Gocce di Java

Selezione e ripetizione

Pierluigi Crescenzi

Università degli Studi di Firenze

- **Variabile booleana**: può assumere valore `true` o `false`
- **Espressione Booleana**: valore di ritorno `true` o `false`
  - Più semplici: confronto di due valori

Notazione matematica	**Java**	**Esempio**
$=$ (uguale a)	`==`	`riga1==0`
$\neq$ (diverso da)	`!=`	`riga1!=riga2`
$>$ (maggiore di)	`>`	`riga1>riga2`
$\geq$ (maggiore oppure uguale a)	`>=`	`riga1>=4`
$<$ (minore di)	`<`	`riga1<riga2`
$\leq$ (minore oppure uguale a)	`<=`	`riga1<=4`

Cattive abitudini

```
(delta = b*b-4*a*c)>0
++contatore<limite
```

- Consentito da Java
- Esempio di cattiva programmazione
  - Mescola verifica con altra attività
    - Effetto collaterale

Confronto di numeri reali

- Inaffidabile confrontare esattamente due numeri reali x e y
  - Limitarsi a verificare se siano sufficientemente vicini
    - x o y uguale a 0: verificare se valore assoluto di altro numero minore di un piccolo valore di soglia
    - x e y diversi da 0: verificare se $\frac{|x-y|}{\max(x,y)}$ minore di valore di soglia

Confronto di array

```
int[] a1 = {0,0,0};
int[] a2 = {0,0,0};
boolean arrayUguali = (a1==a2);
System.out.println( "a1=a2: "+arrayUguali );
```

- a1 e a2 memorizzati in diverse locazioni di memoria
- Confronto tra coppie di variabili di tipo array diverso da confronto di elementi dei due array

- Combinano più espressioni Booleane in modo da formare espressioni Booleane più complesse
- Sintassi

  `(`*espressione_Booleana_1*`)&&(`*espressione_Booleana_2*`)`

  `(`*espressione_Booleana_1*`)||(`*espressione_Booleana_2*`)`

  `!(`*espressione_Booleana*`)`
- Esempio

```
(valore>minimo)&&(valore<massimo)
(valore<minimo)||(valore>massimo)
!(valore>0)
```

- **congiunzione** (`&&`)
- **disgiunzione** (`||`)
- **negazione** (`!`)

e1	e2	e1&&e2	e1||e2	!e1
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

- Esiste anche disgiunzione esclusiva (`^`)
  - `true` se e solo se `e1` e `e2` hanno valori diversi

Operatori logici di disgiunzione e di congiunzione

- Configurazione scacchiera

7		*		*		*		*
6	*		*		*		*	
5		*		*		*		*
4	*		*		*		*	
3	n	*		*		*	n	*
2	*	b	*	b	*		*	b
1		*		*		*		*
0	*		*		*		*	
	0	1	2	3	4	5	6	7

- Decidere se pedina bianca in riga `r` e colonna `c` può muoversi

```
(((c-1)>=0)&&(casella[r+1][c-1]==' '))||
(((c+1)<=7)&&(casella[r+1][c+1]==' '))
```

- (2,1) e (2,2) danno `true`, mentre (2,7) dà `false`

Due osservazioni

- Dipendenza tra operatori logici: leggi di De Morgan
  - `a||b` equivale a `!(!a&&!b)`
  - `a&&b` equivale a `!(!a||!b)`
- Valutazione "pigra"
  - `&&` non valuta seconda espressione se prima falsa
  - `||` non valuta seconda espressione se prima vera
  - Esempio

    ```
    ((ng!=0)&&((premio/ng)>10))
    ```

  - Operatori `&` e `|`: stesso significato logico ma non pigri

Operatori	Associatività
!	destra verso sinistra
(*tipo*)	destra verso sinistra
* / %	sinistra verso destra
+ -	sinistra verso destra
< <= >= >	sinistra verso destra
== !=	sinistra verso destra
&	sinistra verso destra
|	sinistra verso destra
&&	sinistra verso destra
||	sinistra verso destra
= += -= *= /= %= &= |=	destra verso sinistra

- Esempio
  - `a-1>=0&&b==' '||c+1<=7&&d==' '`
  - `(a-1)>=0&&b==' '||(c+1)<=7&&d==' '`
  - `((a-1)>=0)&&b==' '||((c+1)<=7)&&d==' '`
  - `((a-1)>=0)&&(b==' ')||((c+1)<=7)&&(d==' ')`
  - `(((a-1)>=0)&&(b==' '))||(((c+1)<=7)&&(d==' '))`
  - `((((a-1)>=0)&&(b==' '))||(((c+1)<=7)&&(d==' ')))`
- Inserire parentesi per rendere espressioni aritmetiche e Booleane più leggibili
  - Ma non esagerare

- Gruppo di istruzioni racchiuse tra parentesi graffe
- Esempi

```
{
   soluzione = -b/a;
   System.out.println( "Soluzione: "+soluzione );
}
```

```
{
   casella[2][1] = ' ';
   casella[3][0] = 'b';
}
```

  - Istruzioni indentate rispetto a parentesi
  - Buona norma: usare blocchi anche per una sola istruzione

- Assegnano nome a blocco di istruzioni
- Dichiarazione
  - nome
  - dati passati al metodo dall'esterno e utilizzati al suo interno
  - tipo di dato dell'eventuale valore di ritorno
- Definizione: blocco di istruzioni da eseguire
- Sintassi

