

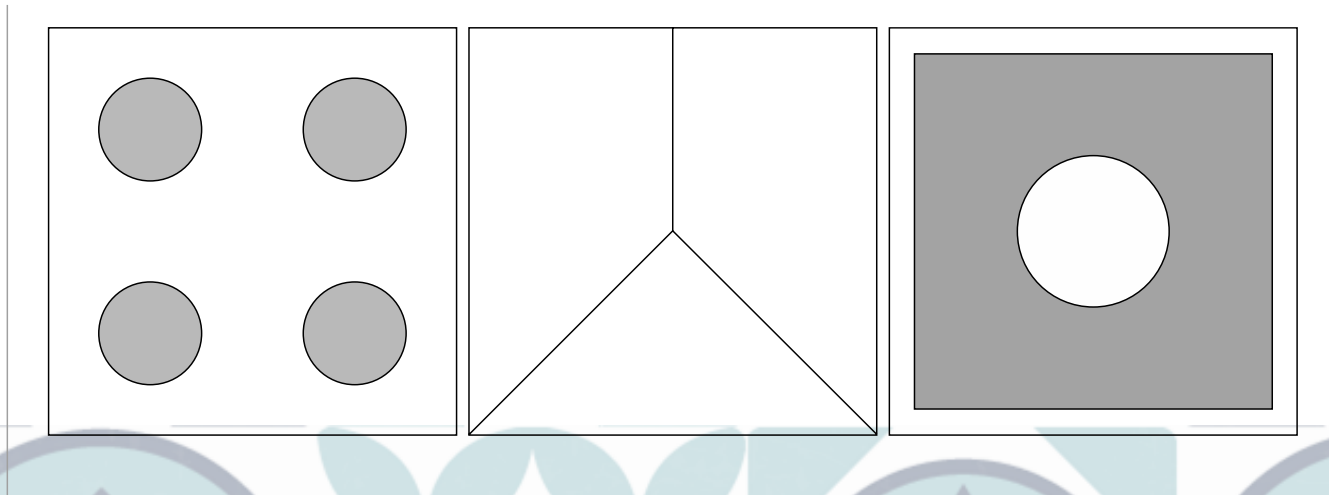
# FONDAMENTI DI INFORMATICA

Alma Artis  
Francesca Pratesi (ISTI, CNR)

Processing – Numeri casuali, istruzioni condizionali,  
cicli

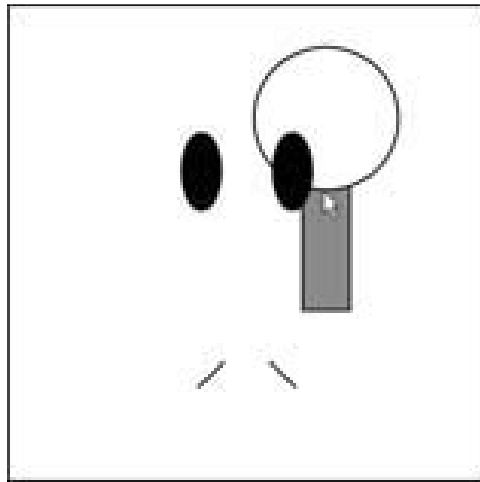
# ESERCIZIO

- Step 1: scrivere il codice per disegnare le seguenti figure con valori hard-coded (potete usare la scala di grigio o i colori)
- Step 2: Rimpiazzare tutti i valori numerici hard-coded con variabili opportune
- Step 3: Scrivere del codice in draw() che cambi il valore delle variabili. Per esempio «variabile1 = variabile1+2».



