Programmazione di Artefatti Interattivi

By:
Davide Rocchesso

Programmazione di Artefatti Interattivi

By:
Davide Rocchesso

Online:
< http://cnx.org/content/col10417/1.9/ >

C O N N E X I O N S

Rice University, Houston, Texas

Collection structure revised: December 9, 2010
PDF generated: October 26, 2012
For copyright and attribution information for the modules contained in this collection, see p. 101.

Table of Contents

1 Pattern emergenti dal design dell'interazione 1
2 Programmare in Processing 9
3 Array e Mappe 25
4 Oscillazioni, ritardi, e fluttuazioni del tempo 37
5 Liste, pile e code 45
6 Thread e I/O non bloccante 57
7 Segnali nel tempo: soglie e filtri 67
8 Programmazione Visuale 77
9 Comunicazione tra Applicazioni 91
Glossary 99
Index 100
Attributions 101

Chapter 1

Pattern emergenti dal design dell'interazione[1]

"It is unfortunate that the teaching of design often concentrates almost exclusively on the visual aspect of things – worse still, without its reformulation as propositional knowledge." - ***Jorge Frascara, in Design Issues 23(4), 2007***

1.1 La Programmazione nel Design dell'Interazione

Un oggetto interattivo contiene almeno alcune delle seguenti componenti: sensori, attuatori, microprocessore, collegamento telematico. Queste componenti, per poter essere di supporto ad una qualsiasi forma di interazione, devono essere programmate. Il design dell'interazione, quando arriva alla costruzione di prototipi che consentano di esperire l'interazione in maniera diretta e veridica, non può prescindere dalla formulazione di programmi informatici. Questi sono una combinazione di algoritmi e strutture dati, cioè di metodi per organizzare e manipolare in maniera efficiente ed efficace le informazioni. In realtà, strumenti di programmazione efficienti si possono usare in tutte le attività che normalmente precedono o costituiscono il design di un prodotto interattivo:

- Basic design
- Sketching
- Prototyping

1.1.1 Basic Design

Nella progettazione di oggetti interattivi, in special modo se basati su interazione multisensoriale continua[2], ci sono molti gradi di libertà e lo spazio di progettazione risulta vastissimo. Una strategia per affrontarlo è quella di pensare costruttivamente in termini di fenomeni elementari. Cioè, dovremmo cercare le interaction gestalt[3] , astraendole dalle interazioni effettivamente effettuate. Questo processo è facilitato dalla pratica pedagogica del basic design[4] , nella quale gli studenti affinano le proprie abilità compositive mediante la ricerca di soluzioni a problemi ben posti, e per condivisione e discussione dei risultati. Il basic design non è orientato al prodotto, e si svolge per esercitazioni che esplorano le direzioni[5] :

- Esperienza sensoriale diretta;

[1] This content is available online at <http://cnx.org/content/m14501/1.11/>.
[2] "Continuous Multisensory Interaction" <http://cnx.org/content/m14293/latest/>
[3] http://www.idi.ntnu.no/~dags/interactivity.pdf
[4] http://newbasicdesign.it/
[5] http://homepage.mac.com/ranjanmp/.Public/Bauhaus_Ulm_NID_2005.pdf

- Oggettivizzazione dell'esperienza a livello intellettuale;
- Realizzazione per sintesi di fenomeni.

Example 1.1
The body of basics (C. Chiappini, 2007)[6] : "L'obiettivo dell'esercitazione è quello di ricavare un percorso interattivo attraverso delle immagini di azioni del vostro corpo". Risultati[7] .

Example 1.2
Multi-stage yet continuos coupling (Rocchesso e Polotti, 2008[8]): "L'obiettivo dell'esercitazione è quello di progettare il feedback multisensoriale per una connessione a vite (come quella della moka) in modo che il raggiungimento del giusto punto di chiusura sia facile, continuo, ed espressivo".

1.1.2 Sketching

Lo sketching[9] è una attività archetipica del processo di design. Il fatto che lo sketching tradizionalmente usato in architettura e nel design di prodotto non sia più adeguato pone la questione su cosa significhi fare sketching nel design dell'interazione (*Sketching User Experiences – Bill Buxton; Morgan Kaufmann, 2007*, Buxton's web site[10]). Tra le sue caratteristiche importanti, troviamo:

- Velocità
- Collezionabilità
- Bassa risoluzione
- Evocazione al posto di ri-produzione

Gli sketch servono a facilitare una miglior comprensione del problema e, quindi, a generare nuove idee. Così come nel brainstorming, la quantità è più importante della qualità. È normale, nelle fasi iniziali di un design, produrre molti sketch e confrontarli per giustapposizione. Se ci sono pattern importanti, questi dovrebbero emergere a prima vista. Una collezione di sketch abilita il confronto dialogico, come nel basic design. Sugli sketch non si fa valutazione. Attraverso gli sketch si cercano soluzioni.

1.1.3 Prototyping

Spesso[11] , lo sketching viene visto come prototipazione a bassa fedeltà. Tuttavia, mentre gli sketch hanno uno scopo principalmente generativo, i prototipi sono prodotti a fine di valutazione.

[6] http://www.room50.org/collettiva/chiappini/BRIEF_The%20body%20of%20basics.pdf
[7] http://www.room50.org/basicdesign/basic2007
[8] http://www.soundobject.org/BasicSID/basicSID/
[9] http://www.cs.cmu.edu/~bam/uicourse/Buxton-SketchesPrototypes.pdf
[10] http://www.billbuxton.com/
[11] http://www.id-book.com/chapter11_teaching.htm

Sketch vs. Prototype (Bill Buxton)

SKETCH	PROTOTYPE
evocative	didactic
suggest	describe
explore	refine
question	answer
propose	test
provoke	resolve
tentative	specific
non committal	depiction

Table 1.1

Le tre attività di basic design, sketching, e prototyping sono diverse e non c'è tra esse una relazione d'ordine. Nelle esercitazioni di basic design, si può procedere per produzione rapida di molti esemplari di soluzione (come nello sketching) e raffinarne alcuni per sottoporli a valutazione (come nel prototyping).

Nel design dell'interazione, è importante avere materiali, strumenti, e tecniche adeguate a tutte e tre le attività descritte. Processing[12] è un linguaggio e un ambiente di sviluppo adeguato per buona parte di questi scopi. È interessante notare come i programmi Processing vengano chiamati sketch, a enfatizzare la velocità con cui si possono ottenere bozzetti interattivi, utili a esplorare un vasto ventaglio di soluzioni.

1.2 Emergenza dall'Esperienza

Nel Laboratorio di Prototipazione di Artefatti Interattivi[13] , gli studenti di Gillian Crampton Smith[14] e Philip Tabor[15] hanno sviluppato alcuni prototipi di oggetti interattivi. Nel corso della realizzazione dei progetti si sono evidenziati alcuni problemi di programmazione, per i quali sono state trovate soluzioni ad hoc. Molti di questi problemi si ritrovano in molti ambiti diversi, e vale la pena di analizzarli per studiare delle soluzioni efficaci e di applicabilità generale.

Progetti del 2007[16] :

- Aequilibrium[17]
  - Object Oriented Programming (the fish class)
  - Ray tracing: distance, direction, reflection (of the fish)
  - Oscillations (of fish tails)
- Flyer Cafe[18]
  - Using a library API[19] (reacTIVision[20])
  - Stack (of geometric transformations)
  - Object Oriented Programming (not used yet)

[12] http://processing.org/
[13] http://www.interaction-venice.com/interaction-design-programme.html
[14] http://www.iuav.it/Facolta/facolt–di1/English-ve/faculty/teaching-f/GillianCra/index.htm
[15] http://www.iuav.it/Didattica1/pagine-web/facolt–di1/Philip-Tab/index.htm
[16] http://www.interaction-venice.com/courses/06-07Lab2/
[17] http://www.interaction-venice.com/courses/06-07Lab2/?page_id=127
[18] http://www.interaction-venice.com/courses/06-07Lab2/?page_id=128
[19] http://en.wikipedia.org/wiki/Api
[20] http://sourceforge.net/projects/reactivision/

- Secret Garden[21]
  - Using a library API[22] (BlobDetection[23])
  - Filtering (temporal smoothing)
  - Thresholding, hysteresis, and adaptation
  - Timers
- That Sinking Feeling[24]
  - Multithreading (not used yet)
  - Oscillations (waves)
  - Timers
- Tree of Life[25]
  - Mapping (distance to pixels)
  - Timers
  - Using a library API[26] (Sonia[27] , Video[28])
  - Thresholding and hysteresis
- Venice 360[29]
  - Using a library API[30] (Sonia[31])
  - Thresholding and hysteresis
  - Buffers (audio)
- WAV[32]
  - Oscillations (visual wave)
  - Matrices (of LEDs)
  - Timers (in visual programming)
  - Multiplexing/demultiplexing (in visual programming)

Progetti del 2008[33] :

- Si presentano pattern di programmazione simili a quelli dell'anno precedente. In alcuni casi si registra la necessità di fare comunicare tra loro applicazioni diverse (client-server).

1.3 Pattern

Nel design degli artefatti interattivi, così come in altre aree della progettazione, si può procedere alla risoluzione di un problema mediante identificazione di pattern[34] , cioè di configurazioni che ricorrono sovente, e per le quali esistono soluzioni di efficacia consolidata.

Un pattern si può descrivere mediante alcuni campi, tra i quali:

- Esempio (es., un sensore rileva la posizione di una persona in un corridoio, un'immagine sulla parete segue la persona come un'ombra)

[21] http://www.interaction-venice.com/courses/06-07Lab2/?page_id=129
[22] http://en.wikipedia.org/wiki/Api
[23] http://www.v3ga.net/processing/BlobDetection/
[24] http://www.interaction-venice.com/courses/06-07Lab2/?page_id=132
[25] http://www.interaction-venice.com/courses/06-07Lab2/?page_id=133
[26] http://en.wikipedia.org/wiki/Api
[27] http://sonia.pitaru.com/
[28] http://processing.org/reference/libraries/video/index.html
[29] http://www.interaction-venice.com/courses/06-07Lab2/?page_id=134
[30] http://en.wikipedia.org/wiki/Api
[31] http://sonia.pitaru.com/
[32] http://www.interaction-venice.com/courses/06-07Lab2/?page_id=135
[33] http://www.interaction-venice.com/projects/iuav07-08Lab2/
[34] http://en.wikipedia.org/wiki/Pattern_language

- Problema (es., il sensore produce numeri che dipendono nonlinearmente dalla distanza in metri. Lo spazio dell'immagine sulla parete è misurabile in pixel. Come traduco in pixel i numeri prodotti dal sensore?)
- Vincoli (es., la traduzione deve essere efficiente e precisa in tutti i punti)
- Principio (es., è un mapping non lineare uno-a-uno)
- Soluzione (es., Se ho mille pixel, posso riempire un array di mille celle con i valori misurati dal sensore, eventualmente interpolando a partire da poche misure. Tuttavia, a me serve la mappa inversa, che dal sensore va ai pixel. Cioè, dovrei trovare l'indice di una cella dell'array dato il suo contenuto. Una maniera per rendere efficiente l'accesso per contenuto è di organizzare una struttura dati che si chiama hash. Oppure, usando l'array ordinato proveniente dalle misure, posso effettuare una ricerca per bisezione.)

1.4 Riferimenti

Al mondo ci sono più testi di programmazione che esemplari di Panda gigante. Molti di questi sono buoni e utili, e pochi tra i buoni e gli utili sono liberamente accessibili. In particolare, segnalo per la sua chiarezza e compattezza il free book[35] *How to Think Like a Computer Scientist: Java Version – Allen B. Downey; Green Tea Press 2003.* E' anche accessibile nei suoi contenuti[36] *Data Structures and Algorithms with Object-Oriented Design Patterns in Java – Bruno R. Preiss, John Wiley and Sons 1999.* Tra i vari linguaggi ai quali fanno riferimento i manuali di programmazione, certamente Java è quello più indicato per gli utilizzatori di Processing[37] , visto che Processing è costruito su Java, e di questo mette a disposizione l'intero repertorio di classi. Gli autori di Processing hanno scritto il libro *Processing: A Programming Handbook for Visual Designers and Artists – Casey Reas and Ben Fry, MIT Press 2007.* Un buon testo di introduzione alla programmazione in Processing è *Learning Processing: A Beginner's Guide to Programming Images, Animation, and Interaction – Daniel Shiffman, Morgan Kaufmann, 2008.*

Exercise 1.1 ***(Solution on p. 6.)***

There is a crowd of autonomous communicating creatures in a room, like in a cocktail party. Each has some kind of visual display, such as a light bulb. You want to select one individual in the crowd and make that selection known to everybody by coordinating individual light modulations. One possibility is a propagating wavefront, like in Mexican waves [38] at the stadium. A Processing sketch to test this phenomenon can be written in less than fifteen minutes.

Exercise 1.2 ***(Solution on p. 6.)***

Modify the sketch of problem Exercise 1.1 to implement an imploding wavefront, and compare the two simulations to see which is the most effective for highlighting one individual in the crowd.

Exercise 1.3 ***(Solution on p. 7.)***

A crowd in a room is not static. Add a sort of random walk[39] to the sketch of problem Exercise 1.2, and you will notice that the wavefront becomes barely visible. You will need to play with color to making it noticeable again.

