italiano
- ad esempio **andare, aula, corso, acqua, soquadro**, - che **sono parole** della lingua italiano, cioè **corrette grammaticalmente**

. . . Linguaggi naturali

- ❑ Con le parole, corrette, possiamo formare un **insieme di parole, dette frasi**
- ❑ Non tutte le sequenze sono frasi del linguaggio naturale
- ❑ La **sintassi** del linguaggio fornisce le regole per decidere quali sequenze sono **frasi** del linguaggio
 - frasi **corrette sintatticamente**, o sintatticamente **ben formate**

... Linguaggi naturali

□ In italiano la regola base della sintassi dice che le frasi sono costruite con sequenze di parole che seguono la struttura

soggetto verbo complemento

- soggetto, verbo e complemento non sono altro che dei nomi, cioè **denotano**, alcuni particolari e ben precisi **sottoinsiemi** dell'insieme di tutte le parole del linguaggio
- ad esempio la sequenza di parole **il lo la** non è quindi una **frase** della lingua italiana
- ad esempio la sequenza di parole **gatto mangia topo** è una **frase** della lingua italiana, ovvero è sintatticamente **ben formata**

... Linguaggi naturali

- ❑ Solo alcune delle **frasi** del linguaggio, cioè di quelle **ben formate** sono anche **valide**, cioè hanno un significato
- ❑ La **semantica** del linguaggio stabilisce quali tra le **frasi** ben formate sono anche **valide** e quindi si occupa dell'**interpretazione** (del **significato**) delle frasi
 - ad esempio la frase **il gatto mangia il topo** è una ben formata ma è anche **valida**, cioè **ha un significato**

Sintassi

□ La **sintassi** di un linguaggio si occupa della forma delle frasi del linguaggio, ovvero delle regole che permettono di costruire frasi ben formate del linguaggio

□ Esempio di frase in italiano

il gatto mangia il topo

□ Frammento della sintassi della lingua italiana

frase → *soggetto verbo complemento*

soggetto → *articolo nome*

verbo → **mangia, beve**

complemento → *articolo nome*

articolo → **il, lo, la**

nome → **gatto, monte, topo**

Semantica

- La **semantica** di un linguaggio si occupa dell'interpretazione del linguaggio, ovvero del significato delle frasi corrette sintatticamente
- Esempio di frasi corrette sintatticamente in italiano, ma non tutte valide
 - il **gatto mangia il topo**
 - il topo mangia il monte
 - il **cane mangia il topo**
 - il monte beve il cane

Regole sintattiche . . .

- ❑ Le regole della **sintassi** sono chiamate **regole di produzione**, come nell'esempio precedente
- ❑ Nelle regole di produzione compaiono **elementi (simboli) terminali**, come - **mangia, beve, il, lo, la, gatto, monte, topo** - ed **elementi (simboli) non-terminali**, come – **frase, soggetto, verbo,** – che sono “**categorie sintattiche**” cioè nomi che denotano insiemi di simboli terminali
- ❑ Una fissata categoria sintattica, detta **assioma**, è quella dalla quale deve partire il processo di produzione
- ❑ Nel caso della lingua italiana l'assioma è **frase**

. . . Regole sintattiche

frase → *soggetto verbo complemento*

soggetto → *articolo nome*

verbo → **mangia, beve**

complemento → *articolo nome*

articolo → **il, lo, la**

nome → **gatto, monte, topo**

- Una fissata categoria sintattica, detta **assioma**, è quella dalla quale deve partire il processo di produzione
- Nel caso della lingua italiana l'assioma è **frase**

Linguaggi artificiali e grammatiche

- Un linguaggio di programmazione è un **linguaggio artificiale** e, per poterlo definire in modo rigoroso, introduciamo di seguito alcuni strumenti necessari, con le relative definizioni
 - Alfabeto, o vocabolario
 - Universo linguistico
 - Grammatica, o sintassi
 - Generazione di un linguaggio da una grammatica
- Dopo alcuni esempi vedremo anche possibili **descrizioni alternative per la generazione di un linguaggio da una grammatica**

Universo linguistico

□ Definizione. Dato un insieme finito non vuoto V , si definisce **Universo linguistico su V** , e si indica con V^* , l'insieme delle sequenze finite di lunghezza arbitraria di elementi di V

- L'insieme V viene di solito chiamato **alfabeto**, oppure **vocabolario** o **lessico**. Gli **elementi di V** sono chiamati **simboli terminali**. Si noti che talvolta i simboli di V possono essere più complessi di una singola lettera dell'alfabeto della lingua italiana; per esempio '**main**', '**Class**', '**void**', ecc. sono simboli dell'alfabeto di Java. Gli **elementi di V^*** vengono detti **stringhe** costruite su V , o **frasi su V** .

Linguaggio

□ Definizione. Un linguaggio L sull'alfabeto V è un sottoinsieme di V^* .

- Sebbene V sia finito, V^* non lo è; esso è numerabile, ed i sottoinsiemi di V^* sono in quantità non numerabile.
- Nel considerare i linguaggi di programmazione, non siamo interessati a tutti i sottoinsiemi di V^* , ma solo a quelli che sono descrivibili in maniera finita.
- Questa descrizione può essere per esempio fornita attraverso una grammatica, nel modo che verrà precisato dalle seguenti definizioni.

Grammatica . . .

□ Definizione. Una **grammatica o sintassi G** è definita da:

- **V**, un **alfabeto di simboli terminali**
- **N**, un alfabeto di **simboli non terminali** (detti anche **categorie sintattiche**)
- **S** \in **N**, detto **assioma**, o **simbolo iniziale**, o anche simbolo distinto
- **P**, un insieme finito di **regole sintattiche** (o **produzioni**) del tipo

$$X \rightarrow \alpha$$

dove $X \in N$ ed $\alpha \in (N \cup V)^*$

... Grammatica ...

- Le produzioni sono talora scritte nella forma

$X ::= \alpha$ invece che $X \rightarrow \alpha$

- Se in una grammatica esistono più regole aventi la stessa parte sinistra, ad esempio

$X \rightarrow \alpha_1 \quad X \rightarrow \alpha_2, \dots, X \rightarrow \alpha_n$

esse sono raggruppate, usando la convenzione notazionale

$X \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$

e in tal caso si dice che $\alpha_1, \alpha_2, \dots, \alpha_n$ sono parti destre alternative per X .