# ESERCIZIO: ZOOG INTERATTIVO

- Step 1: Disegna Zoog in modo da seguire la posizione del mouse
- Step 2: Il colore degli occhi dipende dalla posizione del mouse
- Step 3: Quando si clicca con il mouse viene visualizzato il messaggio "Take me to your leader!" (la stampa è possibile con la funzione `println()`)



*fig. 3.5*

# ESERCIZIO

- Modifica l'esempio del cerchio che si muove per fare in modo che il cerchio cresca anche di dimensione

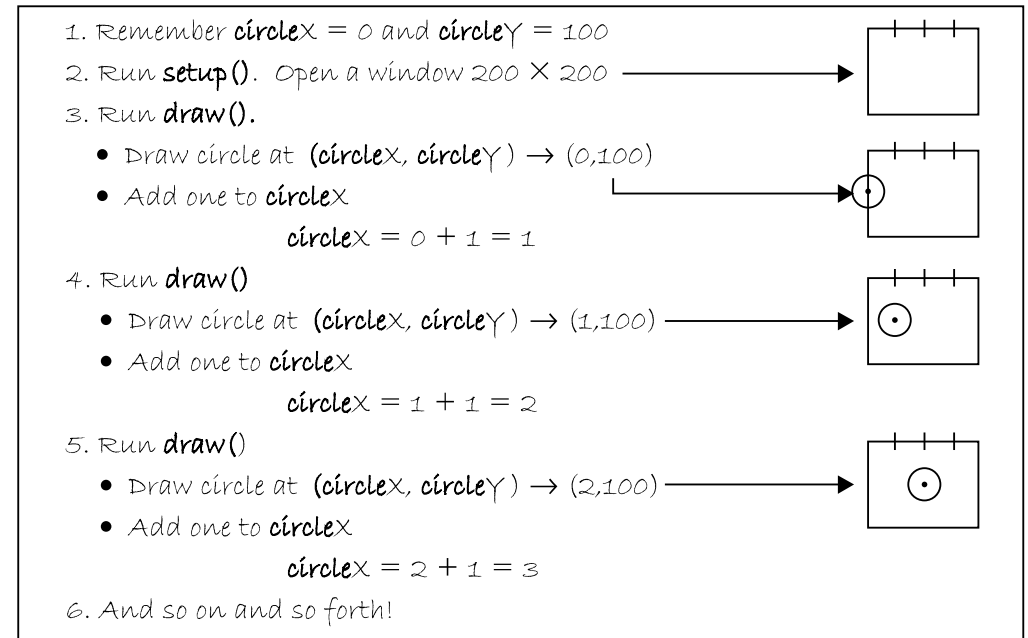


fig. 4.5



# NUMERI CASUALI

# NUMERI RANDOM

- La funzione `random()` è una funzione speciale che ritorna un valore casuale
- Rispetto alle altre funzioni viste finora (`ellipse`, `line`, `point`), questa risponde con un valore numerico
- La funzione ha bisogno di due numeri: viene ritornato un numero compreso tra i due
- La funzione ritorna un numero con la virgola: float
  - `float w = random(1,100);`  
`rect(100,100,w,50);`





# RANDOM ELLIPSES

```
float r, g, b, a;  
float diam;  
float x, y;
```

```
void setup() {  
  size(200,200);  
  background(0);  
  smooth();  
}
```

```
void draw() {  
  // assegno valori random alle variabili  
  r = random(255);  
  g = random(255);  
  b = random(255);  
  a = random(255);  
  diam = random(20);  
  x = random(width);  
  y = random(height);  
  // uso i valori soprastanti per disegnare ellissi  
  noStroke();  
  fill(r,g,b,a);  
  ellipse(x,y,diam,diam);  
}
```

Ad ogni chiamata di draw vengono estratti nuovi valori random per tutte le proprietà






# **ISTRUZIONI CONDIZIONALI**



# CONTROLLO DEL FLUSSO

- Quando un programma è formato da più istruzioni, queste vengono eseguite in ordine dall'alto verso il basso 
- Il seguente programma è composto da due istruzioni

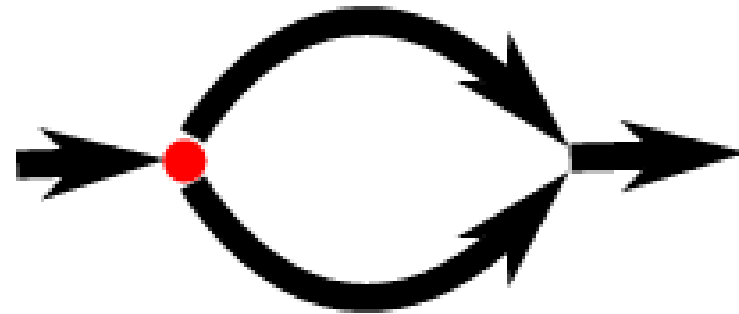
```
int theNumber = int(random(10, 100));  
println("Your number is " + theNumber);
```