[35] http://www.greenteapress.com/thinkapjava/
[36] http://www.brpreiss.com/books/opus5/
[37] http://processing.org/
[38] http://en.wikipedia.org/wiki/Mexican_wave
[39] http://en.wikipedia.org/wiki/Random_walk

Solutions to Exercises in Chapter 1

Solution to Exercise 1.1 (p. 5)
Click over the window produced by the following code:

```
int NOBJ = 100;
int WINSIZE = 100;
int[] objects_x = new int[NOBJ];
int[] objects_y = new int[NOBJ];
int[] distance = new int[NOBJ];
int counter = 0;

void setup() {
 size(WINSIZE, WINSIZE);
 for (int i=0; i<NOBJ; i++) {
   objects_x[i] = int(random(0, WINSIZE-1));
   objects_y[i] = int(random(0, WINSIZE-1));
 }
}

void mouseReleased() {
 for (int i=0; i<NOBJ; i++) {
   distance[i] = int(sqrt(sq(objects_x[i] - mouseX) + sq(objects_y[i] - mouseY)));
 }
 counter = 0;
}

void draw() {
  background(200);
  counter = counter+1;
  for (int i=0; i<NOBJ; i++) {
    if (distance[i] == counter){
      strokeWeight(4);
      point(objects_x[i], objects_y[i]);
    }
    else {
      strokeWeight(2);
      point(objects_x[i], objects_y[i]);
    }
  }
}
```

Solution to Exercise 1.2 (p. 5)
Click over the window produced by the following code:

```
int NOBJ = 100;
int WINSIZE = 100;
```

```
int[] objects_x = new int[NOBJ];
int[] objects_y = new int[NOBJ];
int[] distance = new int[NOBJ];
int counter = 0;

void setup() {
 size(WINSIZE, WINSIZE);
 for (int i=0; i<NOBJ; i++) {
   objects_x[i] = int(random(0, WINSIZE-1));
   objects_y[i] = int(random(0, WINSIZE-1));
 }
}

void mouseReleased() {
 for (int i=0; i<NOBJ; i++) {
   distance[i] = int(sqrt(sq(objects_x[i] - mouseX) + sq(objects_y[i] - mouseY)));
 }
 counter = WINSIZE - 1;
}

void draw() {
  background(200);
  counter = counter - 1;
  for (int i=0; i<NOBJ; i++) {  //ciclo for
    if (distance[i] == counter){
      strokeWeight(4);
      point(objects_x[i], objects_y[i]);
    }
    else {
      strokeWeight(2);
      point(objects_x[i], objects_y[i]);
    }
  }
}
```

Solution to Exercise 1.3 (p. 5)
Click over the window produced by the following code:

```
int NOBJ = 100;
int WINSIZE = 100;
int[] objects_x = new int[NOBJ];
int[] objects_y = new int[NOBJ];
int[] distance = new int[NOBJ];
int counter = 0;
float randomConstant = 1.1;

void setup() {
 size(WINSIZE, WINSIZE);
 for (int i=0; i<NOBJ; i++) {
```

```
   objects_x[i] = int(random(0, WINSIZE-1));
   objects_y[i] = int(random(0, WINSIZE-1));
 }
 frameRate(WINSIZE/2);
}

void mouseReleased() {
 for (int i=0; i<NOBJ; i++) {
   distance[i] = int(sqrt(sq(objects_x[i] - mouseX) + sq(objects_y[i] - mouseY)));
 }
 counter = WINSIZE - 1;
}

void draw() {
  background(200);
  counter = counter - 1;
  for (int i=0; i<NOBJ; i++) {
    objects_x[i] =constrain(objects_x[i]+int(random(-randomConstant,randomConstant)),
       0, WINSIZE-1);
    objects_y[i] =constrain(objects_y[i]+int(random(-randomConstant,randomConstant)),
       0, WINSIZE-1);
    if (distance[i] == counter){
      strokeWeight(8);
      stroke(255,0,0);
      point(objects_x[i], objects_y[i]);
    }
    else {
      strokeWeight(2);
      stroke(120,120,0);
      point(objects_x[i], objects_y[i]);
    }
  }
}
```

Chapter 2

Programmare in Processing[1]

2.1 Introduzione

NOTE: Questa introduzione è basata sul tutorial di Daniel Shiffman [2] .

Processing è un linguaggio ed un ambiente di sviluppo orientato all' **interaction design** . Nel corso **Elaborazione di Media in Processing**, Processing è uno degli strumenti principali utilizzati per introdurre elementi di elaborazione di suoni e immagini. Processing è una estensione di Java che supporta molte delle strutture Java con una sintassi semplificata.

Processing può essere utilizzato in tre:

Modi di Programmazione

Basic - Sequenza di comandi per il disegno di primitive grafiche –

applet senza naso[3]	`size(256,256);`
	`background(0);`
	`stroke(255);`
	`ellipseMode(CORNER);`
	`ellipse(72,100,110,130);`
	`triangle(88,100,168,100,128,50);`
	`stroke(140);`
	`strokeWeight(4);`
	`line(96,150,112,150);`
	`line(150,150,166,150);`
	`line(120,200,136,200);`

Table 2.1

Intermediate - Programmazione procedurale –

[1] This content is available online at <http://cnx.org/content/m12614/1.16/>.
[2] http://www.shiffman.net/itp/classes/ppaint/
[3] http://cnx.org/content/m12614/latest/pinocchiononose.html

applet con naso[4]	`void setup() {`
	`  size(256,256);`
	`  background(0);`
	`}`
	
	`void draw() {`
	`  stroke(255);`
	`  strokeWeight(1);`
	`  ellipseMode(CORNER);`
	`  ellipse(72,100,110,130);`
	`  triangle(88,100,168,100,128,50);`
	`  stroke(140);`
	`  beginShape(TRIANGLES);`
	`  vertex(114, 180);`
	`  vertex(mouseX, mouseY);`
	`  vertex(140, 180);`
	`  endShape();`
	`  strokeWeight(4);`
	`  line(96,150,112,150);`
	`  line(150,150,166,150);`
	`  line(120,200,136,200);`
	`}`

Table 2.2

Complex - Programmazione Orientata agli Oggetti (Java) –

[4] http://cnx.org/content/m12614/latest/pinocchionose.html

applet con naso colorato[5]	`Puppet pinocchio;`
	
	`void setup() {`
	`  size(256,256);`
	`  background(0);`
	`  color tempcolor = color(255,0,0);`
	`  pinocchio = new Puppet(tempcolor);`
	`}`
	`void draw() {`
	`  background(0);`
	`  pinocchio.draw();`
	`}`
	`class Puppet  {`
	`  color colore;`
	`  Puppet(color c_) {`
	`  colore = c_;`
	`  }`
	`  void draw () {`
	`  stroke(255);`
	`  strokeWeight(1);`
	`  ellipseMode(CORNER);`
	`  ellipse(72,100,110,130);`
	`  stroke(colore);`
	`  beginShape(TRIANGLES);`
	`  vertex(114, 180);`
	`  vertex(mouseX, mouseY);`
	`  vertex(140, 180);`
	`  endShape();`
	`  strokeWeight(4);`
	`  line(96,150,112,150);`
	`  line(150,150,166,150);`
	`  }`
	`}`

Table 2.3

I programmi Processing possono essere convertiti in applet Java. Per fare ciò è sufficiente andare nel menu **File** e scegliere **Export** . Il risultato finale sarà dunque la creazione di cinque file, inseriti nel folder `applet`:

- **index.html** - sorgente html per visualizzare la applet
- **filename.jar** - la applet compilata, completa di tutti i dati (immagini, suoni, ecc.) necessari
- **filename.pde** - il codice sorgente Processing
- **filename.java** - il codice Java che incorpora il codice sorgente Processing
- **loading.gif** - un'immagine che viene mostrata mentre si attende il caricamento della applet.

Inoltre, mediante **Export Application** è possibile generare una applicazione eseguibile per piattaforma Linux, MacOS, o Windows.

[5] http://cnx.org/content/m12614/latest/pinocchioclassy.html

2.2 Tipi di Dati

2.2.1 Variabili

Una variabile è un puntatore a una locazione di memoria, e può riferirsi a valori primitivi (`int`, `float`, ecc.) oppure ad oggetti o array (tabelle di elementi di un tipo primitivo).

L'operazione di assegnamento `b = a` produce

- Se le variabili si riferiscono a tipi primitivi, la copia del contenuto di `a` in `b`.
- Se le variabili si riferiscono a oggetti o array, la creazione di un nuovo riferimento allo stesso oggetto o array.

NOTE: Per avere chiaro il significato di termini informatici quali, ad esempio, quelli che seguono, si consiglia di cercare in Wikipedia[6]

Definition 2.1: scope
all'interno di un programma, regione in cui si può accedere ad una variabile e modificarne il valore

Definition 2.2: global scope
definita fuori da setup() e draw(), la variabile è visibile e usabile ovunque nel programma

Definition 2.3: local scope
definita all'interno di un blocco di codice o di una funzione, la variabile assume valori locali al blocco o alla funzione, ed eventuali valori assunti da una variabile globale omonima vengono ignorati

Example 2.1: Dichiarazione e allocazione di un array

```
int[] arrayDiInteri = new int[10];
```

2.3 Strutture di Programmazione

2.3.1 Istruzioni Condizionali

- if:

```
if (i == NMAX) {
   println("finito");
   }
else {
   i++;
   }
```

2.3.2 Iterazioni

- while:

```
int i = 0; //contatore intero
while (i < 10) { //scrivi i numeri da 0 a 9
  println("i = "+ i);
```

[6] http://wikipedia.org/

```
  i++;
}
```

- for:

```
for (int i = 0; i < 10; i++) { //scrivi i numeri da 0 a 9
  println("i = "+ i);
}
```

Example 2.2: Inizializzazione di una tabella di numeri casuali

```
int MAX = 10;
float[] tabella = new float[MAX];
for (int i = 0; i < MAX; i++)
   tabella[i] = random(1); //random numbers between 0 and 1
println(tabella.length + " elementi:");
        println(tabella);
```

2.4 Funzioni

Le funzioni consentono un approccio modulare alla programmazione. In Processing, in modalità di programmazione **intermediate**, si possono definire funzioni oltre alla `setup()` e `draw()`, usabili all'interno della `setup()` e della `draw()`.

Example 2.3: Esempio di Funzione

```
int raddoppia(int i) {
   return 2*i;
}
```

Una funzione è caratterizzata dalle seguenti entità (con riferimento all'esempio (Example 2.3: Esempio di Funzione)) :

- tipo di ritorno (`int`)
- nome di funzione (`raddoppia`)
- parametri (`i`)
- corpo della funzione (`return 2*i`)

2.5 Classi e Oggetti

Una classe è definita da un insieme di dati e funzioni. Un oggetto è una istanza di una classe. Viceversa, una classe è una descrizione astratta di un insieme di oggetti.

NOTE: Per una introduzione ai concetti di oggetto e di classe si veda Objects and Classes[7].

Example 2.4: Esempio di Classe

```
Dot myDot;
void setup() {
  size(300,20);
  colorMode(RGB,255,255,255,100);
  color tempcolor = color(255,0,0);
  myDot = new Dot(tempcolor,0);
}

void draw() {
  background(0);
  myDot.draw(10);
}

class Dot
{

  color colore;
  int posizione;

  //****CONSTRUCTOR*****//
  Dot(color c_, int xp) {
    colore = c_;
    posizione = xp;
  }

  void draw (int ypos)    {
    rectMode(CENTER);
    fill(colore);
    rect(posizione,ypos,20,10);
  }
}
```

Una classe è caratterizzata dalle seguenti entità (con riferimento all'esempio (Example 2.4: Esempio di Classe)) :

- nome di classe (`Dot`)
- dati (`colore, posizione`)
- costruttore (`Dot()`)
- funzioni (o metodi, `draw()`)

[7] "Objects and Classes" <http://cnx.org/content/m11708/latest/>

Un oggetto (istanza di una classe) è dichiarato nello stesso modo in cui si dichiara una variabile, ma ad esso va poi allocato uno spazio (come si è visto per gli array) tramite il suo costruttore (con riferimento all'esempio (Example 2.4: Esempio di Classe)).

- Dichiarazione: (`Dot myDot;`)
- Allocazione: (`myDot = new Dot(tempcolor,0)`)
- Utilizzazione: (`myDot.draw(10);`)

NOTE: Per una introduzione alla sintassi Java si veda Java Syntax Primer[8]

Exercise 2.1 ***(Solution on p. 19.)***

Con il metodo `draw()` seguente si vuole dipingere lo sfondo della finestra di un grigio la cui intensità dipenda dalla posizione orizzontale del puntatore del mouse.

```
void draw() {
    background((mouseX/100)*255);
}
```

Il codice però non fa quello che ci si aspetta. Perché?

Exercise 2.2 ***(Solution on p. 19.)***

Cosa stampa il seguente frammento di codice?

```
int[] a = new int[10];
a[7] = 7;
int[] b = a;
println(b[7]);
b[7] = 8;
println(a[7]);
int c = 7;
int d = c;
println(d);
d = 8;
println(c);
```

Exercise 2.3 ***(Solution on p. 19.)***

Il seguente sketch si pone l'obbiettivo di generare un set di 100 cerchi in movimento e di disegnare nella finestra di Processing tutte le corde congiungenti i due punti di intersezione di tutte le coppie di cerchi a intersezione non nulla.

```
/*

   Structure 3

   A surface filled with one hundred medium to small sized circles.
   Each circle has a different size and direction, but moves at the same slow rate.
   Display:
   A. The instantaneous intersections of the circles
```

[8] "Java Syntax Primer" <http://cnx.org/content/m11791/latest/>

```
   B. The aggregate intersections of the circles

   Implemented by Casey Reas <http://groupc.net>
   8 March 2004
   Processing v.68 <http://processing.org>

   modified by Pietro Polotti
   28 March, 2006
   Processing v.107 <http://processing.org>


*/

int numCircle = 100;
Circle[] circles = new Circle[numCircle];

void setup()
{
  size(800, 600);
  frameRate(50);
  for(int i=0; i<numCircle; i++) {
    circles[i] = new Circle(random(width),
      (float)height/(float)numCircle * i,
      int(random(2, 6))*10, random(-0.25, 0.25),
      random(-0.25, 0.25), i);
  }
  ellipseMode(CENTER_RADIUS);
  background(255);
}


void draw()
{
  background(255);
  stroke(0);


  for(int i=0; i<numCircle; i++) {
    circles[i].update();
  }
  for(int i=0; i<numCircle; i++) {
    circles[i].move();
  }
  for(int i=0; i<numCircle; i++) {
    circles[i].makepoint();
  }
  noFill();
}


class Circle
```

```
{
  float x, y, r, r2, sp, ysp;
  int id;

  Circle( float px, float py, float pr, float psp, float pysp, int pid ) {
    x = px;
    y = py;
    r = pr;
    r2 = r*r;
    id = pid;
    sp = psp;
    ysp = pysp;
  }

  void update() {
    for(int i=0; i<numCircle; i++) {
      if(i != id) {
        intersect( this, circles[i] );
      }
    }
  }

  void makepoint() {
     stroke(0);
     point(x, y);
  }

  void move() {
    x += sp;
    y += ysp;
    if(sp > 0) {
      if(x > width+r) {
        x = -r;
      }
    } else {
      if(x < -r) {
        x = width+r;
      }
    }
    if(ysp > 0) {
      if(y > height+r) {
        y = -r;
      }
    } else {
      if(y < -r) {
        y = height+r;
      }
    }
  }
}
```

```
void intersect( Circle cA, Circle cB )
{
  float dx = cA.x - cB.x;
  float dy = cA.y - cB.y;
  float d2 = dx*dx + dy*dy;
  float d = sqrt( d2 );

  if ( d>cA.r+cB.r || d<abs(cA.r-cB.r) ) {
    return; // no solution
  }

  //  calculate the two intersections between the two circles cA and cB, //
  //  whose coordinates are (paX, paY) and (pbX, pbY), respectively.     //

  stroke(255-dist(paX, paY, pbX, pbY)*4);
  line(paX, paY, pbX, pbY);
}
```

1. Completare la parte mancante inerente il calcolo delle intersezioni dei cerchi, per poter disegnare le corde congiungenti le coppie di punti intersezione. E' possibile basarsi sul calcolo delle coordinate delle intersezioni in un sistema di riferimento ad hoc (**"Intersezione cerchio-cerchio"**) e poi riportare il risultato nelle coordinate della finestra di Processing.
2. Rendere le corde variabili nel tempo dando diverse velocità di spostamento ai cerchi.