  `void` *`nome_Metodo`*`(` *`lista_Parametri`* `)` *`blocco`*

  `tipo_Output` *`nome_Metodo`*`(` *`lista_Parametri`* `)` *`blocco`*
- Esempio

```
void stampaNumero( int n ) {
  System.out.println( "Il numero e': "+n );
}
int sommaNumeri( int n ) {
  return n*(n+1)/2;
}
```

- Due tipi di metodi
  - Quelli che restituiscono singolo valore
    - Metodo che restituisce la somma dei primi n numeri interi
  - Quelli che eseguono un'azione senza restituire valore
    - Metodo che stampa messaggio di benvenuto su schermo
- In ogni caso, dichiarazione include tipo di ritorno
  - Uno dei tipi di dato primitivi, un tipo array, o un tipo classe
  - `void`: se metodo non restituisce valore
- Istruzione di ritorno: parola chiave `return` seguita da espressione
  - Obbligatoria per metodi non `void`
  - Espressione produce valore del tipo specificato in dichiarazione
  - Esempio

```
int sommaInteri(int n) {
  int risultato = n*(n+1)/2;
  return risultato;
}
```

- Invocazione: nome metodo seguito da elenco dati su cui operare (separati da virgole)
  - Esempio

    ```
    sommaInteri( 6 )
    ```

- Metodo invocato: istruzioni in corpo eseguite
  - Esecuzione `return` termina invocazione: valore espressione è valore di ritorno invocazione
- Metodo invocabile ovunque sia lecito usare valore di tipo di ritorno
  - Esempio

    ```
    int somma = sommaInteri( 6 );
    ```

    equivalente a

    ```
    int somma = 21;
    ```

- Metodi `void`: analogo a metodi non `void`
  - Istruzione `return` non obbligatoria
    - Esempio

```
void stampaBenvenuto(  ) {
  System.out.println( "Buongiorno!" );
  System.out.println( "E benvenuti!" );
}
```

  - Metodo invocabile ovunque
    - Esempio

```
stampaBenvenuto(  );
```

stampa

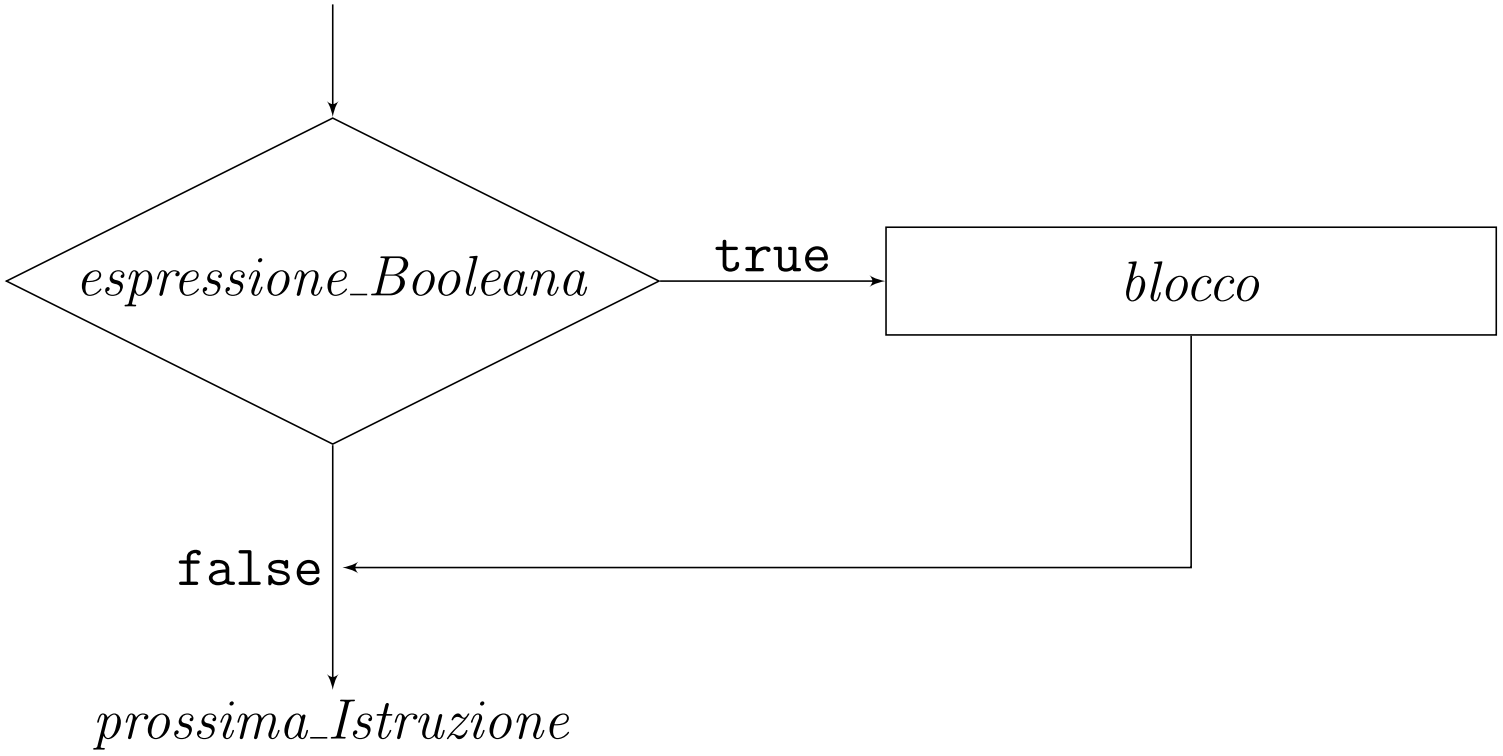
```
Buongiorno!
Benvenuti nel programma!
```