# ISTRUZIONI CONDIZIONALI

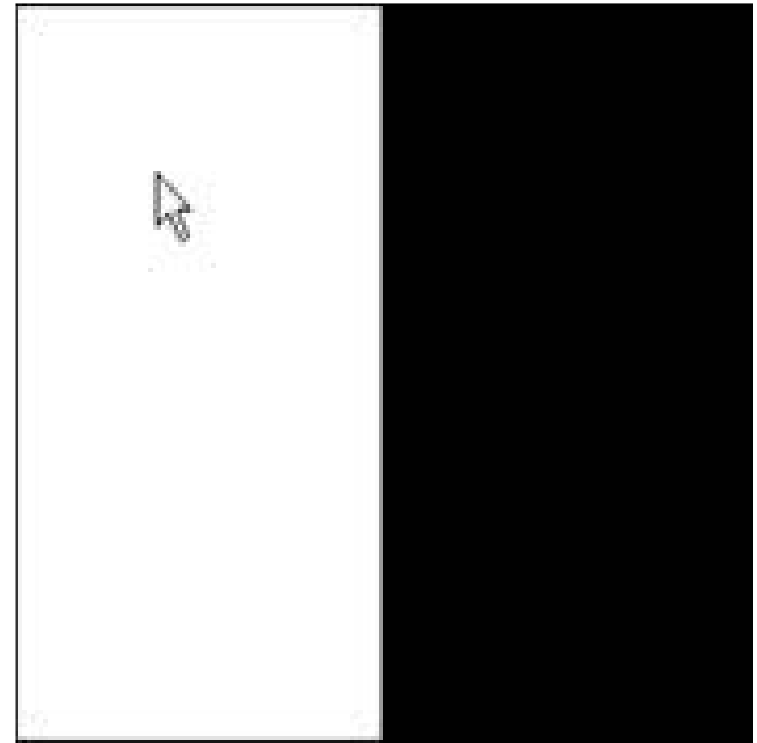
- Tutti i programmi vengono eseguiti in sequenza
- Queste istruzioni di controllo consentono di scegliere due possibili percorsi, sulla base di un valore booleano
- Sintassi:

```
if (espressione)
    istruzione1
else
    istruzione2
```
- Espressione è una espressione booleana
- Istruzione1 rappresenta il ramo eseguito se la valutazione ritorna true
- Istruzione2 rappresenta il ramo eseguito se la valutazione ritorna false



# ESEMPIO

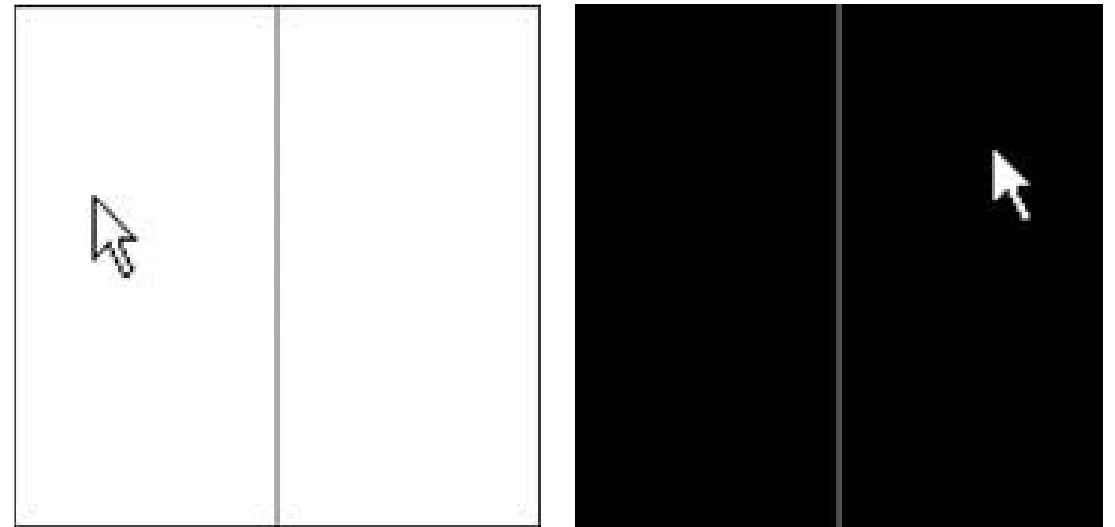
```
if(mouseX < width / 2){  
    fill(255);  
    rect(0,0,width/2,height);  
}
```