Exercise 2.4 ***(Solution on p. 21.)***

Rendere interattivo lo sketch di Exercise 2.3, facendo per esempio dipendere la quantità di spostamento dei cerchi dalla posizione lungo l'asse x del mouse.

Solutions to Exercises in Chapter 2

Solution to Exercise 2.1 (p. 15)
La variabile `mouseX` è di tipo `int`, e quindi la divisione a cui è sottoposta è di tipo intero. E' necessario fare un **type casting** da `int` a `float` mediante `(float)mouseX`.
Solution to Exercise 2.2 (p. 15)

```
7
8
7
7
```

Solution to Exercise 2.3 (p. 15)

```
/*

   Structure 3

   A surface filled with one hundred medium to small sized circles.
   Each circle has a different size and direction, but moves at the same slow rate.
   Display:
   A. The instantaneous intersections of the circles
   B. The aggregate intersections of the circles

   Implemented by Casey Reas <http://groupc.net>
   8 March 2004
   Processing v.68 <http://processing.org>

   modified by Pietro Polotti
   28 March, 2006
   Processing v.107 <http://processing.org>


*/

int numCircle = 100;
Circle[] circles = new Circle[numCircle];

void setup()
{
  size(800, 600);
  frameRate(50);
  for(int i=0; i<numCircle; i++) {
    circles[i] = new Circle(random(width),
      (float)height/(float)numCircle * i,
      int(random(2, 6))*10, random(-0.25, 0.25),
      random(-0.25, 0.25), i);
  }
  ellipseMode(CENTER_RADIUS);
```

```
  background(255);
}


void draw()
{
  background(255);
  stroke(0);


  for(int i=0; i<numCircle; i++) {
    circles[i].update();
  }
  for(int i=0; i<numCircle; i++) {
    circles[i].move();
  }
  for(int i=0; i<numCircle; i++) {
    circles[i].makepoint();
  }
  noFill();
}


class Circle
{
  float x, y, r, r2, sp, ysp;
  int id;

  Circle( float px, float py, float pr, float psp, float pysp, int pid ) {
    x = px;
    y = py;
    r = pr;
    r2 = r*r;
    id = pid;
    sp = psp;
    ysp = pysp;
  }

  void update() {
    for(int i=0; i<numCircle; i++) {
      if(i != id) {
        intersect( this, circles[i] );
      }
    }
  }

  void makepoint() {
     stroke(0);
     point(x, y);
  }
```

```
  void move() {
    x += sp;
    y += ysp;
    if(sp > 0) {
      if(x > width+r) {
        x = -r;
      }
    } else {
      if(x < -r) {
        x = width+r;
      }
    }
    if(ysp > 0) {
      if(y > height+r) {
        y = -r;
      }
    } else {
      if(y < -r) {
        y = height+r;
      }
    }
  }
}

void intersect( Circle cA, Circle cB )
{
  float dx = cA.x - cB.x;
  float dy = cA.y - cB.y;
  float d2 = dx*dx + dy*dy;
  float d = sqrt( d2 );

  if ( d>cA.r+cB.r || d<abs(cA.r-cB.r) ) {
    return; // no solution
  }

  float a = (cA.r2 - cB.r2 + d2) / (2*d);
  float h = sqrt( cA.r2 - a*a );
  float x2 = cA.x + a*(cB.x - cA.x)/d;
  float y2 = cA.y + a*(cB.y - cA.y)/d;

  float paX = x2 + h*(cB.y - cA.y)/d;
  float paY = y2 - h*(cB.x - cA.x)/d;
  float pbX = x2 - h*(cB.y - cA.y)/d;
  float pbY = y2 + h*(cB.x - cA.x)/d;

  stroke(255-dist(paX, paY, pbX, pbY)*4);
  line(paX, paY, pbX, pbY);
}
```

Solution to Exercise 2.4 (p. 18)

```
    /*

   Structure 3

   A surface filled with one hundred medium to small sized circles.
   Each circle has a different size and direction, but moves at the same slow rate.
   Display:
   A. The instantaneous intersections of the circles
   B. The aggregate intersections of the circles

   Implemented by Casey Reas <http://groupc.net>
   8 March 2004
   Processing v.68 <http://processing.org>

   modified by Pietro Polotti
   28 March, 2006
   Processing v.107 <http://processing.org>


*/

int numCircle = 100;
Circle[] circles = new Circle[numCircle];

void setup()
{
  size(800, 600);
  frameRate(50);
  for(int i=0; i<numCircle; i++) {
    circles[i] = new Circle(random(width),
      (float)height/(float)numCircle * i,
      int(random(2, 6))*10, random(-0.25, 0.25),
      random(-0.25, 0.25), i);
  }
  ellipseMode(CENTER_RADIUS);
  background(255);
}


void draw()
{
  background(255);
  stroke(0);

  if(mousePressed){
  for(int i=0; i<numCircle; i++) {
    circles[i].sp = mouseX*random(-5, 5)/width;
```

```
  }
  }

  for(int i=0; i<numCircle; i++) {
    circles[i].update();
  }
  for(int i=0; i<numCircle; i++) {
    circles[i].move();
  }
  for(int i=0; i<numCircle; i++) {
    circles[i].makepoint();
  }
  noFill();
}


class Circle
{
  float x, y, r, r2, sp, ysp;
  int id;

  Circle( float px, float py, float pr, float psp, float pysp, int pid ) {
    x = px;
    y = py;
    r = pr;
    r2 = r*r;
    id = pid;
    sp = psp;
    ysp = pysp;
  }

  void update() {
    for(int i=0; i<numCircle; i++) {
      if(i != id) {
        intersect( this, circles[i] );
      }
    }
  }

  void makepoint() {
     stroke(0);
     point(x, y);
  }

  void move() {
    x += sp;
    y += ysp;
    if(sp > 0) {
      if(x > width+r) {
        x = -r;
      }
```

```
    } else {
      if(x < -r) {
        x = width+r;
      }
    }
    if(ysp > 0) {
      if(y > height+r) {
        y = -r;
      }
    } else {
      if(y < -r) {
        y = height+r;
      }
    }
  }
}

void intersect( Circle cA, Circle cB )
{
  float dx = cA.x - cB.x;
  float dy = cA.y - cB.y;
  float d2 = dx*dx + dy*dy;
  float d = sqrt( d2 );

  if ( d>cA.r+cB.r || d<abs(cA.r-cB.r) ) {
    return; // no solution
  }

  float a = (cA.r2 - cB.r2 + d2) / (2*d);
  float h = sqrt( cA.r2 - a*a );
  float x2 = cA.x + a*(cB.x - cA.x)/d;
  float y2 = cA.y + a*(cB.y - cA.y)/d;

  float paX = x2 + h*(cB.y - cA.y)/d;
  float paY = y2 - h*(cB.x - cA.x)/d;
  float pbX = x2 - h*(cB.y - cA.y)/d;
  float pbY = y2 + h*(cB.x - cA.x)/d;

  stroke(255-dist(paX, paY, pbX, pbY)*4);
  line(paX, paY, pbX, pbY);
}
```

Chapter 3

Array e Mappe[1]

3.1 Array

Un array è una lista numerata di valori, tutti dello stesso tipo (`int`, `float`, ecc.). La numerazione, in Processing (e Java), parte da zero. La dichiarazione e creazione di un array avviene in maniera uguale a quella di un oggetto istanza di classe. Ad esempio

```
    float[] vettore = new float[100];
println(vettore);
```

crea un array di 100 elementi di tipo `float` e ne stampa il contenuto, che è inizialmente costituito da zeri. L'array così dichiarato è a tutti gli effetti un oggetto istanza di classe, e tale classe ha anche una variabile `length`, che ne contiene la lunghezza. Quindi, per inizializzare l'array come una rampa lineare che va da 0 a 1, si può fare

```
for (int i = 0; i < vettore.length; i++)
    vettore[i] = (float)i/vettore.length;
println(vettore);
```

NOTE: Si provi a togliere il typecasting[2] `(float)` e si spieghi perché la rampa non viene più generata.

Example 3.1: 1-to-1 mapping
Un MIDI controller[3] continuo, come uno slider, può inviare valori tra 0 e 127, che possono essere usati come indici per leggere i valori di una certa proprietà. Un array può servire a mappare un controllo in un valore di proprietà. La relazione tra controllo (indice) e proprietà è una mappa uno a uno. Nel codice che segue, la mappa è costruita muovendo sopra alla finestra il mouse con tasto premuto. Invece, se il mouse viene spostato senza essere premuto, la posizione orizzontale del puntatore (tra 0 e 127) viene interpretata come indice della mappa, che restituisce il valore della proprietà di brillantezza dello sfondo.

[1] This content is available online at <http://cnx.org/content/m14505/1.10/>.
[2] http://en.wikipedia.org/wiki/Type_conversion#Explicit_type_conversion
[3] http://en.wikipedia.org/wiki/Midi_controller

```
int ris = 128;
int[] midimap = new int[128];

void setup() {
 size(ris,2*ris);
}

void draw() {
 background(255);
 if ((mouseX >= 0) && (mouseX < ris))
   if (!mousePressed)
     background((256 - midimap[mouseX]));
   else
     midimap[mouseX] = mouseY;

 for (int i=0; i<midimap.length; i++) point(i, midimap[i]);
}
```

Come si possono evitare i "buchi" nella mappa dovuti alla lettura non continua delle posizioni del mouse?

Estensione di risoluzione

Ho un controllore MIDI (a 128 valori) e voglio scandire una tabella di 256 valori. Si può iterare la lettura avanzando il punto di lettura con passo 2.

```
int  sup_ris = 256;
int  inf_ris = 128;
float[] midimap = new float[sup_ris];

void setup() {
 size(inf_ris,inf_ris);
 colorMode(RGB, 1.0);
 for (int i = 0; i < midimap.length; i++) midimap[i] = sin(PI*(float)i/midimap.length);
}

void draw() {
 if ((mouseX >= 0) && (mouseX < inf_ris) && (mousePressed))
 {
   point(mouseX, inf_ris - inf_ris*midimap[sup_ris/inf_ris*mouseX]);
 }
}
```

Riduzione di risoluzione

Ho un controllore a 10 bit (1024 valori) e devo mapparlo in una proprietà secondo una tabella di 128 valori che descrivono un semiperiodo di seno.

```
int inf_ris = 128;
int sup_ris = 1024;
```

```
float[] midimap = new float[inf_ris];

void setup() {
 size(sup_ris,32);
 colorMode(RGB, 1.0);
 for (int i = 0; i < midimap.length; i++) midimap[i] = sin(PI*(float)i/midimap.length);
}

void draw() {
 if ((mouseX >= 0) && (mouseX < sup_ris) && (mousePressed))
   background(midimap[int(float(mouseX)/sup_ris*inf_ris)]);
}
```

Come posso accedere alla tabella sfruttando per quanto possibile l'accuratezza dei 10 bit? Si può fare una **interpolazione lineare**. Il beneficio dell'interpolazione diventa molto evidente nel passaggio da 10 a 3 bit, cioè ponendo `inf_ris = 8`. Con l'interpolazione lineare, il metodo `draw()` va così riscritto:

```
void draw() {
 if ((mouseX >= 0) && (mouseX < sup_ris) && (mousePressed))
 {
   int flo = floor(float(mouseX)/sup_ris*inf_ris);
   float alpha = float(mouseX)/sup_ris*inf_ris - float(flo);
   background((1-alpha)*midimap[flo] + alpha*midimap[(flo+1)%midimap.length]);
 }
}
```

`alpha` è il resto del troncamento a $\log_2$inf_res bit dell'indice di $\log_2$sup_res bit, e viene usato per pesare due elementi contigui dell'array a `inf_res` elementi. Quale è la funzione dell'operazione di modulo `%midimap.length`?

Inversione di una mappa

Ha senso costruire una mappa inversa, ad esempio per ritrovare l'indice a partire da un valore di proprietà, solo se la mappa di partenza è monotona, crescente o decrescente, in modo da stabilire una corrispondenza biunivoca tra indice e valore. Ad esempio, pre-caricando un array con un quarto di seno campionato in 128 punti, come costruisco un array che contiene la mappa inversa, cioè l'arcoseno?

```
int ris = 128;
float[] midimap = new float[ris];
float[] invmap = new float[ris];

void setup() {
 size(ris,ris);
 colorMode(RGB, 1.0);
 /* inizializza */
 for (int i = 0; i < midimap.length; i++) midimap[i] = (ris - 1) * sin(PI/2*(float)i/midimap.length);
 /* inverti */
 for (int i = 0; i < midimap.length; i++) invmap[round(midimap[i])] = i;
 /* disegna */
 for (int i = 0; i < midimap.length; i++)
```

```
   {
     point(i, ris - midimap[i]);
     point(i, ris - invmap[i]);
   }
 println(invmap);
}
```

Exercise 3.1 *(Solution on p. 34.)*

Si noti, dal printout prodotto dalla inversione di una mappa (Inversione di una mappa, p. 27), che la mappa inversa così costruita ha parecchi buchi (zeri). Come si possono eliminare tali buchi?