... Grammatica

- Gli alfabeti dei simboli terminali e dei simboli non terminali sono disgiunti.
- Per distinguere i simboli di questi due alfabeti spesso si usa una delle due seguenti convenzioni:
 - nella prima i simboli **non terminali** sono distinti dai terminali perché **racchiusi tra parentesi angolate** come ad esempio **<frase>**, **<cifra>**
 - nella seconda i **non terminali** sono scritti **in corsivo** e a volte i **terminali** sono scritti **tra apici** come ad esempio *frase*, *cifra*

Derivazione diretta

□ Definizione. Data una grammatica **G** e due stringhe

$\beta, \gamma \in (N \cup V)^*$, si dice che “ γ deriva direttamente da β in **G**” e si scrive

$$\beta \rightarrow \gamma$$

se le stringhe si possono decomporre come

$$\beta = \eta A \delta, \gamma = \eta \alpha \delta \text{ con } A \in N; \alpha, \eta, \delta \in (N \cup V)^*$$

ed esiste la produzione $A \rightarrow \alpha \in P$

- Si noti che le stringhe α , η e δ nella definizione precedente possono anche essere vuote

Derivazione

- In modo semplice si può definire una **catena di derivazioni dirette**

$$\beta_0 \rightarrow \beta_1 \rightarrow \beta_2 \dots \rightarrow \beta_n \text{ o anche } \beta_0 \rightarrow^n \beta_n$$

- Definizione. Data una grammatica **G** e due stringhe $\beta, \gamma \in (N \cup V)^*$, si dice che “ **γ deriva da β in **G****” e si scrive

$$\beta \rightarrow^* \gamma$$

se esiste un $n \geq 0$ tale che

$$\beta_0 \rightarrow^n \beta_n \text{ e } \beta_0 = \beta, \beta_n = \gamma$$

Linguaggi generati

□ Definizione.

Data una grammatica G , dicesi **linguaggio generato da G** , e si indica con L_G , l'insieme delle frasi di V^* **derivabili a partire dall'assioma S**

Linguaggio di programmazione

- **Definizione.** Un **linguaggio di programmazione** **L** su un alfabeto **V** è un sottoinsieme di **V*** per cui esiste una grammatica **G**, tale che **L=L_G**, cioè **L** è un **linguaggio generato da G**
- Per **definire un linguaggio di programmazione** c'è bisogno di avere un **alfabeto** e una **grammatica**
- Le stringhe o frasi di un linguaggio di programmazione vengono dette **programmi** (di tale linguaggio)

Backus-Naur-Form - BNF

- Il **formalismo** appena **introdotta** per descrivere la grammatica di un linguaggio di programmazione è un **metalinguaggio formale** che prende il nome di **BNF** (**Backus-Naur-Form**, forma di Backus e Naur, dai nomi dei due studiosi che per primi l'hanno introdotta negli anni '50)
- Un **metalinguaggio** è un linguaggio usato per parlare di un altro linguaggio
 - per esempio, se diciamo "**l'articolo determinativo in inglese è 'the'** ", od anche "**il pronome personale di terza persona singolare è 'he', oppure 'she' oppure 'it'**", stiamo usando l'**italiano** come **metalinguaggio** per descrivere l'**inglese**.

Extended - BNF . . .

- Il **formalismo** BNF viene spesso usato non nella forma originale, ma utilizzando alcune estensioni che permettono una scrittura più concisa delle grammatiche; si parla in questi casi di **EBNF (Extended BNF)**
 - Se nella parte destra di una produzione un simbolo (o sequenza di simboli, o alternativa di simboli) è racchiuso tra **parentesi quadre**, questo significa che esso è opzionale, che cioè può comparire **zero** oppure **una** volta, per esempio

$X \rightarrow [\alpha] \beta$ equivale a $X \rightarrow \beta \mid \alpha\beta$

... Extended - BNF

- Se invece esso è **racchiuso tra parentesi graffe, con un numero intero ad apice**, questo significa zero, una o più occorrenze del simbolo stesso, fino ad un massimo di n ; per esempio

$$X \rightarrow \{\alpha\}^n\beta$$

significa che da X si può derivare:

$$\beta \quad \alpha\beta \quad \alpha\alpha\beta \quad \alpha\alpha\alpha\beta \dots$$

con un massimo di n occorrenze di α

- Se invece un simbolo α è **racchiuso tra parentesi graffe (senza apice)**, questo significa zero, una o più (in **numero finito, ma arbitrario**) occorrenze del simbolo stesso; come in $X \rightarrow \{\alpha\}\beta$

Albero sintattico

- Il **processo di derivazione** di una frase mediante un grammatica può essere convenientemente **illustrato** mediante un albero, detto albero di derivazione sintattica, o più semplicemente **albero sintattico**
- Piuttosto che definire formalmente la nozione di albero sintattico, la introduciamo attraverso due esempi di derivazione per
 - la frase (già vista) '**il gatto mangia il topo**', della lingua italiana
 - i numeri interi senza segno di una o due cifre

Frammento della grammatica italiana

$V = \{ \text{il, lo, gatto, topo, monte, mangia, beve} \}$

$N = \{ \langle \text{frase} \rangle, \langle \text{soggetto} \rangle, \langle \text{verbo} \rangle, \langle \text{complemento} \rangle, \langle \text{articolo} \rangle, \langle \text{nome} \rangle \}$

$S = \langle \text{frase} \rangle$

P consiste di:

$\langle \text{frase} \rangle ::= \langle \text{soggetto} \rangle \langle \text{verbo} \rangle \langle \text{complemento} \rangle$

$\langle \text{soggetto} \rangle ::= \langle \text{articolo} \rangle \langle \text{nome} \rangle$

$\langle \text{articolo} \rangle ::= \text{il} \mid \text{lo}$

$\langle \text{nome} \rangle ::= \text{gatto} \mid \text{topo} \mid \text{monte}$