*fig. 5.1*

# ESEMPIO

```
if(mouseX < width/2){  
    background(255);  
}else{  
    background(0);  
}
```



*fig. 5.2*

# CONDIZIONI MULTIPLE

```
if (boolean expression #1) {  
    // codice da eseguire se expression #1 è true  
} else if (boolean expression #2) {  
    // codice da eseguire se expression #2 è true  
} else if (boolean expression #n) {  
    // codice da eseguire se expression #n è true  
} else {  
    // codice da eseguire se nessuna delle  
precedenti espressioni booleane è true  
}
```

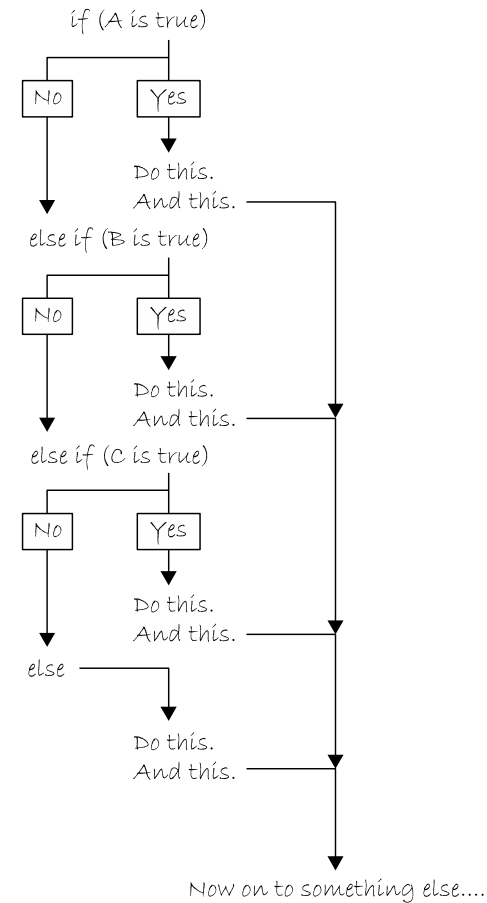
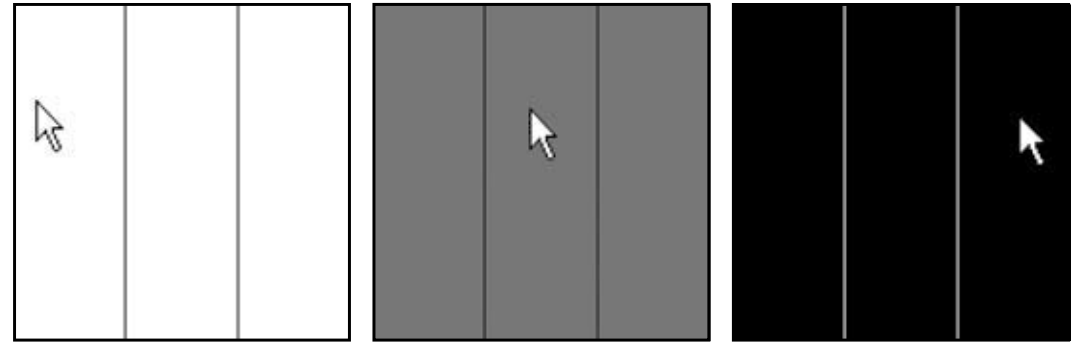


fig. 5.3

# ESEMPIO

```
if (mouseX < width/3) {  
    background(255);  
} else if (mouseX < 2*width/3) {  
    background(127);  
} else {  
    background(0);  
}
```