- Nomi metodi: stesse regole di nomi variabili

- Realizza selezione semplice: sceglie o ignora determinata azione
- Sintassi

  `if (`*espressione_Booleana*`)`
  blocco
- Esempio

```
if (delta==0) {
   sol = -b/(2*a);
   System.out.println( "Una soluzione: "+sol );
}
```


espressione_Booleana
true
blocco
false
prossima_Istruzione

Istruzione if e numeri pari

- Metodo per determinare se un dato numero sia pari e comunicare risultato all'esterno

```
void decidiPari(int n) {
  if ((n%2==0)) {
    System.out.println( "Numero "+n+" e' pari." );
  }
}
```

Istruzione if e il gioco della morra cinese

- Codifica dei segni
  - Sasso: 0
  - Forbici: 1
  - Carta: 2
- Metodo per determinare vincitore di due segni

```
int vincitore(int segno1, int segno2) {
  if (segno2==(segno1+1)%3) {
    return 1;
  }
  if (segno1==(segno2+1)%3) {
    return 2;
  }
  return 0;
}
```

Istruzione if ed equazioni di primo grado

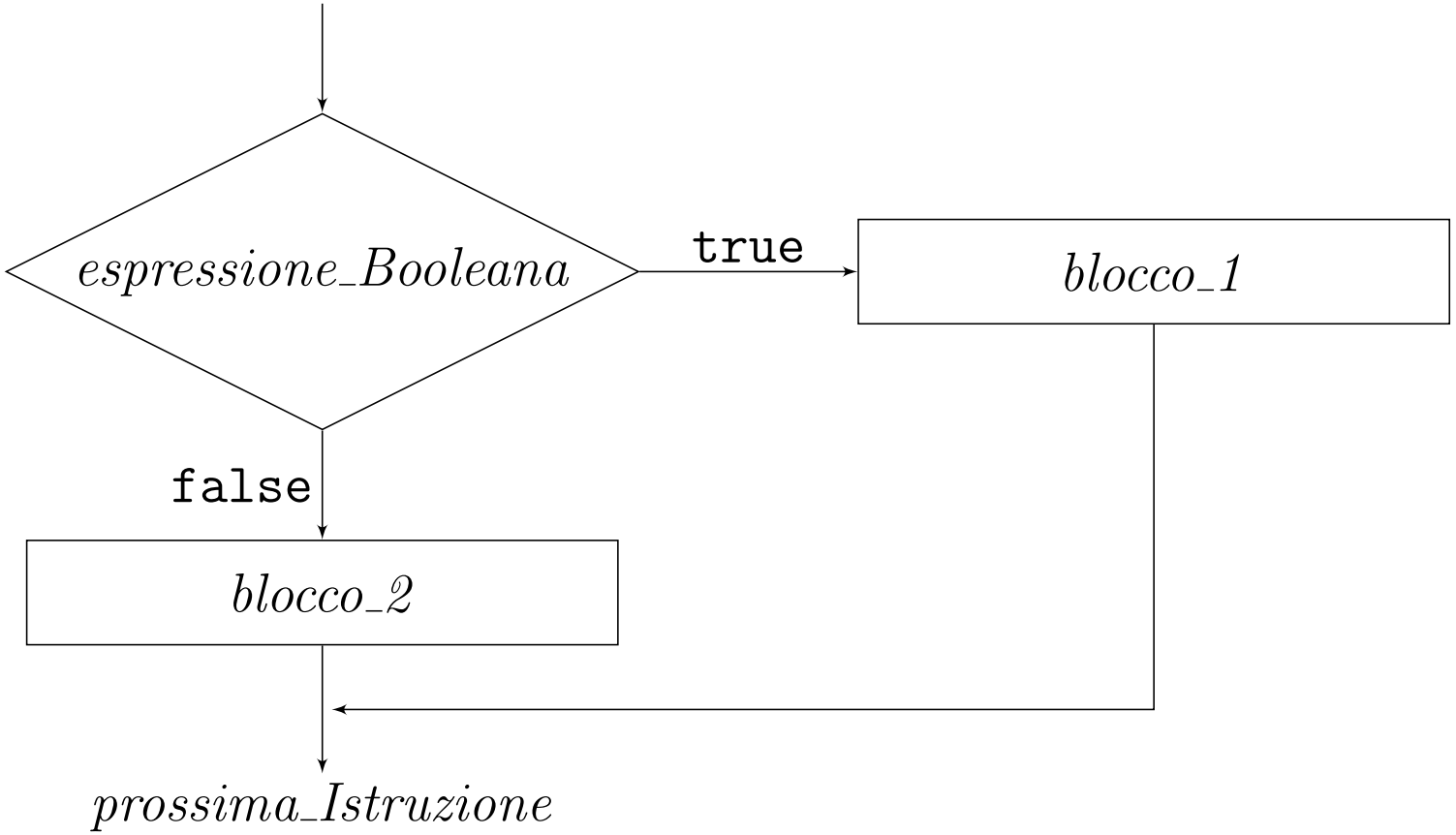
- Metodo per comunicare soluzione di un'equazione di primo grado con coefficienti a e b

```
void risolviEquazioneIGrado(double a, double b) {
  if (a==0) {
    if (b==0) {
      System.out.println( "Indeterminata" );
    }
    if (b!=0) {
      System.out.println( "Impossibile" );
    }
  }
  if (a!=0) {
    System.out.println( "Soluzione: "+(-b/a) );
  }
}
```