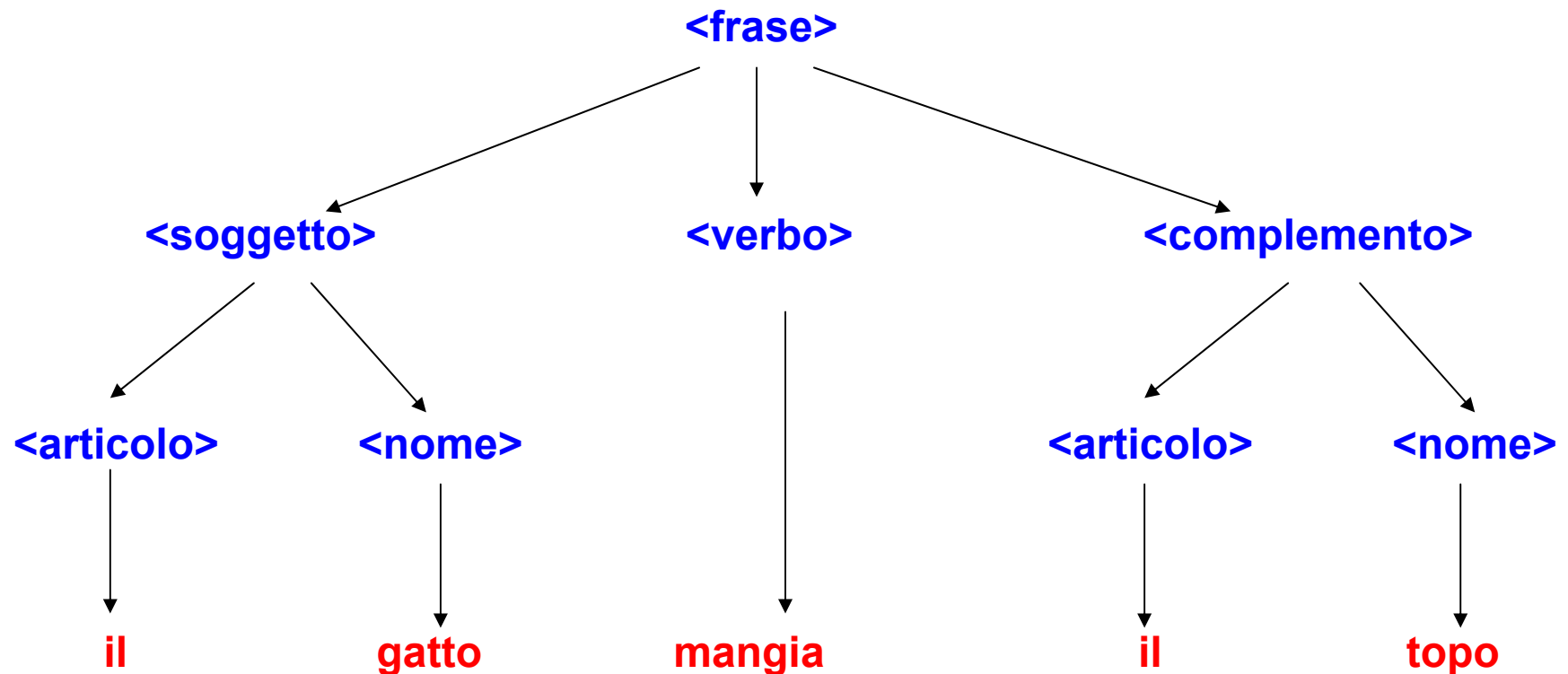
$\langle \text{verbo} \rangle ::= \text{mangia} \mid \text{beve}$

$\langle \text{complemento} \rangle ::= \langle \text{articolo} \rangle \langle \text{nome} \rangle$

N.B. In nero i meta-simboli

Esempio di albero sintattico

□ Deriviamo la frase **'il gatto mangia il topo'**



Questi ultimi sono simboli terminali del linguaggio

Grammatica per interi senza segno di una o due cifre

$V = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$N = \{\langle\text{intero-senza-segno}\rangle, \langle\text{cifra-non-nulla}\rangle, \langle\text{cifra}\rangle\}$

$S = \langle\text{intero-senza-segno}\rangle$

P consiste di:

$\langle\text{intero-senza-segno}\rangle ::=$

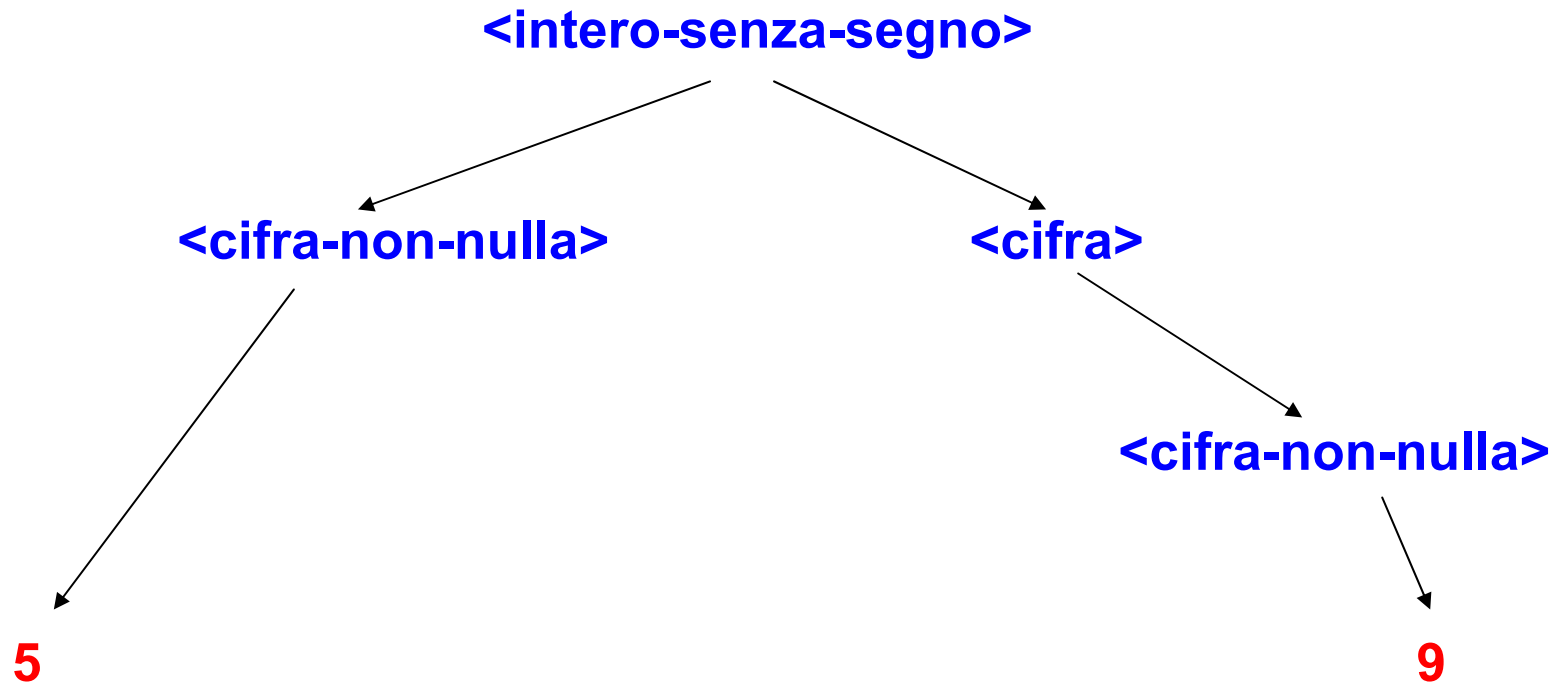
$[\langle\text{cifra-non-nulla}\rangle]\langle\text{cifra}\rangle$