*fig. 5.4*



# SKETCH CONDIZIONALI

```
float r = 150;  
float g = 0;  
float b = 0;  
  
void setup(){  
  size(200,200);  
}
```

```
void draw(){  
  background(r,g,b);  
  stroke(255);  
  line(width/2,0,width/2,200);
```

```
  if(mouseX > width/2){  
    r=r+1;  
  } else {  
    r = r-1;  
  }
```

```
  if (r>255){  
    r=255;  
  } else if (r<0){  
    r=0;  
  }  
}
```

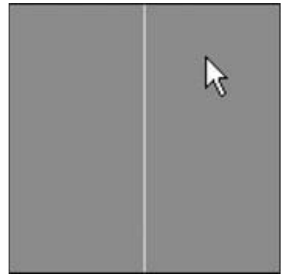


fig. 5.5

# FUNZIONE constrain()

```
if (r > 255) {  
    r = 255;  
} else if (r < 0) {  
    r = 0;  
}
```

Constrain with an “if” statement.

```
r = constrain(r, 0, 255);
```

Constrain with the ***constrain()*** function.

# ESERCIZIO 5-5

- Scrivere uno skect che implementi un rollover su un rettangolo
  - Quando il mouse passa sopra un rettangolo, questo cambia colore



# PICCOLO PROGETTO – ROLLOVER MULTIPLO



*Figure 5-7*



# ALGORITMO

- `setup()`
  - Crea una finestra di dimensioni 200x200
- `draw()`
  - Riempiamo il background di bianco
  - Disegniamo due linee per dividere lo spazio in quattro quadranti
  - Se il mouse è nell'angolo in alto a sinistra, colora il rettangolo corrispondente
  - Se il mouse è nell'angolo in alto a destra, colora il rettangolo corrispondente
  - Se il mouse è nell'angolo in basso a sinistra, colora il rettangolo corrispondente
  - Se il mouse è nell'angolo in basso a destra, colora il rettangolo corrispondente





# LOOPS - CICLI

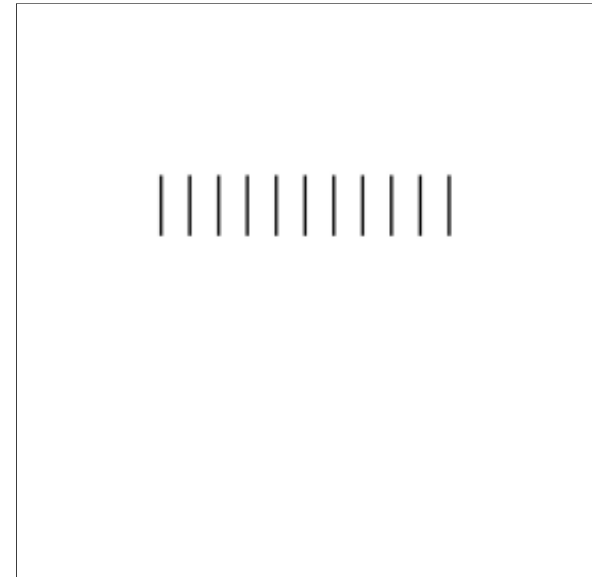


# ESEMPIO 6-1

## Example 6-1. Many lines

---

```
size(200, 200);  
background(255);  
  
// Legs  
stroke(0);  
line(50, 60, 50, 80);  
line(60, 60, 60, 80);  
line(70, 60, 70, 80);  
line(80, 60, 80, 80);  
line(90, 60, 90, 80);  
line(100, 60, 100, 80);  
line(110, 60, 110, 80);  
line(120, 60, 120, 80);  
line(130, 60, 130, 80);  
line(140, 60, 140, 80);  
line(150, 60, 150, 80);
```



*Figure 6-1*

# ESEMPIO 6-2

## Example 6-2 Many lines with variables

---

```
size(200, 200);
background(255);

// Legs
stroke(0);

int y = 80;      // Vertical location of each line
int x = 50;      // Initial horizontal location for first line
int spacing = 10; // How far apart is each line
int len = 20;    // Length of each line

line(x, y, x, y+len);

x = x + spacing;
line(x, y, x, y+len);

x = x + spacing;
line(x, y, x, y+len);

x = x + spacing;
line(x, y, x, y+len);

x = x + spacing;
```

Draw the first leg.