- Realizza selezione doppia: sceglie tra due possibili alternative
- Sintassi

  `if (`*`espressione_Booleana`*`)`
  `blocco_1` `else`
  `blocco_2`
- Esempio

```
if (primoNumero>secondoNumero) {
   minimo = secondoNumero;
} else {
   minimo = primoNumero;
}
```


espressione_Booleana
true
blocco_1
false
blocco_2
prossima_Istruzione

Istruzione if-else e numeri pari e dispari

- Metodo per comunicare all'esterno se un dato numero sia pari o dispari

```
void decidiPariDispari(int n) {
  if ((n%2==0)) {
    System.out.println( "Numero "+n+" e' pari." );
  } else {
    System.out.println( "Numero "+n+" e' dispari." );
  }
}
```

Istruzione if-else e il gioco dei fiammiferi

- Sul tavolo n fiammiferi
  - Assumiamo $n > 1$
- Ogni giocatore ne può prendere al massimo k
  - Assumiamo n diverso da multiplo di $(k+1)$ più uno
- Metodo per decidere prima mossa
  - Se $n\%(k+1) = 0$: raccoglie k fiammiferi
  - Se $n\%(k+1) \neq 0$: altrimenti ne raccoglie $n\%(k+1) - 1$

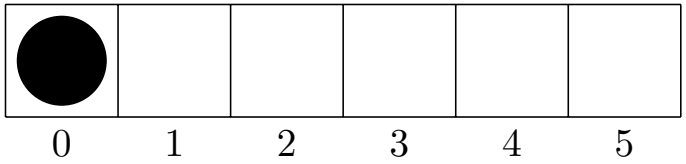
```
int primaMossa(int n, int k) {
  int x = n%(k+1);
  if (x==0) {
    return k;
  } else {
    return x-1;
  }
}
```

Istruzione if-else e il gioco dei fiammiferi

- Metodo per decidere mosse successive primo giocatore
  - b: fiammiferi raccolti dall'avversario ($1 \leq b \leq k$)
  - Raccoglie $(k + 1) - b$ fiammiferi

```
int mossaSuccessiva(int b, int k) {
  return (k+1)-b;
}
```

Istruzione if-else e il gioco della corsa della pedina



- A turno, giocatori avanzano verso destra pedina di 1 o 2 posizioni
- Metodo per decidere mosse primo giocatore
  - Prima mossa: 2
  - Mosse successive: $5 - p$ con p posizione attuale

```
int mossa(int p) {
  if (p==0) {
    return 2;
  } else {
    return 5-p;
  }
}
```

- Istruzioni if-else possono essere annidate
- Metodo per la soluzione equazione primo grado (rivisto)

```
void risolviEquazioneIGrado(double a, double b) {
  if (a==0) {
    if (b==0) {
      System.out.println( "Indeterminata" );
    } else {
      System.out.println( "Impossibile" );
    }
  } else {
    System.out.println( "Soluzione: "+(-b/a) );
  }
}
```

Istruzione if-else ed equazioni di secondo grado

- Metodo per soluzione equazione II grado

```
void risolviIIGrado(double a, double b, double c) {
  double delta = b*b-4*a*c;
  if (delta<0) {
    System.out.println( "Soluzioni coniugate" );
  } else {
    if (delta>0) {
      System.out.println( "Soluzioni distinte" );
    } else {
      System.out.println( "Soluzioni coincidenti" );
    }
  }
}
```

Istruzione if-else e parentesi graffe

- Con parentesi

```
if (a==0) {
  if (b>0) {
    System.out.println( "Impossibile" );
  }
} else {
  System.out.println( "Almeno una soluzione" );
}
```

- Senza parentesi

```
if (a==0)
  if (b>0)
    System.out.println( "Impossibile" );
else
  System.out.println( "Almeno una soluzione" );
```

- Senza parentesi: `else` accoppiato a `if` più vicino

Diramazioni multiple

- Diramazione doppia sufficiente per diramazioni multiple
- Metodo per convertire voti in lettere

```
char convertiVoto(int voto) {
  char letteraVoto;
  if (voto>27) {
    letteraVoto = 'A';
  } else if (voto>24) {
    letteraVoto = 'B';
  } else if (voto>21) {
    letteraVoto = 'C';
  } else if (voto>17) {
    letteraVoto = 'D';
  } else {
    letteraVoto = 'E';
  }
  return letteraVoto;
}
```

- Indentazione diversa da standard ma più leggibile

L'operatore condizionale

- Espressione Booleana seguita da ? e due espressioni separate da :
- Esempio

```
max = (primo>secondo)?primo:secondo;
```