$\langle\text{cifra}\rangle ::= \langle\text{cifra-non-nulla}\rangle \mid 0$

$\langle\text{cifra-non-nulla}\rangle ::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Esempio di albero sintattico

- Deriviamo il numero intero senza segno **59**



Questi ultimi sono simboli terminali del linguaggio

Sintassi dei linguaggi di programmazione

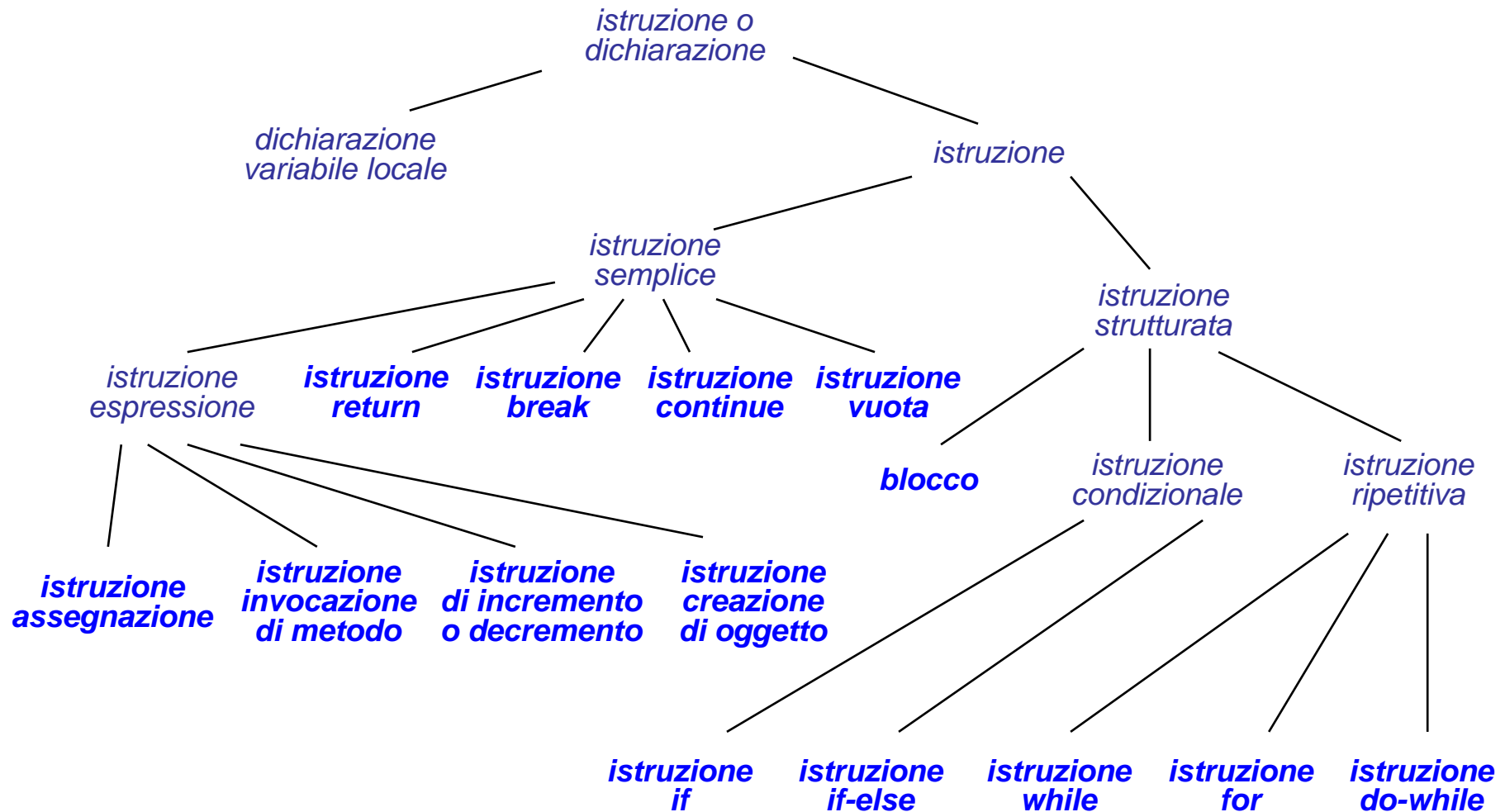
- La definizione della sintassi di un **linguaggio di programmazione** viene data definendo la **grammatica da cui viene generato**
 - il **lessico** (cioè un insieme di **simboli terminali**, che è il **vocabolario**) del linguaggio
 - un insieme di **simboli non terminali**, tra cui ne viene scelto uno come **simbolo iniziale**, cioè l'**assioma**
 - un insieme di **regole di produzione**, in genere espresse in una qualche variante della notazione BNF **grammatica**

Il lessico

□ Il lessico è costituito da

- Un **alfabeto di caratteri e cifre** che servono a costruire identificatori (ad esempio di classi, oggetti, metodi, variabili, . . .)
- Un **insieme di simboli speciali** corrispondenti ad operatori e simboli di interpunzione
- Un **insieme finito di parole chiave**, cioè sequenze di caratteri dell'alfabeto che sono riservate in quanto assumono, a livello semantico, significati particolari nel linguaggio

Una classificazione delle principali istruzioni di Java



Istruzioni strutturate o di controllo

- Ogni istruzione strutturata, pur essendo composta da più istruzioni, viene considerata **sintatticamente una singola istruzione**
 - questo consente la composizione delle istruzioni

- **Semantica delle istruzioni strutturate**
 - è definita con riferimento alle sue componenti
 - istruzioni e condizioni

Introduzione alla programmazione strutturata

- I linguaggi di programmazione, oltre alle istruzioni semplici, definiscono delle **istruzioni strutturate o di controllo**
 - permettono di “controllare” il flusso di esecuzione di altre istruzioni
- **Teorema di Jacopini Böhm**
 - Qualunque **algoritmo** può essere implementato utilizzando tre sole istruzioni:
 - **sequenza**
 - **selezione**
 - **iterazione**

Istruzione di selezione

- **Seleziona una determinata sequenza di istruzioni da eseguire sulla base di una condizione**

se (il pneumatico è sgonfio)

allora

se (è bucato)

allora

sostituiscilo

altrimenti

gonfialo

- **In Java: istruzione **if-else****

Istruzione if-else: sintassi

```
if ( <condizione> )  
    <istruzione-parte-if>  
[ else  
    <istruzione-parte-else> ]
```