Mappe lineari

In molti casi i sensori si comportano linearmente e ciò che serve è una mappa lineare. In tal caso, non è necessario scomodare un array per realizzarla. Ad esempio, si supponga che un sensore di distanza restituisca un valore in metri che corrisponde alla quantizzazione della distanza massima di 10 metri in 1024 valori (10 bit). Se con tale sensore vogliamo mappare diverse posizioni da 1 a 3 metri in indici di pixel da 0 a 127, si può semplicemente notare che la distanza di 1 metro produce il valore 102, mentre la distanza di 3 metri produce il valore 307. Detto d il valore di distanza tra 0 e 1023 restituito dal sensore, il pixel da accendere è $\frac{128}{307-102}(d-102)$

Istogrammi

Le operazioni che si basano sul conteggio di elementi vengono supportate bene da rappresentazioni mediante array. E' questo il caso della produzione dell'istogramma[4] . In elaborazione di immagini questa è una operazione che si fa assai di frequente in quanto supporta diverse elaborazioni efficaci dei colori.

Lettura

Si legga il capitolo 10 di How to Think Like a Computer Scientist[5]

3.2 Array nel visual programming

Gli array, come liste ordinate di numeri, si prestano naturalmente ad una rappresentazione visuale, e quindi trovano un impiego esteso nel visual programming[6] , quale è quello che si pratica con gli ambienti Max/MSP[7] e Pure Data[8] (Pd).

In Pd un array può essere allocato mediante la primitiva `table`, alle quale viene passato un nome e una lunghezza. Tutti gli array sono memorizzati come numeri floating point. Il contenuto degli array può essere visualizzato in una finestra facendo "open" sull'oggetto `table`. Ad esempio, se creo un oggetto con `table mappa 64`, l'interprete costruisce un subpatch contenente un array, il quale si può visualizzare cliccando sull'oggetto stesso. Si può accedere ad ogni singolo elemento dell'array con gli oggetti `tabread mappa` e `tabwrite mappa`, rispettivamente. In più, gli array in Pd hanno metodi che supportano varie operazioni. Ad esempio:

- una sequenza di valori crescente tra -3 a 3 che inizia alla locazione di posizione 4 si può impostare con il messaggio `; mappa 4 -3 -2 -1 0 1 2 3`;
- l'array di nome `mappa` si può rinominare `map` mediante il messaggio `; mappa rename map`;
- il riquadro che contiene la rappresentazione visuale dell'array si può ridimensionare con un messaggio del tipo `; map bounds 0 -2 10 2`
- Sulla cornice del riquadro si possono porre delle marcature metriche con un messaggio del tipo `; map xticks 0 0.1 5` nonché dei valori di riferimento con `; map xlabel -1.1 0 1 2 3 4 5`

[4] http://cnx.org/content/m12838/latest/#histogramp
[5] http://www.greenteapress.com/thinkapjava/
[6] http://en.wikipedia.org/wiki/Visual_programming
[7] http://en.wikipedia.org/wiki/Max/MSP
[8] http://en.wikipedia.org/wiki/Pure_Data

3.3 Matrici

Qualsiasi tipo si può usare come tipo base per un array (`int`, `float`, `string`, ecc.). In particolare, un array può essere tipo base per un array, in questo modo consentendo di fare array di array, o array bidimensionali. Un array bidimensionale si dichiara e si istanzia con

```
int[][] A = new int[ROWS][COLUMNS];
```

dove `ROWS` e `COLUMNS` conterranno i numeri di righe e colonne che si vogliono riservare per l'array. Ogni riga è essa stessa un array, al quale si può accedere con `A[i]`, mentre all'elemento di riga `i` e colonna `j` si accede con `A[i][j]`, che in questo caso è una variabile di tipo `int`. Questa volta `A.length` restituisce il numero di righe dell'array, mentre il numero di colonne si può conoscere con `A[0].length`.

Un array, mono- o bi-dimensionale, può essere inizializzato all'atto della creazione. Nel caso di un array a 3 righe e 4 colonne ciò si può fare con

```
int[][]  A  =  {  {  1,  0, 12, -1 },
                        {  7, -3,  2,  5 },
                        { -5, -2,  2, -9 }
                     };
```

Così come gli array monodimensionali vengono scanditi agevolmente con un ciclo `for`, così gli array bidimensionali vengono scanditi con due cicli `for` annidati, uno che scandisce le righe e uno che scandisce le colonne:

```
for (int i = 0; i < ROWS; i++) {
     for (int j = 0; j < COLUMNS; j++) {
         A[i][j] = ...
     }
}
```

Ad esempio, nel codice seguente viene inizializzato un array bidimensionale di colori, e successivamente questa tavolozza di colori viene visualizzata.

```
int ROWS = 4;
int COLUMNS = 4;
color[][] grid = new color[ROWS][COLUMNS];

for (int row = 0; row < ROWS; row++) {
   for (int col = 0; col < COLUMNS; col++) {
      grid[row][col] = color(row*60, 0, col*60);
   }
}

for (int row = 0; row < ROWS; row++) {
   for (int col = 0; col < COLUMNS; col++) {
      fill(grid[row][col]);
```

```
        rect(row*25, col*25, 25, 25);
    }
  }
```

Exercise 3.2 *(Solution on p. 34.)*

Si visualizzi una tavolozza di 4x4 colori che ruota in senso orizzontale, come se fosse la superficie di un cilindro che ruota intorno all'asse verticale.

Gli array multidimensionali di tipo `float` o `double` sono utilizzati in moltissimi contesti, in quanto sono di supporto alla Aritmetica delle Matrici[9]. Questa aritmetica consente di rappresentare ed effettuare in maniera compatta operazioni su array di numeri. Ad esempio, per invertire l'ordine dei numeri di un array di 4 elementi è sufficiente pre-moltiplicarlo per una matrice $\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$ oppure, per scambiare i numeri in posizione pari con quelli in posizione dispari si può usare la matrice $\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

Nella computer grafica interattiva, punti e vettori sono rappresentati in coordinate omogenee [10] mediante array di quattro elementi, e tutte le trasformazioni geometriche[11] (traslazioni, rotazioni, scalamenti, proiezioni) sono realizzate con prodotti matrice-vettore.

Prodotto Matrice-Vettore

L'inversione tra elementi di posto pari ed elementi di posto dispari mediante prodotto matrice-vettore è illustrata dal codice

```
int DIM = 4;
float[][] myMatrix = {{0, 1, 0, 0},
                      {1, 0, 0, 0},
                      {0, 0, 0, 1},
                      {0, 0, 1, 0}
                     };

        float[] myVector = {0.1, 0.2, 0.3, 0.4};
        float[] newVector = new float[4];

        for (int i=0; i<DIM; i++)
          for (int j=0; j<DIM; j++)
             newVector[i] += myMatrix[i][j] * myVector[j];

        println(myVector); println();
        println(newVector);
```

[9] "Matrix Arithmetic" <http://cnx.org/content/m10090/latest/>

[10] "Rappresentazione di Media in Processing", Definition 1: "Coordinate Omogenee" <http://cnx.org/content/m12664/latest/#coordinate_omogenee>

[11] "Rappresentazione di Media in Processing": Section Traslazioni, Rotazioni, e Trasformazioni di Scala <http://cnx.org/content/m12664/latest/#rotazioni_traslazioni>

Exercise 3.3 *(Solution on p. 35.)*
Come è fatta la matrice che, pre-moltiplicata per un array, ne ruota circolarmente di una posizione gli elementi?

Example 3.2: Many-to-Many Mapping
In molte applicazioni di interaction design (si veda, ad esempio, il caso della interactive sonification[12]), ci si trova a dover mappare una molteplicità di variabili, il cui valore può venire dalla lettura di sensori, in una molteplicità di parametri algoritmici. Spesso tale mapping è lineare ed esprimibile con un prodotto matrice-vettore. Ad esempio, **distanza, velocità, e temperatura** si possono mappare in **posizione, colore, e frequenza di oscillazione** di un oggetto grafico, in maniera che, ad esempio, il colore dipenda da una combinazione di velocità e temperatura. Il funzionamento delle Mixing Console[13] si può interpretare come mapping multivariato lineare degli ingressi nelle uscite. Purtroppo, molti mapping di interesse sono non-lineari, e non esprimibili mediante prodotto matrice-vettore. Ad esempio, la trasformazione delle coordinate di rappresentazione dei colori[14] da RGB a HSB[15] è non lineare.

Lettura
Si legga la sezione Multi-Dimensional Arrays[16] di Data Structures and Algorithms with Object-Oriented Design Patterns in Java[17] o, ancor meglio, la sezione Multi-Dimensional Arrays[18] di Introduction to Programming Using Java[19] .

3.4 Mappe

Gli array soffrono di due limitazioni principali:

- la loro occupazione di memoria va pre-impostata;
- gli indici sono obbligatoriamente numeri interi.

Quando si parla di mappe, in ambito delle strutture dati, ci si riferisce di solito a strutture più generali, la cui dimensione è dinamicamente allocabile, e nelle quali l'accesso agli elementi può avvenire attraverso oggetti di tipo arbitrario, chiamati **key** (chiave).
Lettura
Si legga il capitolo 19 di How to Think Like a Computer Scientist[20]

3.5 Matrici nel Physical Computing

Nel progetto WAV[21] , la classe Matrix[22] per l'ambiente Wiring[23] di physical computing[24] è stata usata per pilotare una matrice di LED a 8 righe e 16 colonne per mezzo di un paio di chip MAXIM MAX7219[25] . Supponiamo per semplicità di dover gestire una matrice a 8 righe e 8 colonne per mezzo di un solo chip MAX7219 e di voler propagare una cresta d'onda da sinistra a destra. La matrice viene dichiarata e istanziata come

[12] http://www.icad.org/websiteV2.0/Conferences/ICAD2004/papers/hunt_hermann.pdf
[13] http://en.wikipedia.org/wiki/Mixing_console
[14] "Rappresentazione di Media in Processing" <http://cnx.org/content/m12664/latest/#color_type>
[15] http://en.wikipedia.org/wiki/HSV_color_space#From_RGB_to_HSV
[16] http://www.brpreiss.com/books/opus5/html/page89.html#SECTION005200000000000000000
[17] http://www.brpreiss.com/books/opus5/
[18] http://math.hws.edu/javanotes/c7/s5.html
[19] http://math.hws.edu/javanotes/
[20] http://www.greenteapress.com/thinkapjava/
[21] http://www.interaction-venice.com/courses/06-07Lab2/?page_id=135
[22] http://wiring.org.co/reference/libraries/Matrix/index.html
[23] http://wiring.org.co/index.html
[24] http://itp.nyu.edu/physcomp/
[25] http://www.maxim-ic.com/quick_view2.cfm/qv_pk/1339

```
Matrix myMatrix = Matrix(0, 2, 1);
```

Si può predisporre un array bidimensionale di bit 8x8 mediante istanziazione della classe Sprite[26] :

```
Sprite wave = Sprite(
                    8, 8,
                    b10000000,
                    b10000000,
                    b01000010,
                    b01000010,
                    b00100100,
                    b00100100,
                    b00011000,
                    b00011000,
                    );
```

e scrivere una funzione da eseguirsi a clockrate

```
void loop()
        {
          myMatrix.write(i, 1, wave);     // place sprite on screen (i is an int)
          delay(1000);                    // wait a little bit
          myMatrix.clear();               // clear the screen for next animation frame
          i = 1 + (i+1)%8;                // circular (between 1 and 8) increment of i
        }
```

oppure, alternativamente, procedere direttamente alla scrittura ciclica

```
void loop()
        {
          myMatrix.clear(); // clear display

          delay(1000);

          // turn pixels on (i is an int that is forced to stay between 0 and 7)
  // decreasing ramp
          myMatrix.write(1, 1+(i)%8, HIGH);
          myMatrix.write(2, 1+(i)%8, HIGH);
          myMatrix.write(3, 1+(i+1)%8, HIGH);
          myMatrix.write(4, 1+(i+1)%8, HIGH);
          myMatrix.write(5, 1+(i+2)%8, HIGH);
          myMatrix.write(6, 1+(i+2)%8, HIGH);
          myMatrix.write(7, 1+(i+3)%8, HIGH);
          myMatrix.write(8, 1+(i+3)%8, HIGH);
  // increasing ramp
          myMatrix.write(3, 1+(i+6)%8, HIGH);
          myMatrix.write(4, 1+(i+6)%8, HIGH);
          myMatrix.write(5, 1+(i+5)%8, HIGH);
          myMatrix.write(6, 1+(i+5)%8, HIGH);
```

[26] http://wiring.org.co/reference/libraries/Sprite/index.html

```
        myMatrix.write(7, 1+(i+4)%8, HIGH);
        myMatrix.write(8, 1+(i+4)%8, HIGH);
i++;
      }
```

Solutions to Exercises in Chapter 3

Solution to Exercise 3.1 (p. 28)
Il codice seguente riempie i buchi per interpolazione, ma non funziona se i buchi sono fatti da più zeri contigui:

```
for (int i = 1; i < invmap.length-1; i++)
   if (invmap[i] < 0.0001) // se buco, fai la media degli adiacenti
     invmap[i] = (invmap[i-1] + invmap[i+1])/2; /* non funziona se i buchi sono fatti
       da più zeri contigui */
 println(invmap);
```

Solution to Exercise 3.2 (p. 30)

```
int ROWS = 4;
int COLUMNS = 4;
color[][] grid = new color[ROWS][COLUMNS];
int step;


void setup(){
 for (int row = 0; row < ROWS; row++) {
    for (int col = 0; col < COLUMNS; col++) {
       grid[row][col] = color(row*60, 0, col*60);
    }
         frameRate(5);
         noStroke();
 }

 for (int row = 0; row < ROWS; row++) {
    for (int col = 0; col < COLUMNS; col++) {
       fill(grid[row][col]);
       rect(row*25, col*25, 25, 25);
    }
 }
}

void draw(){
  for (int row = 0; row < ROWS; row++) {
    for (int col = 0; col < COLUMNS; col++) {
       fill(grid[(row+step)%COLUMNS][col]);
       rect(row*25, col*25, 25, 25);
    }
 }
  step = (step+1)%COLUMNS;
}
```

Solution to Exercise 3.3 (p. 31)

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$ Una matrice fatta così è un caso particolare di matrice circolante[27] .

[27] http://en.wikipedia.org/wiki/Circulant_matrix

Chapter 4

Oscillazioni, ritardi, e fluttuazioni del tempo[1]

Tra i pattern emergenti (Chapter 1) nel design dell'interazione, si nota la necessità di produrre fenomeni oscillatori, di tipo uditivo o visuale. La gestione più versatile di questi fenomeni si fa mediante array (Chapter 3) contenenti un periodo del fenomeno oscillatorio da produrre.