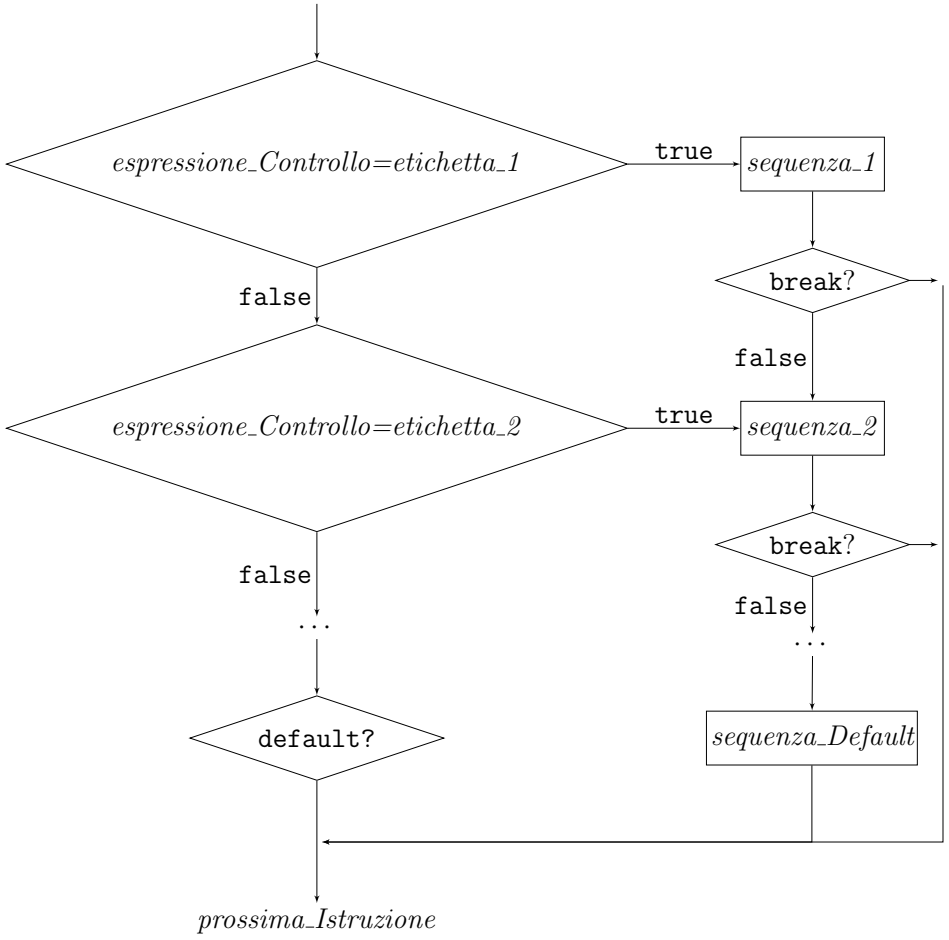
equivalente a

```
if (primo>secondo) {
  max = primo;
} else {
  max = secondo;
}
```

- Realizza selezione mutipla (in casi particolari)
- Sintassi

```
switch (espressione_Controllo)
{
  case etichetta_Caso_1:
    sequenza_Istruzioni_1
    break; //opzionale
  ...
  case etichetta_Caso_n:
    sequenza_Istruzioni_n
    break; //opzionale
  default:  //opzionale
    sequenza_Istruzioni_Default
}
```

- Espressione controllo di tipo `char`, `int`, `short`, `byte`


espressione_Controllo=etichetta_1
true
sequenza_1
break?
false
false
espressione_Controllo=etichetta_2
true
sequenza_2
break?
false
false
...
...
default?
sequenza_Default
prossima_Istruzione

Istruzione switch e il gioco della pedina

- Metodo per mosse gioco della corsa della pedina

```
int mossa(int p) {
  switch (p) {
    case 0: return 2;
    case 1: return 1;
    case 2: return 1;
    case 3: return 2;
    case 4: return 1;
    default: return -1;
  }
}
```

- `break` salta a istruzione successiva a istruzione `switch`
  - Se non presente, continua con istruzioni caso successivo
- Metodo per mosse gioco della corsa della pedina (rivisto)

```
switch (p) {
  case 0:
  case 3: return 2;
  case 1:
  case 2:
  case 4: return 1;
  default: return -1;
}
```

- Metodo per conversione voto in lettere (rivisto)