Dove

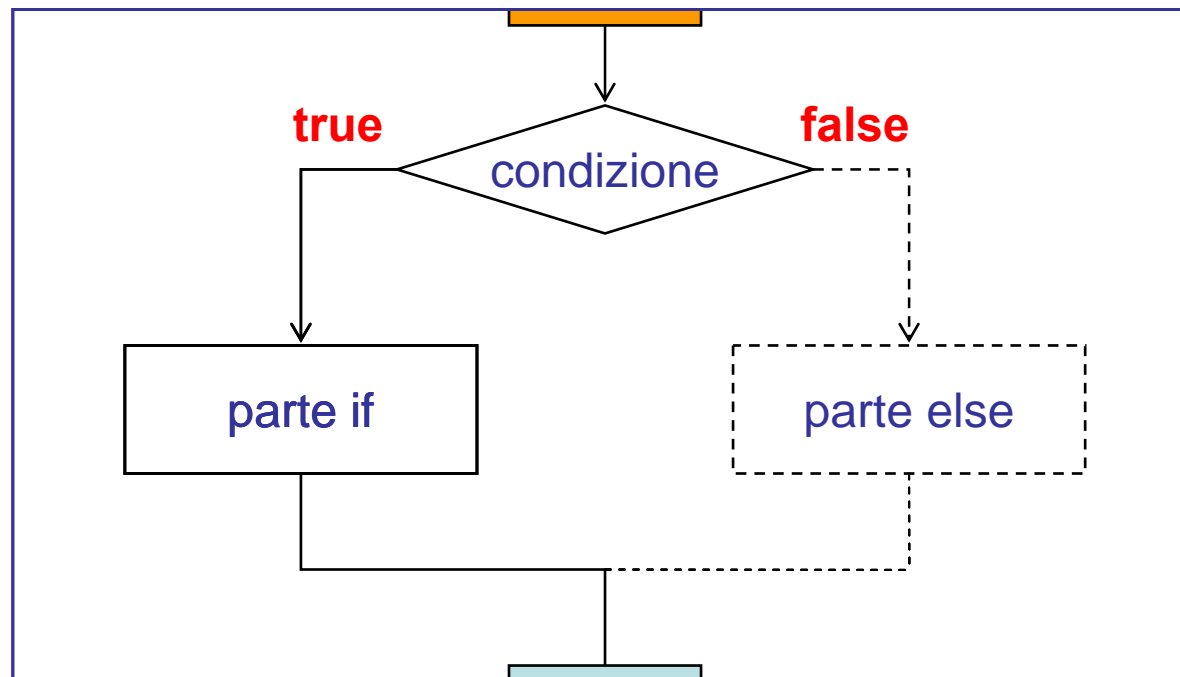
<condizione>	::= <espressione-booleana>
<istruzione-parte-if>	::= <istruzione>
<istruzione-parte-else>	::= <istruzione>
<istruzione>	::= <istruzione-semplice> <istruzione-strutturata>

Istruzione if-else: semantica ...

- **valuta il valore $v_{\text{espressione}}$ della condizione**
 - **il valore $v_{\text{espressione}}$ può essere **true** (la condizione si è verificata, è vera) oppure **false** (la condizione non si è verificata, è falsa)**
- **se il valore $v_{\text{espressione}}$ vale **true**, allora esegui l'istruzione parte-if**
- **se invece il valore $v_{\text{espressione}}$ vale **false**, allora esegui l'istruzione parte-else**

... semantica

- La semantica dell'istruzione if-else può essere descritta graficamente mediante il seguente **diagramma di flusso**



istruzione if-else

Esempio

- *... calcola il minore tra due numeri interi a e b ...*

- `int minore; // il minore tra a e b`

- `if (a < b)`

condizione

- `minore = a;`

istruzione che deve essere eseguita se la condizione è vera

- `else`

- `minore = b;`

istruzione che deve essere eseguita se la condizione è falsa

Sequenza

- Spesso, al verificarsi di una condizione bisogna eseguire *una sequenza di istruzioni* (e non una singola istruzione), altrimenti bisogna eseguire *un'altra sequenza di istruzioni*

... calcola il minore e il maggiore tra a e b ...

```
if (a > b)
    minore = b;
    maggiore = a;
else
    minore = a;
    maggiore = b;
```

*Come algoritmo e' corretto,
ma come porzione di codice Java no!*

Blocco (istruzione composta)

- un blocco è un'istruzione strutturata;**
- è una sequenza di istruzioni (semplici o strutturate) che devono essere eseguite una dopo l'altra.**

Blocco: sintassi

<blocco> ::= { {<istruzione>}* }

**<istruzione> ::= <istruzione-semplce> |
 <istruzione-strutturata>**

Uso di blocchi

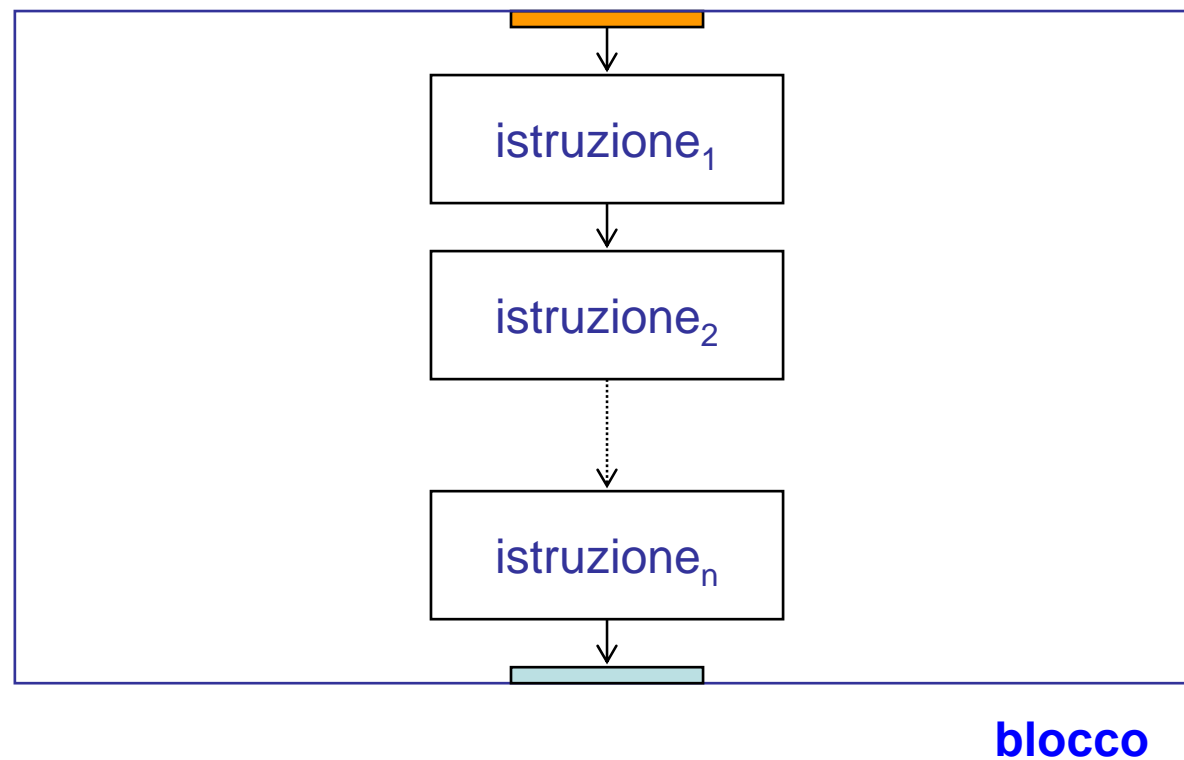
... calcola il minore e il maggiore tra a e b ...

```
if (a > b) {  
    minore = b;  
    maggiore = a; }  
else {  
    minore = a;  
    maggiore = b; }
```