4.1 L'oscillatore tabellare

L'approccio più classico e versatile alla sintesi di forme d'onda periodiche è la lettura ciclica di una tabella contenente un periodo della forma d'onda periodica da generare. Detto `buf[]` il buffer (è un array) contenente la forma d'onda da generare, l'oscillatore tabellare funziona per lettura ciclica della tabella, con un passo di avanzamento $I = \frac{Bf}{R}$, dove B è la lunghezza del buffer, f è la frequenza (numero di oscillazioni al secondo) che si vuole generare, e R è il rate (frequenza di campionamento) al quale si vuole produrre l'oscillazione.

Example 4.1: Onda propagantesi
Per dare l'illusione di una onda prodotta da un gesto di interazione e propagantesi in una certa direzione si può fare uso dell'oscillatore tabellare. Nel codice che segue il buffer viene precaricato con un periodo di sinusoide. Il punto in cui l'utilizzatore clicca sulla finestra determina la posizione iniziale della cresta, e viene generata una oscillazione la cui frequenza è impostata tramite la variabile `f`.

```
int B = 256; // lunghezza di tabella (risoluzione)
int R = 20;   // rate
float f = 0.8; // frequenza in cicli al secondo
float[] buf = new float[B];
int HEIGHT = 128; // altezza della finestra
float amp; // ampiezza della oscillazione
int readPoint; // posizione orizzontale della cresta
float decayFactor = 0.95; // fattore di decadimento dell'onda

void setup() {
 size(B,HEIGHT);
 stroke(255); strokeWeight(2);
 for (int i = 0; i < buf.length; i++) buf[i] = sin(2*PI*(float)i/buf.length);
```

[1] This content is available online at <http://cnx.org/content/m14532/1.4/>.

```
}

void draw() {
 background(0);

 if ((mouseX >= 0) && (mouseX < B) && (mousePressed)) {
   amp = 2*(HEIGHT/2 - mouseY)/float(HEIGHT);
   readPoint = mouseX;
 }
 for (int i = 0; i < buf.length; i++) {
   point(i, HEIGHT - HEIGHT/2 - HEIGHT/2*amp*buf[(i - readPoint + B/4 + B)%B]);
 }
 amp = decayFactor*amp;
 readPoint = (readPoint + round(f*B/R))%B; // incremento dell'oscillatore
}
```

Exercise 4.1 *(Solution on p. 42.)*
Si renda la frequenza di oscillazione dipendente dalla lunghezza dell'intervallo di tempo durante il quale il tasto del mouse viene tenuto premuto.

Exercise 4.2 *(Solution on p. 42.)*
Si imponga un inviluppo di ampiezza sulla oscillazione generata, come se a vibrare fosse una corda vincolata alle estremità.

Exercise 4.3
Si renda l'oscillazione più fluida sostituendo l'arrotondamento (`round()`) con l'interpolazione lineare.

4.2 Oscillazioni video

L'oscillatore tabellare può essere la struttura di riferimento anche per ripetizioni cicliche, a frequenza controllabile, di frammenti video. In questo caso la tabella da leggere ciclicamente contiene una sequenza di immagini.

Example 4.2: Iterazione di frammento video
In questo esempio, quando si manda in esecuzione il programma un frammento di 4 secondi di video viene ripreso dalla videocamera a 16 frame al secondo. Poi tale frammento è riprodotto ciclicamente a 20 frame al secondo, con la possibilità di controllare la velocità di riproduzione (frequenza di oscillazione) cliccando in diverse posizioni orizzontali del mouse.

```
import processing.video.*;
Capture myCapture;
int captureRate = 16;
float f=1;
int B = 64;
int R = 20;
PImage[] sequenza = new PImage[B];

int i;
float j=0;
```

```
float inc = f*B/R;
boolean via = false;

void setup() {
  for (int i=0; i<B; i++) sequenza[i]=loadImage("vetro.jpg"); // immagine dummy, per inizializzare
  size(200, 200);
  String s = "IIDC FireWire Video";
  myCapture = new Capture(this, width, height, s, 30);
  myCapture.frameRate(captureRate);
  frameRate(R);
}

void mousePressed(){
  f = float(mouseX)/width; inc = f*B/R;
}

void draw() {
  if (!via) {
    if(myCapture.available()) {
      // Reads the new frame
      myCapture.read();
      sequenza[i].copy(myCapture,0,0,myCapture.width,myCapture.height,0,0,sequenza[i].width,sequenza[i]
      i = (i+1)%B;
      println(i);
      if (i==0) via = true;
    }
  }
  if (via) {  image(sequenza[floor(j)], 0, 0,200,200);
              j = (j+inc);  if (j>=B) j=j-B;
            }
}
```

4.3 Oscillazioni nel visual programming

Nel visual programming le oscillazioni tabellari spesso si costruiscono utilizzando un segnale sawtooth[2] che consente di percorre ciclicamente gli indici dell'array, ad una velocità dipendente dalla frequenza del segnale stesso.

[2] http://en.wikipedia.org/wiki/Sawtooth

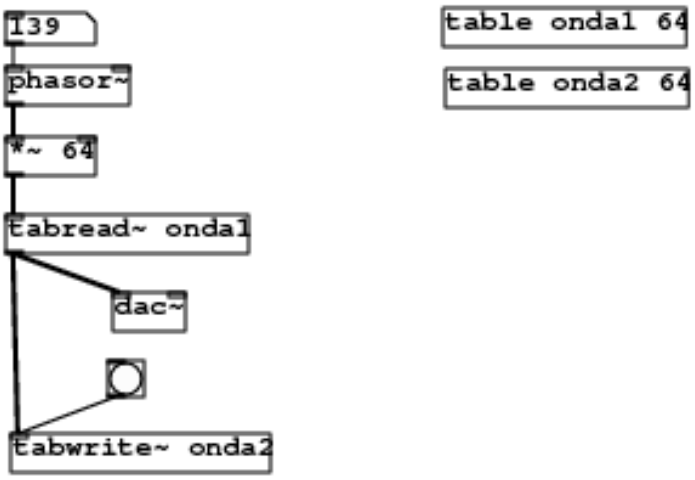

Figure 4.1

In questo patch sono dichiarati due array `onda1` e `onda2` di 64 elementi ciascuno. Il segnale a dente di sega viene generato ad una frequenza di 139 cicli al secondo. Il suo range viene moltiplicato per 64 in maniera da occupare tutto il campo degli indici (da 0 a 63) dell'array `onda1`. Ogni volta che si preme il bottone di `bang`, 64 campioni del segnale prodotto dalla lettura della tabella `onda1` vengono scritti nella tabella `onda2`. Il segnale prodotto dalla lettura ciclica della tabella `onda1` è udibile attraverso conversione da digitale ad analogico (`dac~`). Si noti che in Pd tutti gli operatori che hanno il simbolo di tilde lavorano in maniera sincrona ad audio rate, e quindi la lettura dalla tabella `onda1` fornisce campioni al sample rate di funzionamento (per default 44100 Hz). Si noti anche la scalinatura del segnale scritto nella tabella `onda2`, dovuto alla lettura non interpolata dei campioni di `onda1`.

Exercise 4.4 ***(Solution on p. 43.)***

Come si può evitare la scalinatura nella forma d'onda prodotta dall'oscillatore tabellare di Figure 4.1?

4.4 Ritardi variabili

Nei sistemi interattivi c'è spesso la necessità di ritardare nel tempo un segnale, un processo, o un evento. Ad esempio, se ci sono più flussi di video o audio in streaming, ciascuno viaggiante su canali indipendenti per i quali non è possibile avere lo stesso tempo di propagazione da trasmittente a fruitore, è opportuno ritardare diversamente ciascun flusso in maniera da avere un riallineamento ed una sincronizzazione in ricezione. In campo audio, le **linee di ritardo** a lunghezza variabile sono alla base di molti effetti[3], quali sono echi e riverberazione.

La struttura di supporto per la realizzazione delle linee di ritardo è il buffer circolare, cioè un array ai cui elementi si accede con aritmetica circolare dei puntatori. In particolare, c'è una variabile `IN`, chiamata puntatore di ingresso, che contiene l'indice dell'elemento dell'array nel quale si va a scrivere. Una variabile `OUT`, chiamata puntatore di uscita, contiene l'indice dell'elemento dell'array dal quale si va a leggere. Si

[3] "Time Delays for Audio Effects" <http://cnx.org/content/m11657/latest/>

tratta di incrementare i puntatori di accesso in maniera circolare, mantenendo la loro distanza relativa pari al numero di passi temporali di cui si vuole che sia fatto il ritardo. Ad ogni passo temporale (o istante di campionamento) il segnale di ingresso è scritto nella locazione puntata da `IN` e letto dalla locazione puntata da `OUT`, `D` passi indietro. Quindi, i due puntatori sono aggiornati con le operazioni

```
IN = (IN + 1) % B;
OUT = (OUT + 1) % B;
```

dove `B` è la lunghezza del buffer, scelta in maniera da essere maggiore del più grande valore di ritardo `D` che si intende usare. Il ritardo `D` può variare dinamicamente, ad esempio oscillando tra un minimo ed un massimo, ma è chiaro che se esso assume un valore non intero bisognerà adottare una strategia di interpolazione per la lettura del segnale ritardato. Ad esempio, si può ancora una volta fare interpolazione lineare tra locazioni adiacenti, oppure scegliere l'intero immediatamente inferiore a `D`.

Exercise 4.5 ***(Solution on p. 43.)***

Si costruisca un programma Processing che legge caratteri dalla tastiera e ne fornisce una eco sulla finestra grafica con un ritardo crescente mano a mano che si aggiungono linee di testo.

Solutions to Exercises in Chapter 4

Solution to Exercise 4.1 (p. 38)

```
int B = 256; // lunghezza di tabella (risoluzione)
int R = 20;   // rate
float f = 1.1; // frequenza in cicli al secondo
float[] buf = new float[B];
int HEIGHT = 128; // altezza della finestra
float amp; // ampiezza della oscillazione
float readPoint; // posizione orizzontale della cresta
float decayFactor = 0.92; // fattore di decadimento dell'onda
float tempo;

void setup() {
 size(B,HEIGHT);
 stroke(255); strokeWeight(2);
 for (int i = 0; i < buf.length; i++) buf[i] = sin(2*PI*(float)i/buf.length);
}

void mousePressed() {
  tempo = millis();
}

void mouseReleased() {
  tempo = millis() - tempo; println(tempo);
  f = 500/tempo;
}

void draw() {
 background(0);

 if ((mouseX >= 0) && (mouseX < B) && (mousePressed)) {
   amp = 2*(HEIGHT/2 - mouseY)/float(HEIGHT);
   readPoint = mouseX;
 }
 for (int i = 0; i < buf.length; i++) {
   point(i, HEIGHT - HEIGHT/2 - HEIGHT/2*amp*buf[(i - round(readPoint) + B/4 + B)%B]);
 }
 amp = decayFactor*amp;
 readPoint = (readPoint + (f*B/R))%B; // incremento dell'oscillatore
}
```

Solution to Exercise 4.2 (p. 38)
Quella che segue è la soluzione a Exercise 4.1 con i vincoli aggiunti agli estremi:

```
int B = 256; // lunghezza di tabella (risoluzione)
```

```
int R = 20;   // rate
float f = 1.1; // frequenza in cicli al secondo
float[] buf = new float[B];
int HEIGHT = 128; // altezza della finestra
float amp; // ampiezza della oscillazione
float readPoint; // posizione orizzontale della cresta
float decayFactor = 0.92; // fattore di decadimento dell'onda
float tempo;
float ampli=0;

void setup() {
 size(B,HEIGHT);
 stroke(255); strokeWeight(2);
 for (int i = 0; i < buf.length; i++) buf[i] = sin(2*PI*(float)i/buf.length);
}

void mousePressed() {
  tempo = millis();
}

void mouseReleased() {
  tempo = millis() - tempo; println(tempo);
  f = 500/tempo;
}

void draw() {
 background(0);

 if ((mouseX >= 0) && (mouseX < B) && (mousePressed)) {
   amp = 2*(HEIGHT/2 - mouseY)/float(HEIGHT);
   readPoint = mouseX;
 }
 for (int i = 0; i < buf.length; i++) {
   if (i < readPoint) ampli += 1/(float)readPoint;
   else ampli -= 1/(float)(B - readPoint);
   point(i, HEIGHT - HEIGHT/2 - HEIGHT/2*amp*ampli*buf[(i - round(readPoint) + B/4 + B)%B]);
 }
 ampli = 0;
 amp = decayFactor*amp;
 readPoint = (readPoint + (f*B/R))%B; // incremento dell'oscillatore
}
```

Solution to Exercise 4.4 (p. 40)
Basta sostituire `tabread~` con `tabread4~`, in questo modo facendo una lettura interpolata di ordine 3. In altri termini, l'interpolatore fa passare un polinomio di terzo grado (una cubica) per quattro punti contigui della tabella.
Solution to Exercise 4.5 (p. 41)

```
PFont font;
int B = 1000;
int INTERLINEA = 30;
int xpos=0, ypos=INTERLINEA;
char[] buffer = new char[B];
int OUT = 0;
int IN = 0;
int D = 0;
char carattere;
char inchar = '\0';

void setup() {
  size(800,800);
  font = loadFont("Courier-48.vlw");
  textFont(font, 32);
  frameRate(30);
}

void keyReleased() {
  inchar = key;
}

void draw() {
    if (inchar=='\n') {
      D+=10;
      IN = (OUT + D)%B;
      xpos = 0;
      ypos += INTERLINEA;
      inchar='\0';
      }
    else {
      buffer[IN] = inchar;
      carattere = buffer[OUT];
      text(carattere, xpos, ypos);
      xpos += textWidth(carattere);
      IN = (IN + 1)%B; OUT = (OUT + 1)%B;
    }
    inchar = '\0';
}
```

Chapter 5

Liste, pile e code[1]

5.1 Liste

Una istanza di classe può contenere, tra le sue variabili, anche un puntatore ad un oggetto della classe stessa. Questo consente di concatenare oggetti di una stessa classe, a formare strutture complesse quali liste o alberi. In particolare, una lista è ottenuta dalla concatenazione di **nodi**, ciascuno dei quali contiene un puntatore al nodo successivo, nonché un campo dati talvolta chiamato **cargo**.