Add spacing so the next leg appears 10 pixels to the right.

Continue this process for each leg, repeating it over and over.

# ITERAZIONI

- Le istruzioni iterative permettono di ripetere l'esecuzione di parti del programma una o più volte
- Il numero di ripetizioni può essere fissato (iterazione determinata)
- Oppure può dipendere da una condizione (iterazione indeterminata)

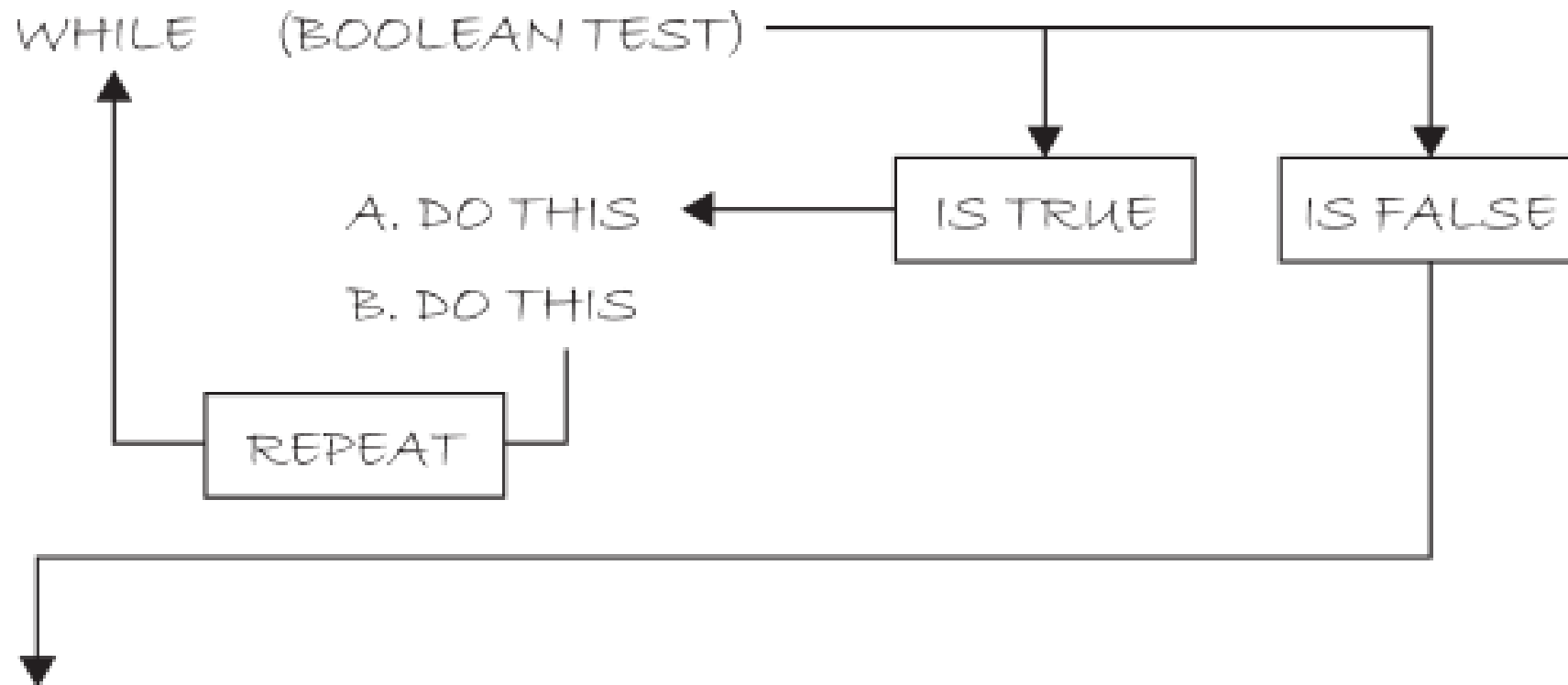


# ISTRUZIONE `while`

- L'espressione `while` è uno dei costrutti di iterazione di Java
- Sintassi  
    `while` (espressione)  
        istruzione
- espressione è una espressione booleana (detta **guardia**): se viene valutata a **true** allora viene eseguita l'istruzione interna (detta **corpo**) e si ripete il ciclo; se l'espressione booleana ritorna **false**, la ripetizione termina
- Se espressione è **false** fin dall'inizio, l'istruzione non viene mai eseguita