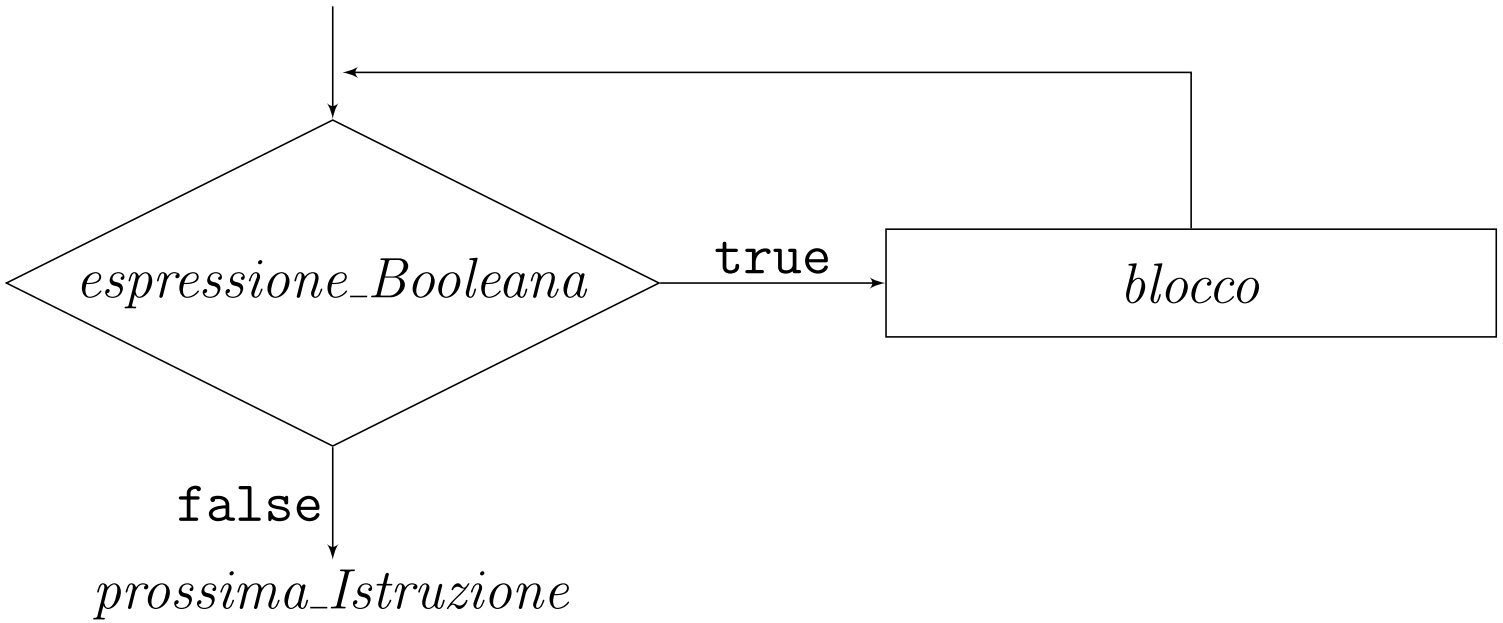
```
char convertiVoto(int voto) {
  char letteraVoto;
  switch (voto) {
    case 30:
    case 29:
    case 28: letteraVoto = 'A';
      break;
    case 27:
    case 26:
    case 25: letteraVoto = 'B';
      break;
    case 24:
    case 23:
    case 22: letteraVoto = 'C';
      break;
    case 21:
    case 20:
    case 19:
    case 18: letteraVoto = 'D';
      break;
    default: letteraVoto = 'E';
  }
  return letteraVoto;
}
```

- **Ciclo**: porzione di programma che ripete blocco istruzioni
  - Istruzioni da ripetere: **corpo** del ciclo
  - Ogni ripetizione: **iterazione** del ciclo
  - Numero di iterazioni dipende da
    - Verificarsi determinata condizione: cicli **controllati da condizioni**
    - Da numero di valori assunti da variabile: cicli **controllati da contatori**

- Realizza cicli controllati da condizioni
- Sintassi

  `while (`*`espressione_Booleana`*`)`
  `blocco`
- Esempio

```
while (numero<=limiteSuperiore) {
    somma = somma+numero;
    numero = numero+1;
}
```


espressione_Booleana
true
blocco
false
prossima_Istruzione

Istruzione while e cifre decimali

- Metodo per calcolare numero cifre decimali di numero intero
  - Incrementa contatore fintantoché divisione numero per 10 restituisce valore maggiore di 0

```
int numeroCifre(int numero) {
  int numeroCifre = 1;
  while (numero/10>0) {
    numeroCifre = numeroCifre+1;
    numero = numero/10;
  }
  return  numeroCifre;
}
```

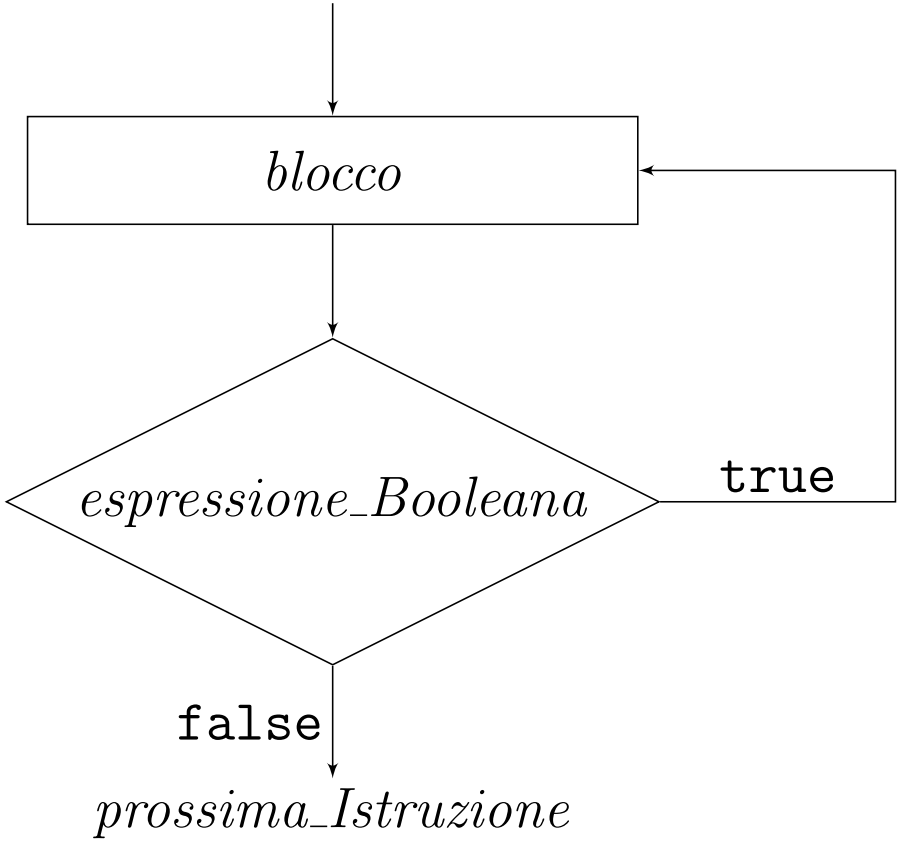
- Corpo di ciclo `while` può essere eseguito zero volte
- Metodo per calcolo di MCD (algoritmo di Euclide)

```
int mcd(int a, int b) {
  while (a>0 && b>0) {
    if (a<b) {
      b = b%a;
    } else {
      a = a%b;
    }
  }
  if (a==0) {
    return b;
  } else {
    return a;
  }
}
```