Example 5.1: Attraversamento di una lista
Nel codice Processing che segue viene creata una lista di tre nodi concatenati. Quindi tale lista viene visitata per attraversamento e il cargo di ogni nodo viene stampato. Si notino le seguenti cose:

- La classe `Node` ha due tipi di costruttore, uno senza parametri ed uno con parametri. Quest'ultimo consente la creazione con inizializzazione del cargo e del puntatore al nodo successivo.
- C'è un metodo `toString` che converte il cargo in una stringa stampabile ogni volta che viene invocata la funzione print sul nodo.

```
void setup() {
  Node node3 = new Node (3, null);
  Node node2 = new Node (2, node3);
  Node node1 = new Node (1, node2);
  printList(node1);
}

void printList (Node list) {
    Node node = list;

    while (node != null) {
        print(node);
        node = node.next;
    }
    println();
}
```

[1] This content is available online at <http://cnx.org/content/m14556/1.5/>.

```
class Node {
  int cargo;
  Node next;

  Node() {
    cargo = 0;
    next = null;
  }

  Node (int cargo, Node next) {
    this.cargo = cargo;
    this.next = next;
  }

  String toString() { //rende l'oggetto di classe Node stampabile
    return cargo + " ";
  }
}
```

Così come gli array sono il supporto naturale dei cicli `for` e delle iterazioni sequenziali, così le liste sono il supporto naturale della algoritmica ricorsiva. Gli algoritmi ricorsivi sono definiti induttivamente stabilendo l'operazione per un elemento base (la base della ricorsione) e stabilendo come utilizzare il risultato dell'elaborazione sulla lista privata di tale elemento. Inoltre, bisogna assicurarsi che l'elaborazione così arrangiata in maniera "telescopica" raggiunga ad un certo punto una condizione di terminazione della ricorsione. Ciò non è sempre banale perché le catene di puntatori degli elementi di una lista potrebbero presentare dei **loop**, la cui visita procederebbe senza fine. Nell'esempio Example 5.1 (Attraversamento di una lista) si può introdurre una funzione di stampa dall'ultimo elemento al primo:

```
void printBackward (Node list) {
 if (list == null) return; // terminazione della ricorsione

 Node head = list;
 Node tail = list.next;

 printBackward(tail);
 print(head);   // base della ricorsione
}
```

Si può costruire una classe che incorpori l'intera lista e le operazioni che su di essa si intendono eseguire. Ad esempio, si può estendere Example 5.1 (Attraversamento di una lista) con l'introduzione di una classe `IntList` che supporta stampe per attraversamento diretto e inverso (ricorsivo). Si noti l'uso di un doppio metodo `printBackward`, uno con e uno senza argomenti, per consentire l'invocazione del metodo sull'oggetto lista senza necessità di passare un puntatore alla testa della lista stessa.

```
IntList lista;
```

```
void setup() {
  lista = new IntList();
  lista.addFirst(3);
  lista.addFirst(2);
  lista.addFirst(1);
  lista.printList();
  lista.printBackward();
}

class IntList {
 int length;
 Node head;

   IntList() {
     length = 0;
     head = null;
   }

   void addFirst (int i) {
     Node node = new Node(i, head);
     head = node;
     length++;
   }

   void printList () {
     Node node = head;

      while (node != null) {
        print(node);
        node = node.next;
      }
      println();
   }

  void printBackward() {
    printBackward(head);
    println();
  }

  void printBackward (Node head) {
   if (head == null) return; // terminazione della ricorsione

   Node tail = head.next;

   printBackward(tail);
   print(head);   // base della ricorsione
  }
 }
```

```
class Node {
  int cargo;
  Node next;

  Node() {
    cargo = 0;
    next = null;
  }

  Node (int cargo, Node next) {
    this.cargo = cargo;
    this.next = next;
  }

  String toString() { //rende l'oggetto di classe Node stampabile
    return cargo + " ";
  }
}
```

Questo è un esempio di classe **wrapper**, cioè di confezionamento object-oriented di una struttura dati. Le classi wrapper esistono per tutti i tipi primitivi di Java (`double`, `int`, ecc.) e consentono di creare e manipolare oggetti che contengono valori primitivi. Ad esempio, la classe wrapper `Integer` può essere interrogata in relazione al più piccolo e più grande valore rappresentabile mediante `Integer.MIN_VALUE` e `Integer.MAX_VALUE`, rispettivamente. Il valore numerico di un oggetto della classe `Integer` si ottiene con il metodo `intValue()`. Anche gli array in Java sono accessibili mediante una classe wrapper. La classe `Arrays`, accessibile importando `java.util.*`, consente di fare operazioni di test di uguaglianza, di ricerca, e di ordinamento. Ad esempio, il semplice codice

```
import java.util.*;

void setup() {
  int[] a = new int[10];
  int[] b = new int[10];

  for (int i=0; i<10; i++) {
    a[i] = i;
    b[i] = i;
  }
  println(Arrays.equals(a,b));
}
```

esegue un test di uguaglianza tra due array di interi.

Lettura

Si legga il capitolo 14 di How to Think Like a Computer Scientist[2]

[2] http://www.greenteapress.com/thinkapjava/

5.2 Pile

Lo **stack**, o pila, è una struttura dati di grande importanza. Ad esempio, essa è implicitamente coinvolta nell'esecuzione di codice ricorsivo, quale quello del metodo `printBackward`. In effetti, a runtime le chiamate a procedura sono sistemate, insieme ai loro parametri, una sopra all'altra nell'ordine in cui sono invocate. Nell'esempio, alla terza chiamata ricorsiva di `printBackward` ci si trova in cima allo stack una chiamata su una lista di un solo elemento. Ad uno stack si accede mediante due sole operazioni, la `push` (immissione) di un elemento in cima allo stack, e la `pop` (rimozione) di un elemento dalla cima dello stack. Una classe `Stack` è già disponibile in Java (e in Processing) dopo aver importato `java.util.*`. Essa funziona su oggetti della classe `Object`, che è la più generica delle classi Java, nonché superclasse di tutte.

Example 5.2: Attraversamento inverso di una lista
La visita in ordine inverso, dalla coda alla testa, degli elementi di una lista si può ottenere per via ricorsiva, come già visto in p. 46. Alternativamente, si può usare uno stack come struttura dati di appoggio.

```
import java.util.*;

IntList lista;
Stack pila;

void setup() {
  pila = new Stack();
  lista = new IntList();
  lista.addFirst(3);
  lista.addFirst(2);
  lista.addFirst(1);
  lista.printList();
  for (Node node = lista.head; node != null; node = node.next) {
   pila.push(node);
  }
  while (!pila.isEmpty()) {
    Node node = (Node) pila.pop();
    print(node);
  }
}

class IntList {
 ... omissis ...

class Node {
 ... omissis ...
```

Si può notare che

- gli elementi della lista sono inseriti nella pila nel loro ordine naturale, essendo così estraibili da questa in ordine inverso,
- esiste un metodo `isEmpty()` della classe `Stack` che consente di verificare la condizione di pila vuota,

- è necessario fare un type casting sull'oggetto restituito dalla `pop()`. Questo è un generico `Object` in quanto la classe `Stack` non è in grado di conoscere il tipo dei suoi elementi.
- in generale è sempre possibile convertire un algoritmo ricorsivo in uno non ricorsivo facendo uso di uno stack. In questo esempio la `pila` viene usata in luogo del run-time stack.

5.2.1 Valutazione di espressioni

Lo stack è anche la struttura dati che con maggior naturalezza supporta la valutazione di espressioni. Per essere più precisi, data una stringa contenente operandi e operatori in notazione postfissa[3] , una scansione lineare della stringa accompagnata dall'utilizzazione di uno stack di appoggio può consentire una facile valutazione della espressione. E' necessario inoltre individuare i **token** della stringa (operandi o operatori), separati da delimitatori che tipicamente sono caratteri di spaziatura. C'è una classe di `java.util`, chiamata `StringTokenizer`, che fa proprio questo. Ad esempio, la "tokenizzazione" di una espressione in notazione postfissa si può eseguire con:

```
import java.util.*;

StringTokenizer st = new StringTokenizer("11 22+33*", " +-*/", true);

while (st.hasMoreTokens())
  println(st.nextToken());
```

dove gli argomenti di `StringTokenizer()`, oltre a comprendere la stringa da analizzare, comprendono la lista dei delimitatori e un parametro boolean che indica che i separatori devono essere considerati anch'essi come token. La valutazione di espressioni su stringhe è un esempio semplice di parsing[4] .

Exercise 5.1 ***(Solution on p. 55.)***

Si costruisca un valutatore di espressioni in notazione postfissa con operatori di somma e moltiplicazione su numeri interi.

5.2.2 Lo stack delle trasformazioni geometriche

Un ambito di impiego molto importante della struttura dati a stack è la grafica interattiva. In particolare, le trasformazioni geometriche che si effettuano in OpenGL[5] avvengono per moltiplicazione matrice-vettore (p. 30), e le matrici sono memorizzate sullo stack delle trasformazioni[6]. Ciò consente di costruire modelli gerarchici di oggetti nello spazio. Nel momento in cui si fa `pushMatrix()` la cima della pila contiene la matrice di trasformazione geometrica che si sta applicando ad un particolare oggetto, o in altri termini lo stato corrente del sistema. Le trasformazioni applicate successivamente alla `pushMatrix()` vengono "dimenticate" nel momento in cui si fa una `popMatrix()`, in questo modo ripristinando lo stato precedente a queste ultime trasformazioni.

[3] http://en.wikipedia.org/wiki/Postfix_notation
[4] http://en.wikipedia.org/wiki/Parsing
[5] "Composizione Grafica in Processing": Section Pillole di OpenGL <http://cnx.org/content/m12665/latest/#pillole_di_opengl>
[6] "Composizione Grafica in Processing": Section Lo stack delle trasformazioni <http://cnx.org/content/m12665/latest/#stack>

Example 5.3: Geometria grafica costruttiva
Un semplice esempio di modello gerarchico è un braccio meccanico costituito da tre parallelepipedi: una base, un omero, e un avanbraccio. La collocazione dell'omero è fissa rispetto alla base, e di questa deve seguirne gli spostamenti. Allo stesso modo, l'avanbraccio è vincolato geometricamente all'omero. Il codice seguente disegna il braccio meccanico che è spostabile nel suo complesso mediante il mouse. Grazie all'uso dello stack, l'omero può ruotare indipendentemente dalla base restando ad essa vincolato. L'avanbraccio può ruotare indipendentemente dall'omero restando a questo vincolato.

```
int LAVANB = 70;
int LOMERO = 80;
int LBASE = 30;
float OMEROT = PI/3;
float AVANROT = PI/5;

void setup(){
          size(200, 200, P3D);
}

void avanbraccio(){
  pushMatrix();
  translate(0, LAVANB/2, 0);
  box(20, LAVANB, 20);
  popMatrix();
}

void omero(){
  pushMatrix();
  translate(0, LOMERO/2, 0);
  box(20, LOMERO, 20);
  popMatrix();
}

void draw(){
          background(0);
          translate(mouseX,mouseY);
          rotateY(PI/6);
          box(LBASE);
          translate(0, LBASE, 0);
          rotateX(OMEROT);
          omero();
          translate(0, LOMERO, 0);
          rotateX(AVANROT);
          avanbraccio();
}
```

Lettura
Si legga il capitolo 15 di How to Think Like a Computer Scientist[7]

5.3 Code

La struttura dati chiamata coda (**queue**) organizza una politica di accesso alle informazioni analoga a quella applicata idealmente quando si fa la fila presso uno sportello. Il primo cliente ad essere servito è il primo che si è presentato alla coda dello sportello. Una coda (e in realtà anche una pila) può essere realizzata mediante una lista concatenata, consentendo in questo modo di non porre limiti alla lunghezza della coda stessa. L'accesso alla lista che supporta la coda avviene mediante un puntatore di testa e uno di coda.

Example 5.4: Inserimento in coda ed estrazione dalla coda
Si può realizzare una coda mediante concatenazione di nodi e preparazione di due punti di accesso. Questo garantisce che l'accesso in inserimento ed in estrazione sia efficiente e a tempo costante. Nella realizzazione qui riportata la classe `Node` è atta a realizzare nodi con cargo di tipo Object generico. La presenza del metodo `toString()` garantirà che il nodo è stampabile.

```
Queue q;
Integer uno = new Integer(1);
Integer due = new Integer(2);
Integer tre = new Integer(3);

void setup() {
  q = new Queue();
  q.add(uno);
  q.add(due);
  q.add(tre);
  println(q.remove());
  println(q.remove());
  println(q.remove());
}

class Queue {
 int length;
 Node first, last;

   Queue() {
     length = 0;
     first = null;
     last = null;
   }

   boolean isEmpty() {
     return first == null;
   }

   void add (Object obj) {
     Node node = new Node(obj, null);
```

[7] http://www.greenteapress.com/thinkapjava/

```
    if (last != null) {
      last.next = node;
    }
    last = node;
    if (first == null) {
      first = last;
    }
    length +=1;
  }

  Object remove () {
    Node result = first;
    if (first != null) {
      first = first.next;
    }
    else {
      last = null;
    }
    length -= 1;
    return result;
  }
}



class Node {
  Object cargo;
  Node next;

  Node() {
    cargo = null;
    next = null;
  }

  Node (Object cargo, Node next) {
    this.cargo = cargo;
    this.next = next;
  }

  String toString() { //rende l'oggetto di classe Node stampabile
    return cargo + " ";
  }
}
```

Spesso si preferisce, per ragioni di semplicità realizzativa ed efficienza, realizzare la coda usando un array come struttura di supporto. In questo caso la crescita della coda è limitata dalla lunghezza dell'array. Questa realizzazione si chiama **buffer circolare** ed è impiegato, ad esempio, per la realizzazione di ritardi variabili (p. 40) nell'elaborazione dei segnali.