*Ora la porzione di codice implementa
correttamente l'algoritmo*

Blocco: semantica

- valuta le istruzioni che compongono il blocco, una alla volta e in sequenza, nell'ordine in cui sono scritte



Istruzioni ripetitive

- **istruzioni di controllo che permettono di eseguire una istruzione in modo ripetuto**
 - **ciascuna istruzione ripetitiva è composta almeno da**
 - **una istruzione** che deve essere eseguita ripetutamente (il corpo dell'istruzione ripetitiva)
 - **una condizione** che permette di determinare se il corpo dell'istruzione ripetitiva deve essere ancora eseguito

- **Istruzione ripetitiva **while****
- **Istruzione ripetitiva **for****
- **Istruzione ripetitiva **do-while****

Istruzione ripetitiva while

- Calcolare il quoziente **q** della divisione intera tra due numeri interi positivi **a** e **b** dati
 - senza usare l'operatore **/**
 - per sottrazioni ripetute – quante volte posso sottrarre **b** da **a** – devo continuare fintanto che **a** è maggiore di zero ma non è minore di **b**

... calcola il quoziente di a diviso b ...

```
int q;           // il quoziente di a diviso b
/* conta quante volte si puo' sottrarre b da a */
q = 0;
while (a>=b) {
    a = a-b;
    q = q+1;
}
```

... il quoziente è q ...

Istruzione while: sintassi

while (*<espressione>*) condizione

<istruzione> ← corpo

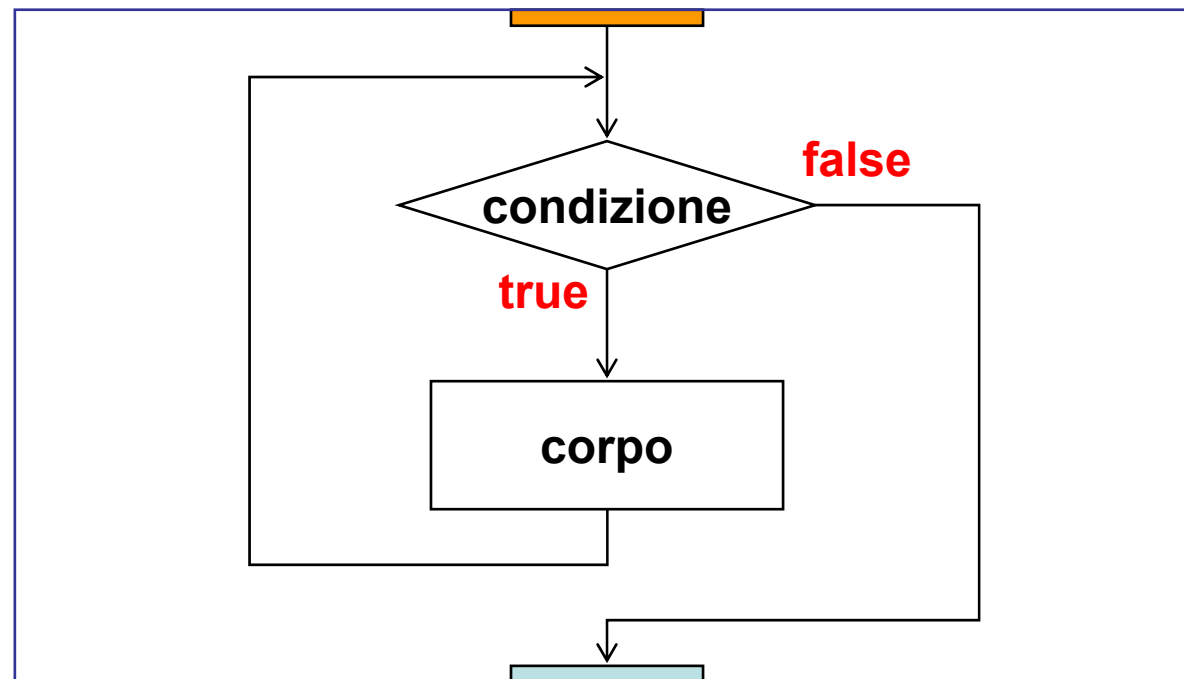
<espressione> ::= *<espressione-booleana>*

<istruzione> ::= *<istruzione-semplice>* |
<istruzione-strutturata>

Istruzione while: semantica

- La semantica dell'istruzione while è la seguente:
- Esegui ripetutamente e in sequenza i seguenti passi
 - valuta la condizione del while
 - se la condizione è vera, esegui il corpo del while
 - se la condizione è falsa, smetti di eseguire questi passi
 - l'esecuzione dell'istruzione while è cioè considerata terminata

... semantica



istruzione while

Osservazioni

□ in una esecuzione di una istruzione while

- il corpo del while può venire eseguito diverse volte
- è possibile che il corpo del while non venga mai eseguito
- l'esecuzione dell'istruzione while potrebbe non terminare

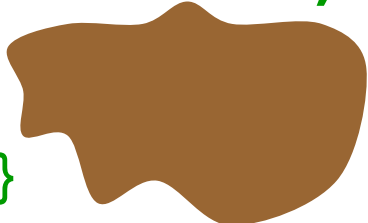
□ se il corpo del while viene eseguito N volte