- Realizza cicli controllati da condizioni
- Sintassi

  do
  blocco `while (`*espressione_Booleana*`);`
- Esempio

```
do {
     somma = somma+numero;
     numero = numero+1;
} while (numero<=limiteSuperiore);
```


blocco
espressione_Booleana
true
false
prossima_Istruzione

Istruzione do-while e media di numeri positivi

- Metodo per calcolare media numeri positivi inseriti da utenti

```
void media() {
  int somma = 0;
  int numeroNumeri = 0;
  double media;
  int numero;
  do {
    numero = Input.getInt( "Intero (<=0 per finire)" );
    if (numero>0) {
      somma += numero;
      numeroNumeri++;
    }
  } while (numero>0);
  if (numeroNumeri==0) {
    System.out.println( "Nessun numero introdotto" );
  } else {
    media = (double)somma/numeroNumeri;
    System.out.println( "Media: "+media );
  }
}
```

Istruzione do-while e numero di Nepero

- Una delle costanti più importanti della matematica
  - Deposito di un milione di euro in banca con interesse percentuale annuo pari a x
    - Dopo un anno: capitale pari a $1 + x$ milioni di euro
  - Interesse semestrale
    - Dopo un anno: capitale pari a $1 + x/2 + (1 + x/2)x/2 = (1 + x/2)^2$
  - Interesse ogni periodo (periodo pari ad anno diviso n)
    - Dopo un anno: capitale pari a $(1 + x/n)^n$
  - $(1 + x/n)^n$ tende a e^x al crescere di n
- Numero e calcolabile mediante equivalenza
  $e = 1 + \frac{1}{1} + \frac{1}{1\cdot 2} + \frac{1}{1\cdot 2\cdot 3} + \frac{1}{1\cdot 2\cdot 3\cdot 4} + \cdots$
  - Somma primi $n + 1$ termini: valore che approssima e entro fattore additivo pari a $\frac{1}{1\cdot 2\cdot 3\cdots n\cdot (n+1)} \left(1 + \frac{2}{n+2}\right)$

Istruzione do-while e numero di Nepero

- Metodo per calcolare e con precisione ϵ

```
double numeroNepero(double epsilon) {
  int n = 0;
  double termine = 1;
  double nepero = 0;
  do {
    nepero = nepero+termine;
    n = n+1;
    termine = termine/n;
  } while ((termine*(1+2.0/(n+2)))>epsilon);
  return nepero;
}
```

Cicli infiniti

- Variabili coinvolte in espressione di controllo non modificate da corpo
- Variabili modificate ma espressione di controllo sempre vera