# ISTRUZIONE while



*Figure 6-2*

# ESEMPIO 6-3

## Example 6-3. While loop

```
int endLegs = 150;
```

```
stroke(0);
```

```
while (x <= endLegs) {  
  line (x, y, x, y+len);  
  x = x + spacing;  
}
```

A variable to mark where the legs end.

Draw each leg inside a while loop.



Figure 6-3



# ATTENZIONE AI CICLI INFINITI

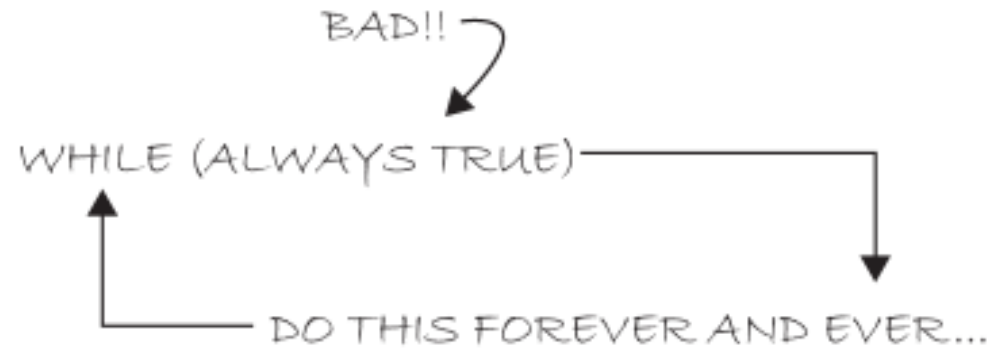


Figure 6-6

## Example 6-4. Infinite loop. Don't do this!

```
int x = 0;
while (x < 10) {
    println(x);
    x = x - 1;
}
```

Decrementing `x` results in an infinite loop here because the value of `x` will never be 10 or greater. Be careful!

# ISTRUZIONE for

- Alcuni casi di cicli visti finora utilizzano una variabile di controllo per contare il numero di iterazioni eseguite
- Questo tipo di soluzione si presenta molto spesso in fase di programmazione
- Il linguaggio prevede un costrutto apposito per delle iterazioni determinate
- Sintassi:  

```
for (expr1; expr2; expr3)  
    istruzione
```
- Dove **expr1** serve a inizializzare la variabile di controllo; **expr2** è la guardia di fine ciclo; **expr3** aggiorna la variabile di controllo; **istruzione** è il corpo del ciclo