- la condizione del while viene valutata N+1 volte
- le prime N volte risulta verificata
- l'N+1-esima volta risulta falsa

Esercizio

□ Si consideri il seguente frammento di codice in parte oscurato

- se si assume che l'esecuzione dell'istruzione while termina normalmente, è possibile dire che cosa viene stampato? perché?

```
int n, k;  
  n = 19;  
  k = 3;  
  while (n!=7) {  
    n += k;  
      
  }  
  System.out.println(n);
```

Istruzione ripetitiva for

- Calcola il fattoriale **f** di un numero naturale **n** dato
 - assegna a **f** il valore **1**
 - per ogni valore **i** compreso tra **1** e **n**, moltiplica **f** per **i**

... calcola il fattoriale f di n ...

```
int f;           // il fattoriale di n
int i;          // per iterare tra 1 e n
```

```
/* calcola il fattoriale di n */
```

```
f = 1;
```

```
/* moltiplica f per ogni numero intero i
 * compreso tra 1 e n */
```

```
for (i=1; i<=n; i++)
```

```
    f = f*i;
```

... il fattoriale di n è f ...

Istruzione for: sintassi

for (<inizializzazione> ; < espressione > ; <aggiornamento>)
 <istruzione> ← **corpo**

<inizializzazione> ::= <istruzione-assegnazione> |
 <istruzione-invocazione-metodo>

<espressione> ::= <espressione-booleana>

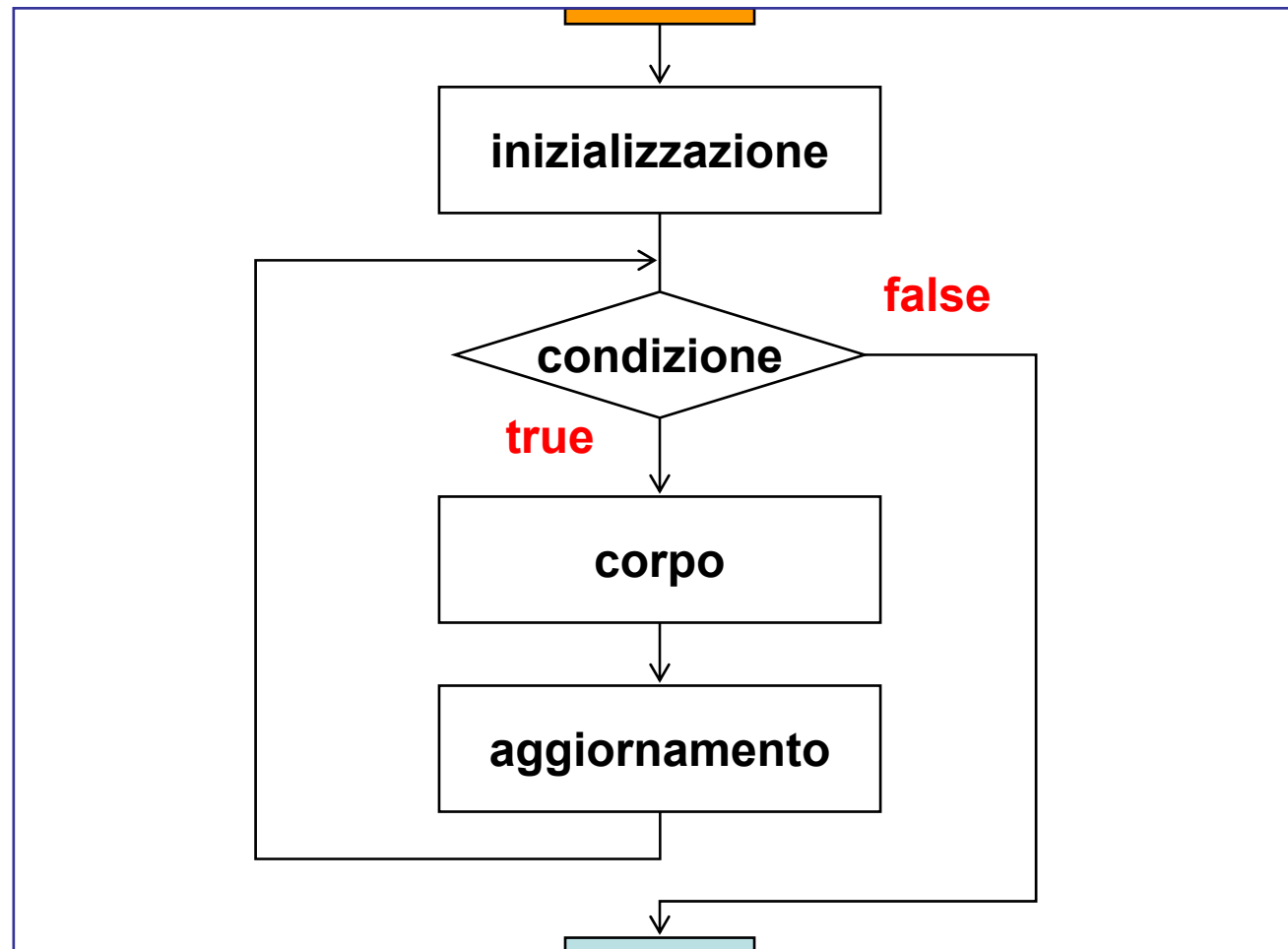
<aggiornamento> ::= <istruzione-incremento> |
 <istruzione-decremento> |
 <istruzione-invocazione-metodo>

<istruzione> ::= <istruzione-semplice> |
 <istruzione-strutturata>

Istruzione for: semantica

- **Esegui l'inizializzazione del for**
- **Esegui ripetutamente e in sequenza i seguenti passi**
 - **valuta la condizione del for**
 - **se la condizione è vera**
 - esegui il corpo del for
 - esegui l'aggiornamento del for
 - **se invece la condizione è falsa, smetti di eseguire questi passi**
 - ovvero, l'esecuzione dell'istruzione for è terminata

... semantica



Semantica istruzione for

Osservazioni

□ in una esecuzione di una istruzione for

- l'**inizializzazione** viene eseguita esattamente **una volta**
- ciascuna esecuzione del corpo è preceduta da una valutazione positiva della condizione e seguita da una esecuzione dell'aggiornamento
- è possibile che il corpo del for non venga **mai eseguito**, venga eseguito uno o più volte, venga **eseguito all'infinito**