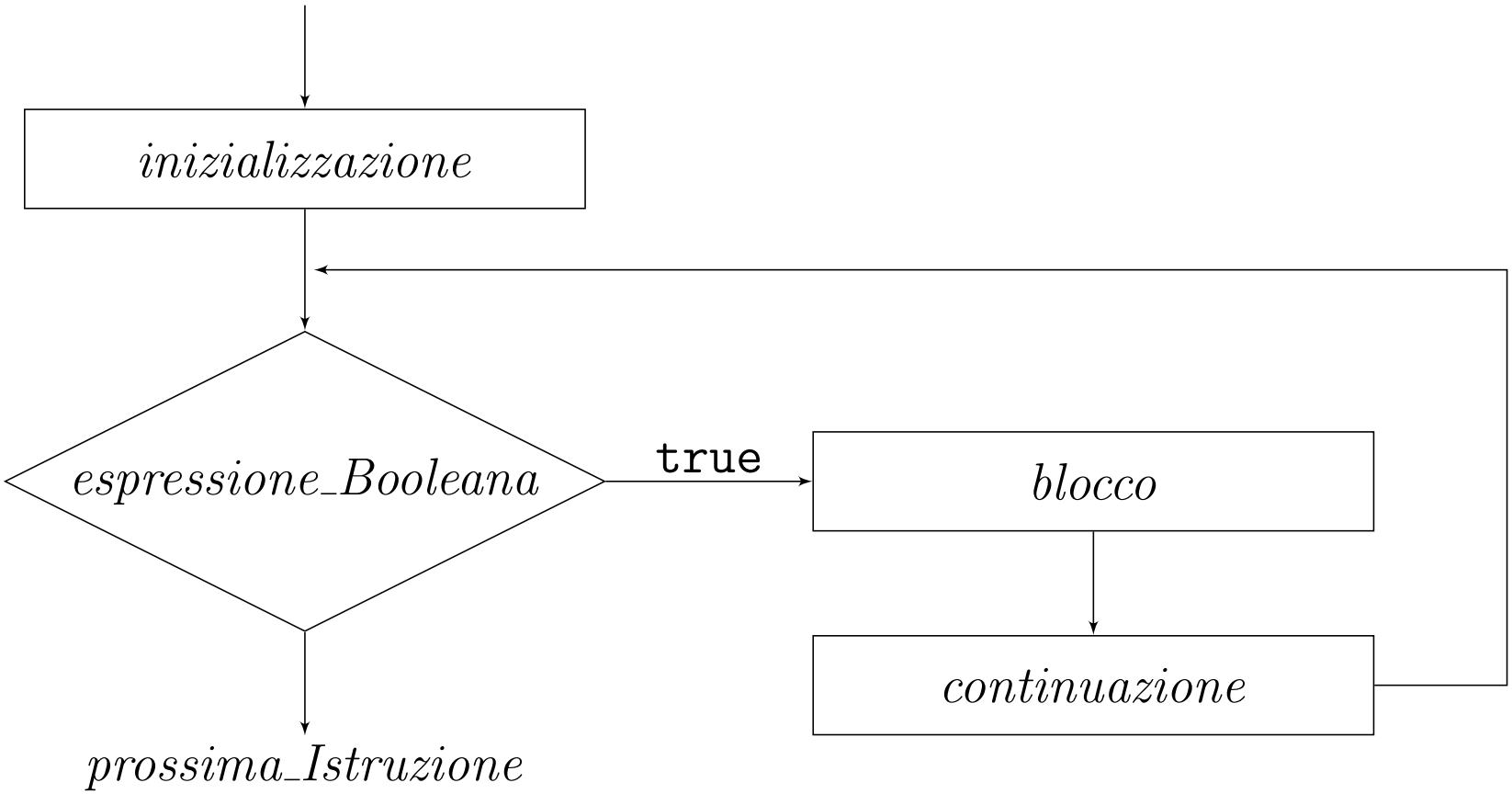
```
int sommaDispari(int n) {
  int numero = 1;
  int somma = 0;
  do {
    somma = somma+numero;
    numero = numero+2;
  } while (numero!=n);
  return somma;
}
```

- Altri errori nei cicli: mancanza di cura in progettazione espressione di controllo
  - Esempio: usare < invece di <= oppure usare == o != con numeri reali

- Realizza cicli controllati da contatori
- Sintassi

  `for (`*`inizializzazione`*`; `*`espressione_Booleana`*`;`
  `continuazione``)`
  `blocco`
- Esempio

```
for (int n = 1; n<=10; n = n+1) {
     somma = somma+n;
     prodotto = prodotto*n;
}
```


inizializzazione
espressione_Booleana
true
blocco
continuazione
prossima_Istruzione

Istruzione for e fattoriale

- Fattoriale $n!$ di numero intero positivo n: prodotto primi n numeri interi positivi
  - Esempio: $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$
- Metodo per calcolare fattoriale di n

```
long fattoriale(int n) {
  long risultato = 1;
  for (int i = 2; i<=n; i = i+1) {
    risultato = risultato*i;
  }
  return risultato;
}
```

Istruzione for e inizializzazione di array

- Metodo per inizializzare array di posizioni giocatori di gioco dell'oca
  - Numero di giocatori non noto a priori

```
int[] posizioni() {
  int ng = Input.getInt( "Numero giocatori" );
  int[] posizioneGiocatore = new int[ng];
  for (int i = 0; i<ng; i = i+1) {
    posizioneGiocatore[i] = 1;
  }
  return posizioneGiocatore;
}
```

- Valore di ritorno: tipo array

Istruzione for e ricerca in array

- Metodo per cercare massimo in array di interi

```
int massimo(int[] numero) {
  int massimo = numero[0];
  for (int i = 1; i<numero.length; i = i+1) {
    if (massimo<numero[i]) {
      massimo = numero[i];
    }
  }
  return massimo;
}
```

- Parametro: tipo array

Istruzione for e valutazione dei polinomi

- Polinomio: $p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n$
- Calcolo di polinomio in valore: calcolare ciascun termine e sommare termini
  - n operazioni di elevamento a potenza
- Algoritmo di Horner
  - Basato su equivalenza
    $p(x) = ((\cdots(((a_0x + a_1)x + a_2)x + a_3)x + \cdots + a_{n-1})x + a_n)$

```
double valorePolinomio(double[] p, double x) {
  double valore = p[0];
  for (int i = 1; i<p.length; i++) {
    valore = valore*x+p[i];
  }
  return valore;
}
```

Istruzione for e stampa di matrici

- Metodo per stampare damiera

```
void stampaDamiera(char[][] damiera) {
  System.out.println( "  +-+-+-+-+-+-+-+-+" );
  int riga, colonna;
  for (riga = 7; riga>=0; riga = riga-1) {
    System.out.print( riga+" |" );
    for (colonna = 0; colonna<7; colonna = colonna+1) {
      System.out.print( damiera[riga][colonna]+"|" );
    }
    System.out.println( damiera[riga][colonna]+"|" );
    System.out.println( "  +-+-+-+-+-+-+-+-+" );
  }
  System.out.print( "   " );
  for (colonna = 0; colonna<7; colonna = colonna+1) {
    System.out.print( colonna+" " );
  }
  System.out.println( colonna );
}
```