# ISTRUZIONE for

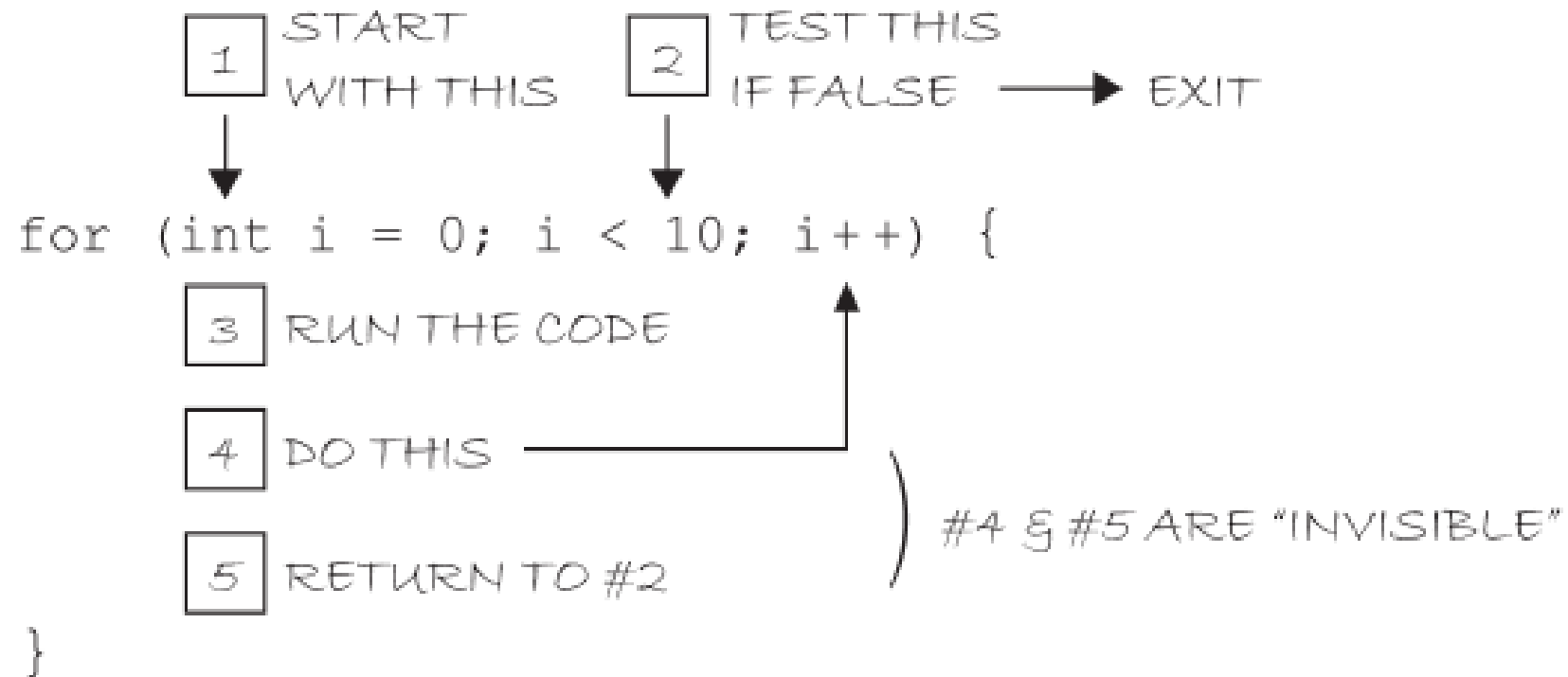


Figure 6-7

# ESEMPIO

- Scrivere un programma che calcoli il valore di  $2^{10}$

```
long result = 1;
```

```
for (int counter = 0; counter < 10; counter = counter + 1)
```

```
    result = result * 2;
```

```
println(result);
```

```
// → 1024
```



# AGGIORNAMENTO SUCCINTO DI VARIABILI

- Abbiamo visto diversi esempi di aggiornamento di variabili all'interno dei cicli
- Ad esempio alcune guardie avevano la seguente sintassi  
`counter = counter + 1`
- Per evitare di scrivere più volte la stessa variabile si può usare la seguente forma compatta  
`counter += 1`
- La stessa forma si può usare per altri operatori e operandi: `*=2`, `-=1`
- Per incrementi o decrementi di singole unità si possono anche usare le espressioni `counter++` e `counter--`



# ESERCIZIO 5-5

- Scrivere uno skect che implementi un rollover su un rettangolo
  - Quando il mouse passa sopra un rettangolo, questo cambia colore





# PICCOLO PROGETTO – ROLLOVER MULTIPLO



*Figure 5-7*



# ALGORITMO

- `setup()`
  - Crea una finestra di dimensioni 200x200
- `draw()`
  - Riempiamo il background di bianco
  - Disegniamo due linee per dividere lo spazio in quattro quadranti
  - Se il mouse è nell'angolo in alto a sinistra, colora il rettangolo corrispondente
  - Se il mouse è nell'angolo in alto a destra, colora il rettangolo corrispondente
  - Se il mouse è nell'angolo in basso a sinistra, colora il rettangolo corrispondente
  - Se il mouse è nell'angolo in basso a destra, colora il rettangolo corrispondente