□ se il corpo del for viene eseguito N volte

- l'**inizializzazione** viene eseguita esattamente **una volta**
- la **condizione** del for viene valutata **N+1 volte**
- l'**aggiornamento** viene eseguito **N volte**

Uso tipico del for

□ L'istruzione **for** si usa quando un'istruzione deve essere **iterata un fissato numero di volte**

□ Ad esempio

```
for (i=1; i<=n; i++)  
    f = f*i;
```

Istruzione for con variabile contatore

□ Il significato di una istruzione for con variabile contatore è facilmente comprensibile

- dalla lettura dell'intestazione del for

```
for (i=0; i<n; i++)  
    ... corpo del for ...
```

```
for (i=1; i<=n; i++)  
    ... corpo del for ...
```

```
for (i=n; i>0; i--)  
    ... corpo del for ...
```

```
for (i=n-1; i>=0; i--)  
    ... corpo del for ...
```

Istruzione ripetitiva do-while

□ Nella scansione di una sequenza di caratteri letti dalla tastiera

- bisogna leggere e ignorare gli spazi bianchi iniziali
- leggere anche il primo carattere non spazio
- ad esempio – indicando gli spazi come #
 - se la sequenza è ##abc
 - il primo carattere non spazio è la a

... legge una sequenza di caratteri fino al primo carattere non spazio ...

```
char car;    // un carattere letto dalla tastiera
/* leggi e ignora gli spazi bianchi iniziali */
do {
    car = Lettore.in.leggiChar();
} while (car==' ');
```

... il primo carattere non spazio è car ...

Istruzione do-while: sintassi

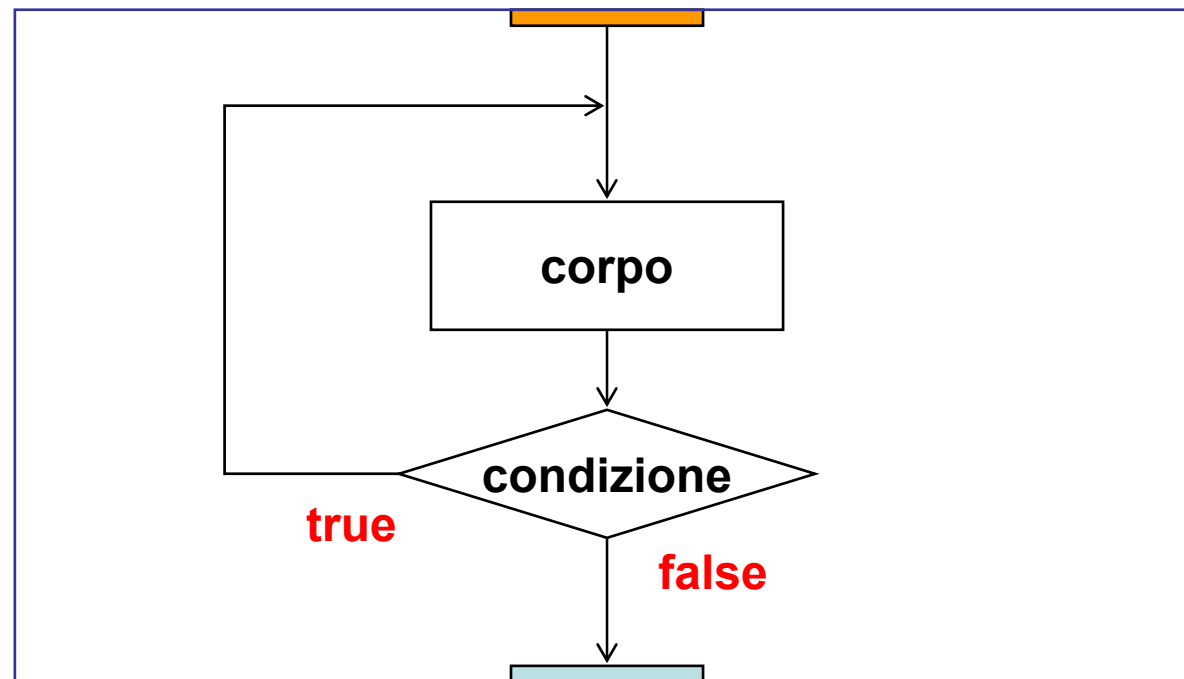
do <istruzione> ← corpo
while (<espressione>) ← condizione

<espressione> ::= <espressione-booleana>
<istruzione> ::= <istruzione-semplice> |
 <istruzione-strutturata>

Istruzione do-while: semantica

- **Esegui ripetutamente e in sequenza i seguenti passi**
 - **esegui il corpo del do-while**
 - **valuta la condizione del do-while**
 - **se la condizione è vera, continua a eseguire questa sequenza di passi**
 - **se invece la condizione è falsa, smetti di eseguire questi passi**
 - **ovvero, l'esecuzione dell'istruzione do-while è terminata**

... semantica



istruzione do-while

Osservazioni

□ Osservazioni

- **in una esecuzione di una istruzione do-while**
 - il corpo del do-while viene eseguito almeno una volta
- **se il corpo del do-while viene eseguito N volte**
 - la condizione del do-while viene valutata N volte
- **è possibile che la condizione del do-while non possa venire valutata se il corpo non è mai stato eseguito**

□ Che succede se la sequenza in ingresso è

- `##abc` ?
- `abc` ?

Confronto tra istruzioni ripetitive

□ Le istruzioni **for** e **while** sono semanticamente equivalenti

- Una istruzione **for** può essere espressa in termini di una **while** e viceversa

```
f = 1;
for (i=1; i<=n; i++)
    f = f*i;
```

- diventa

```
f = 1;
i = 1;
while ( i<=n ) {
    f = f*i;
    i++; }
```

Uso dell'istruzione ripetitiva for

- L'istruzione **for** viene in genere utilizzata per eseguire ripetutamente una istruzione mentre una variabile assume valori in una sequenza prefissata
 - in cui i valori della sequenza sono noti a priori (ovvero, già immediatamente prima di iniziare a eseguire l'istruzione ripetitiva)

Uso dell'istruzione ripetitiva while

- L'istruzione **while** viene solitamente utilizzata per eseguire ripetutamente una istruzione **un numero imprecisato di volte** (ma non necessariamente almeno una volta)
 - per imprecisato si intende imprecisato a priori, ovvero non determinabile in modo elementare già immediatamente prima di iniziare a eseguire l'istruzione ripetitiva

Uso dell'istruzione ripetitiva do-while

- L'istruzione **do-while** viene solitamente utilizzata per eseguire ripetutamente una istruzione **almeno una volta e un numero imprecisato di volte**