Lettura
Si legga il capitolo 16 di How to Think Like a Computer Scientist[8]

[8] http://www.greenteapress.com/thinkapjava/

Solutions to Exercises in Chapter 5

Solution to Exercise 5.1 (p. 50)
Nella soluzione che segue la stringa da analizzare è scritta in testa al codice. Si fa uso del metodo `equals()` della classe `String` e del metodo `parseInt()` della classe `Integer`. Si noti inoltre il matching di espressione regolare[9] supportato dalla classe `String`. In questo caso l'espressione regolare rappresenta una qualsiasi sequenza di cifre.

```
import java.util.*;

StringTokenizer st = new StringTokenizer("3 12 21 ++ 10*", " +*", true);
Stack pila = new Stack();
String tocco;
int op1=0, op2=0;

while (st.hasMoreTokens()) {
    tocco=st.nextToken();
    if (tocco.equals("+")) {
       if (!pila.isEmpty()) op2 = Integer.parseInt((String)pila.pop());
       if (!pila.isEmpty()) {
         op1 = Integer.parseInt((String)pila.pop());
         pila.push(Integer.toString(op1+op2));
       }
    }
    else if (tocco.equals("*")) {
       if (!pila.isEmpty()) op2 = Integer.parseInt((String)pila.pop());
       if (!pila.isEmpty()) {
         op1 = Integer.parseInt((String)pila.pop());
         pila.push(Integer.toString(op1*op2));
       }
    }
    else if (tocco.matches("\\d+")) { // token as a sequence of digits
      pila.push(tocco);
      println(tocco);
    }

    }
println(pila);
```

[9] http://en.wikipedia.org/wiki/Regular_expression

Chapter 6

Thread e I/O non bloccante[1]

6.1 Thread: definizione

Un thread (di controllo) è una lista di istruzioni eseguite sequenzialmente da un programma. I thread di un programma condividono uno stesso spazio di indirizzamento. Pur avendo stack e variabili locali separate, condividono le variabili globali. I thread sono "leggeri", nel senso che la creazione, distruzione e sincronizzazione sono relativamente economiche grazie alla condivisione dello spazio di indirizzamento. Le ragioni per organizzare un programma in un certo numero di thread possono essere molteplici:

- Certi programmi si scrivono più semplicemente, in special modo le collezioni di compiti debolmente connessi (cioè largamente indipendenti).
- I programmi interattivi risultano più efficienti laddove il servizio dell'input o il display dell'output sono organizzati in thread distinti.
- I programmi sono potenzialmente parallelizzabili su architetture multi-processore o multi-core.
- Il problema in esame richiede parti di programma in comunicazione asincrona tra loro.
- E' utile imporre una struttura modulare al codice.

I/O non bloccante
Una delle motivazioni forti per la programmazione concorrente mediante thread è l'ottenimento di servizi non bloccanti[2] per Input/Output. Quando l'applicazione ha necessità di effettuare un I/O, è opportuno che non si blocchi, in modo da consentire che altre operazioni non dipendenti da quell'I/O possano essere effettuate. Una tecnica di gestione dell'I/O non bloccante, utilizzata ad esempio nei microcontrollori[3] usati nelle board per il physical computing, è il **polling**, cioè la verifica ciclica dell'accadimento di eventi su un insieme di dispositivi di input. La ciclicità del polling è gestita da un timer. L'I/O non bloccante si può realizzare mediante i thread. La lettura di un certo dispositivo si può assegnare ad un certo thread il quale si blocca in attesa dei dati. Gli altri thread, che non dipendono dalla lettura del dato, possono però procedere in maniera concorrente.
Diagrammi di attivazione
Il codice Processing seguente invoca il metodo `stampa()` sull'oggetto `cl` della classe `Classe1`.

```
class Classe1 {
 void stampa() {
   for (int i=0; i< 100; i++) println("yep");
 }
```

[1] This content is available online at <http://cnx.org/content/m14565/1.7/>.
[2] http://en.wikipedia.org/wiki/Non-blocking_I/O
[3] http://itp.nyu.edu/~dbo3/cgi-bin/wiki.cgi?Microcontroller

```
}

void setup() {
 Classe1 cl = new Classe1();
 cl.stampa();
}
```

Il flusso di elaborazione si può rappresentare graficamente con il diagramma di attivazione di Figure 6.1, il quale presenta evidentemente un singolo thread.

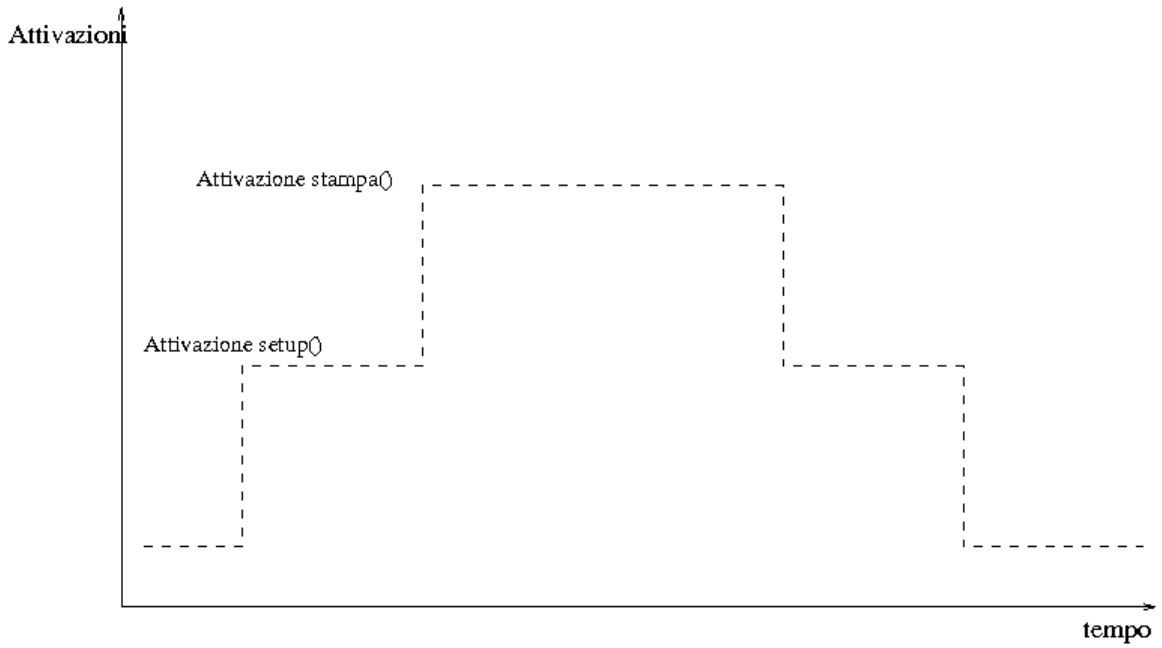

Figure 6.1: Diagramma di attivazione a thread singolo

Se invece la classe viene realizzata come **estensione** della classe `Thread`, allora è possibile procedere all'attivazione di un thread secondario mediante invocazione del metodo `start()`. Il codice va riscritto come

```
class Classe2 extends Thread{
 void run() {
   for (int i=0; i< 100; i++) println("yep");
 }
}

void setup() {
 Classe2 cl = new Classe2();
 cl.start();
}
```

Si noti che il metodo `stampa()` ora si chiama `run()`. Il metodo `start()` esiste nelle superclassi di `Classe2` (nella classe `Thread`, e si occupa dell'invocazione del metodo `run()`. Questa volta il flusso di elaborazione, riportato in Figure 6.2 presenta due thread.

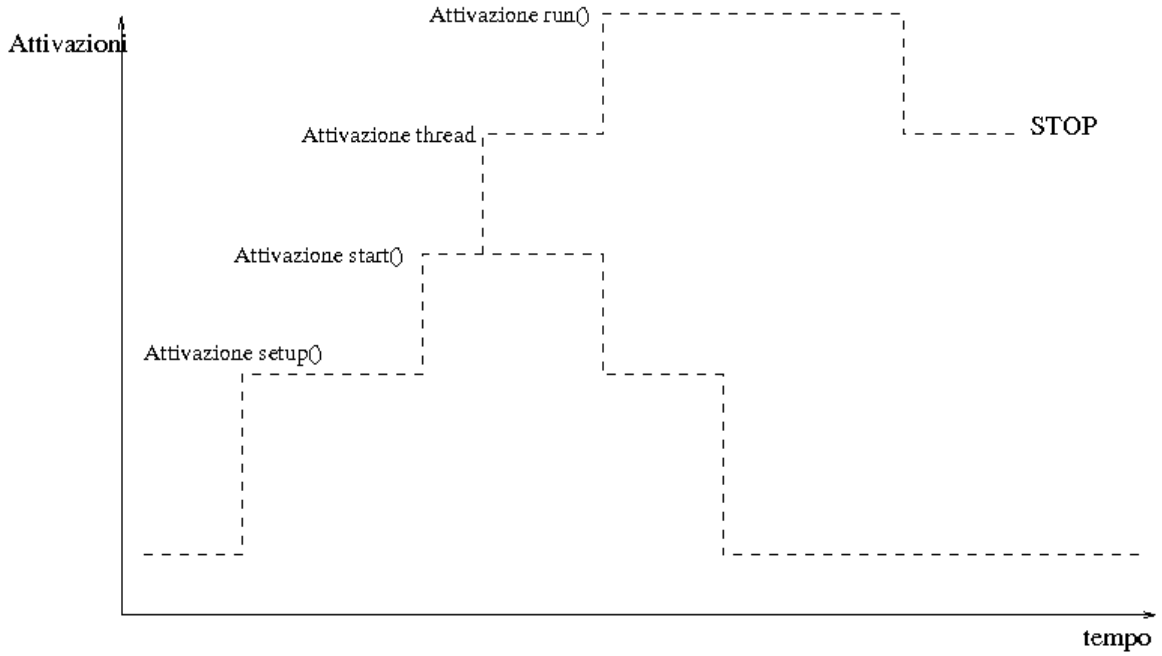

Figure 6.2: Diagramma di attivazione a doppio thread

NOTE: Esiste un secondo modo di dichiarare e attivare un thread[4] , mediante implementazione della interfaccia `Runnable` di Java. E' questa modalità che bisogna usare se si vuole attivare un nuovo thread su un oggetto di una classe che è già dichiarata come estensione di un'altra classe.

6.2 Thread: utilizzazione

Example 6.1: Thread temporizzati
La classe `Thread` possiede un metodo `sleep()` che la mette in stato **blocked** per un certo numero di millisecondi. Ciò consente di programmare in maniera compatta e agevole flussi di eventi tra loro indipendenti. Ad esempio, il codice che segue produce il disegno di ellissi di due colori diversi, e ogni colore corrisponde ad un diverso intervallo di tempo (i.e., 1000 e 1300 millisecondi) tra due eventi successivi di disegno.

```
class TimerThread extends Thread{
  int timediff; // quanto temporale
```

[4] http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Thread.html

```
  color col;

  TimerThread(color c, int td) {
   timediff = td;
   col = c;
  }

 void run() {
   while(true) {
     fill(col);
     try {
       ellipse(int(random(100)), int(random(100)),
         int(random(20)), int(random(20)));
       sleep(timediff);
     } catch (Exception e) {println("Exception in sleep");}
   }
 }
}

void setup() {
 TimerThread tt1 = new TimerThread(color(120,120,0),1000);
 TimerThread tt2 = new TimerThread(color(0,120,120),1300);
 tt1.start();
 tt2.start();
}

void draw() {
}
```

Si nota che il metodo `sleep()` di Java esige di essere invocato all'interno di un costrutto `try catch()`, cioè di una sezione di codice che consenta la cattura delle eccezioni. La gestione delle eccezioni[5] consente di affrontare delle condizioni che alterano il normale flusso di esecuzione di un programma. Nell'esempio, la `sleep()` può fallire e sollevare (`throw`) una eccezione che, in questo caso, è gestita mediante la mera scrittura di una messaggio sulla console.

Exercise 6.1

Si aggiunga al codice di Example 6.1 (Thread temporizzati) una classe che si occupa di ridipingere a intervalli regolari lo sfondo, in maniera da evitare la sovrapposizione delle ellissi.

6.3 I/O non bloccante basato su thread

Quando l'oggetto interattivo ha bisogno di leggere o scrivere da/su file, dispositivi, o network socket[6] , è opportuno che essa non si blocchi completamente in attesa dei dati. Con i thread, ciò viene risolto elegantemente attivando un thread separato che gestisce I/O asincrono, e si blocca laddove è necessario.

Example 6.2: Lettura di comandi da tastiera

Si supponga di dover disegnare in maniera automatica e ripetuta delle ellissi nella finestra grafica, e di voler controllare gli attributi di tali ellissi mediante parole chiave immesse da tastiera. Conviene

[5] http://en.wikipedia.org/wiki/Exception_handling
[6] http://en.wikipedia.org/wiki/Network_socket

separare il compito di lettura e interpretazione (parsing) del flusso di caratteri che viene dalla tastiera dal compito di produzione dell'output grafico. Ciò si può realizzare come segue, nel caso semplice in cui le parole accettate siano "rosso", "verde", e "blu" corrispondenti a diverse colorature delle ellissi prodotte.

```
StringBuffer stdin;
boolean linea;
color colore;

void keyReleased() {
  char c = key;
  if (c!='\n') {
    stdin.append(c);
  }
  else linea=true;
}

class ColorInput extends Thread {
  String results;
  char c;

  void run()  {
     while(true) {
       if (linea) {
            println(stdin);
            results=stdin.toString();
            stdin.setLength(0);
            linea = false;
            if (results.equals("rosso")) {
                colore = color(255, 0, 0);
            }
             if (results.equals("verde")) {
                colore = color(0, 255, 0);
            }
            if (results.equals("blu")) {
                colore = color(0, 0, 255);
            }
         }
        try {
        sleep(5); // to relief the cpu from active waiting
        } catch (Exception e) {println("Exception in sleep");}
       }
    }
  }

class TimerThread extends Thread{
  int timediff; // quanto temporale

  TimerThread(int td) {
   timediff = td;
```

```
  }

 void run() {
   while(true) {
     try {
       fill(colore);
       ellipse(int(random(100)), int(random(100)),
         int(random(20)), int(random(20)));
       sleep(timediff);
     } catch (Exception e) {println("Exception in sleep");}
   }
 }
}

void setup() {
 stdin = new StringBuffer();
 TimerThread tt1 = new TimerThread(100);
 ColorInput ci = new ColorInput();
 ci.start();
 tt1.start();
}

void draw() {
}
```

Sono presenti, in questo caso, due diverse estensioni della classe `Thread`. La prima estensione fa una attesa attiva di linee di testo, impostando il colore ogniqualvolta viene rilevata una linea di testo contenente una delle tre parole chiave riconosciute. L'invocazione di `sleep(5)` rende questa attesa attiva meno onerosa per la CPU. L'altro thread, invece, si occupa di disegnare dieci ellissi al secondo. In Processing è difficile realizzare un input bloccante da tastiera, in quanto non è accessibile direttamente lo stream `System.in`, sul quale in Java si può normalmente applicare lettura bufferizzata bloccante.

NOTE: Per una introduzione all'I/O in Java si veda il Java Java IO Tutorial[7] .

Viceversa, Processing invita ad una programmazione event-based[8] fornendo gli **event handler** `keyReleased()`, `keyPressed()`, e `keyTyped()`. E' possibile però fare un input bloccante di una linea da file di testo con un codice del tipo

```
try {
   BufferedReader stdiin = createReader("nomefile");
   println(stdiin.readLine());
 }catch(Exception e){}
```

dove `createReader()` è una funzione di Processing che crea un `BufferedReader` object da un file o da una URL. Essa consente una leggera semplificazione rispetto al codice Java

[7] http://tutorials.jenkov.com/java-io/index.html
[8] http://en.wikipedia.org/wiki/Event-based_programming

```
try {
   FileReader is = new FileReader("nomefile");
   BufferedReader stdiin = new BufferedReader( is );
   println(stdiin.readLine());
 }catch(Exception e){}
```

Per semplificare la lettura da file di testo, locali o remoti, Processing mette a disposizione la funzione `loadStrings()`, che carica tutte le linee di file di testo in un array di tipo `String[]`. Questo metodo può essere utile se il file non è troppo grande o dinamicamente variabile.

Exercise 6.2

Il codice che segue effettua la lettura periodica di una linea di testo da un sito generatore di testo. Lo si estenda aggiungendo un thread che visualizza queste frasi sulla finestra grafica con animazione tipografica.

```
class sentenceReader extends Thread{
  int timediff; // quanto temporale

  sentenceReader(int td) {
   timediff = td;
  }

 void run() {
    while(true) {
      try {
       BufferedReader stdiin = createReader("http://www.essl.at/cgi-bin/swrap/cgis/lexikon-orakel.pl");
       for (int i=0; i<13; i++) stdiin.readLine();
       println(stdiin.readLine());
       sleep(timediff);
      }catch(Exception e){}
    }
  }
}

void setup() {
 sentenceReader st = new sentenceReader(2000);
 st.start();
}

void draw() {
}
```

6.4 Sincronizzazione

Si supponga di dover estendere la lettura di comandi da tastiera (p. 60) con due thread che prendono comandi da `stdin`. I comandi possono essere consumati dall'uno o dall'altro dei thread, ma uno stesso comando non può essere consumato da entrambi. In questo caso si possono presentare problemi di race condition[9] , cioè configurazioni di codice concorrente che danno luogo a risultati dipendenti dalla sequenza di **scheduling** attribuita ai vari thread. In particolare, la **sezione critica**

```
    results=stdin.toString();
            stdin.setLength(0);
            linea = false;
```

può dare luogo a comportamenti inconsistenti se interrotta dallo **scheduler** per passare il controllo dall'uno all'altro thread. Questi problemi si possono risolvere imponendo la non interrompibilità delle sezioni critiche di codice, per mezzo della parola chiave `synchronized`. Nella fattispecie, si può introdurre una classe con metodi sincronizzati:

```
class IO extends Thread {
 String results;

 synchronized  String acquire() {
   results=stdin.toString();
   linea = false;
   stdin.setLength(0);
   return(results);
 }
}
```

I metodi invocati su un oggetto di questa classe non sono interrompibili da metodi invocati sullo stesso oggetto. In ambito di transazioni bancarie, per esempio, un metodo `prelievo()` dovrà essere sincronizzato per evitare inconsistenze in presenza di prelievi multipli, come possono essere effettuati da co-titolari dello stesso conto. La sincronizzazione di thread è un argomento complesso, per l'approfondimento del quale esistono libri specializzati[10] .

Exercise 6.3 ***(Solution on p. 65.)***

Si estenda la lettura di comandi da tastiera (p. 60) con due thread che prendono comandi da `stdin`.

6.5 Reference

Il capitolo 9 del libro[11] *Visualizing Data – Ben Fry; O'Really 2007* fornisce esempi di acquisizione asincrona di dati da file e da siti remoti, con l'utilizzazione di thread e di sincronizzazione.

[9] http://en.wikipedia.org/wiki/Race_condition
[10] http://www.javaworld.com/javaworld/jw-12-2000/jw-1215-threadbooks.html
[11] http://www.oreilly.com/catalog/9780596514556/

Solutions to Exercises in Chapter 6

Solution to Exercise 6.3 (p. 64)
Nel codice seguente si noti l'uso della `getName()` per stampare il nome del thread interessato.

```
StringBuffer stdin;
boolean linea;
color colore;
IO io = new IO();

void keyReleased() {
  char c = key;
  if (c!='\n') {
    stdin.append(c);
  }
  else linea=true;
}

class IO extends Thread {
 String results;

 synchronized  String acquire() {
   results=stdin.toString();
   linea = false;
   stdin.setLength(0);
   return(results);
 }
}

class ColorInput extends Thread {
  String results;
  char c;

  void ColorInput (IO inout) {
    io = inout;
  }
  void run()  {
     while(true) {
       if (linea) {
            results=io.acquire();
            println(this.getName() + results);
            if (results.equals("rosso")) {
                colore = color(255, 0, 0);
            }
             if (results.equals("verde")) {
                colore = color(0, 255, 0);
            }
            if (results.equals("blu")) {
                colore = color(0, 0, 255);
            }
```

```
          }
         try {
         sleep(2); // to relief the cpu from active waiting
         } catch (Exception e) {println("Exception in sleep");}
        }
    }
  }

class TimerThread extends Thread{
  int timediff; // quanto temporale

  TimerThread(int td) {
   timediff = td;
  }

 void run() {
   while(true) {
     try {
       fill(colore);
       ellipse(int(random(100)), int(random(100)),
         int(random(20)), int(random(20)));
       sleep(timediff);
     } catch (Exception e) {println("Exception in sleep");}
   }
 }
}

void setup() {
 stdin = new StringBuffer();
 TimerThread tt1 = new TimerThread(100);
 ColorInput ci = new ColorInput();
 ColorInput ci2 = new ColorInput();
 ci.start(); ci2.start();
 tt1.start();
}

void draw() {
}
```

Chapter 7

Segnali nel tempo: soglie e filtri[1]

7.1 Isteresi e soglie

L'isteresi[2] è quella proprietà dei sistemi che li fa reagire con un certo ritardo alle forze ad essi applicate. Ad esempio, all'atto di sedersi su una poltrona imbottita, la relazione tra forza e compressione del cuscino può essere di forma isteretica[3] , ed evidenza di ciò si può avere dall'impronta lasciata nel momento in cui ci si rialza. Nel design dell'interazione, l'isteresi viene introdotta deliberatamente per ritardare le azioni del sistema rispetto all'accadere di eventi. Per esempio, il click del mouse in una certa area può attivare un menu, la cui scomparsa viene però ritardata rispetto all'uscita da quella regione, in modo da consentire all'utente di esplorare le voci del menu anche con percorsi del mouse imperfetti. Nei regolatori automatici, quale è ad esempio il termostato per la regolazione di temperatura, è spesso presente un elemento isteretico per evitare commutazioni spurie, ad esempio dovute ai rimbalzi degli interruttori. Il termostato, in particolare, attiva il bruciatore quando la temperatura scende al di sotto di una soglia (**threshold**) A, ma non lo spegne fintantoché la temperatura non raggiunge una seconda soglia $B > A$. Quindi, l'accensione e spegnimento del bruciatore dipendono dalla storia del segnale di temperatura. Ciò previene sequenze rapide di accensione/spegnimento per oscillazioni di temperatura intorno alla soglia. Nella sua forma più stilizzata[4] , un sistema con isteresi è bistabile. Cioè caratterizzato da due stati stabili. Quindi, per programmare un interruttore con isteresi è sufficiente utilizzare una variabile `boolean acceso` e due soglie `basso` e `alto`:

```
if (acceso) {
   if (signal < basso) {
      // operazioni di spegnimento
      acceso = false;
   }
}
else //spento
   if (signal > alto) {
      // operazioni di accensione
      acceso = true;
   }
```

[1] This content is available online at <http://cnx.org/content/m14574/1.3/>.
[2] http://en.wikipedia.org/wiki/Hysteresis
[3] http://upload.wikimedia.org/wikipedia/commons/2/2a/Hysteresiscurve.png
[4] http://upload.wikimedia.org/wikipedia/commons/a/aa/Hysteresis_sharp_curve.svg

Exercise 7.1 *(Solution on p. 70.)*

Si scriva un programma Processing che disegni la curva della pressione acustica catturata dal microfono. A basse pressioni la curva sia disegnata in verde. Quando la pressione acustica supera una soglia superiore la curva deve diventare di colore rosso, e tornare ad essere verde quando torna al di sotto di una soglia inferiore.

Exercise 7.2 *(Solution on p. 71.)*

Si modifichi il codice di Exercise 7.1 in modo da ottenere la curva di livello di un noise gate[5] . Quando il livello scende al di sotto della soglia verde, si applica uno smorzamento che gradualmente forza il segnale ad essere inudibile. Quando il livello supera la soglia rossa, si applica una amplificazione che riporta il segnale al livello originale.

Gli esempi di sogliatura, isteresi, e **gating** visti nel caso dell'elaborazione del segnale audio sono di utilità generale per qualsiasi flusso di segnale proveniente da un sensore e, in particolare, anche nell'elaborazione di segnale video.

7.2 Filtri

Nella maggior parte delle applicazioni che trattano flussi di segnale provenienti da sensori, c'è la necessità di addolcire (**smoothing**) il profilo dei segnali stessi, in modo da evitare picchi improvvisi e rumore a rapida variabilità. Ciò si può fare con i filtri[6]. Il più semplice filtro di smoothing è il filtro di media [7], il quale restituisce ad ogni istante il valor medio tra due campioni temporalmente contigui del segnale di ingresso. Interpretando i segnali nel dominio delle frequenze, l'operazione di smoothing corrisponde ad un filtraggio passa-basso, cioè ad una attenuazione delle frequenze elevate. Infatti, le variazioni repentine corrispondono a componenti di frequenza elevata. Le capacità di smoothing del filtro di media sono limitate, e per ottenere degli smoothing più decisi (maggiore attenuazione delle alte frequenze) bisogna fare medie pesate di un numero maggiore di campioni del segnale di ingresso. Un filtro molto utile allo scopo è il filtro del secondo ordine simmetrico[8]. Qualunque sia l'ordine del filtro, la sua realizzazione come media pesata di campioni del segnale di ingresso può essere effettuata convenientemente appoggiandosi alla realizzazione a buffer circolare (p. 40). In questo caso, la lettura avverrà da un certo numero di celle contigue a quella puntata da `OUT`, queste letture saranno pesate per i coefficienti del filtro, e il puntatore `IN` coinciderà con il puntatore `OUT`.

Exercise 7.3 *(Solution on p. 72.)*

Si modifichi il codice di Exercise 7.1 pre-filtrando il segnale di livello con un filtro del secondo ordine simmetrico.

Come si può dedurre dalla sperimentazione proposta in Exercise 7.3, prendendo medie pesate di pochi campioni di segnale l'effetto di smoothing non è macroscopico. Per avere un filtraggio più radicale bisogna usare filtri di ordine elevato (elaborazione di molti campioni di segnale) ovvero ricorrere a filtri ricorsivi.

Un filtro è un oggetto che, preso un segnale di ingresso, produce un segnale di uscita. Quando la produzione di un campione del segnale di uscita si ottiene elaborando campioni precedenti del segnale di uscita stesso (oltre che del segnale di ingresso) si dice che il filtro è ricorsivo[9]. La ricorsività consente filtraggi radicali a complessità modeste. Ad esempio, il filtro del primo ordine rappresentato dall'equazione alle differenze

$$y(n) = ay(n-1) - 1x(n) \tag{7.1}$$

è molto efficace come filtro di smoothing. E' importante mantenere la stabilità del filtro, cioè evitare che un segnale di ingresso limitato produca un'uscita illimitata. Nel caso del filtro ricorsivo del primo ordine, ciò è garantito se $|a| < 1$. Lo smoothing del segnale è giocoforza accompagnato da una certa latenza. Un segnale a gradino presentato in ingresso provoca in uscita un segnale che raggiunge asintoticamente il livello

[5] http://www.doctorproaudio.com/doctor/temas/dynamics-processors-noisegates_en.shtml
[6] "Signal Processing in Processing: Filtri Elementari" <http://cnx.org/content/m12827/latest/>
[7] "Signal Processing in Processing: Filtri Elementari" <http://cnx.org/content/m12827/latest/#averagingp>
[8] "Signal Processing in Processing: Filtri Elementari", (3) <http://cnx.org/content/m12827/latest/#convfir2>
[9] "Signal Processing in Processing: Filtri Elementari" <http://cnx.org/content/m12827/latest/#recursivediffeq>

del gradino. Nel caso del filtro ricorsivo del primo ordine, la transizione è tanto più lenta quanto più a è prossimo a 1. Per la precisione, il numero di campioni necessari per raggiungere il 99 per cento del valore finale è pari a $n = \frac{-2}{\log a}$. Ad esempio, con $a = 0.9$ la transizione dura circa 44 campioni. Naturalmente, la relazione $a = 10^{\frac{-2}{n}}$ consente di ricavare il valore del coefficiente che consente una transizione in n campioni. Tale formula è stata usata anche per impostare i tempi di apertura e chiusura del noise gate (Exercise 7.2).

Exercise 7.4 ***(Solution on p. 74.)***

Si modifichi il codice di Exercise 7.1 pre-filtrando il segnale di livello con un filtro ricorsivo del primo ordine.

Particolare importanza rivestono i filtri ricorsivi del secondo ordine[10], i quali fungono da modello per la realizzazioni di risonanze. Una risonanza elementare si manifesta come oscillazione smorzata. Se i coefficienti del filtro del secondo ordine sono opportunamente scelti in modo da produrre una risposta ai limiti della stabilità, quello che si ottiene è una realizzazione dell'oscillatore, alternativa a quelle proposte in oscillazioni, ritardi, e fluttuazioni del tempo (Chapter 4).

[10] "Signal Processing in Processing: Filtri Elementari" <http://cnx.org/content/m12827/latest/#iir2>

Solutions to Exercises in Chapter 7

Solution to Exercise 7.1 (p. 67)
Una soluzione che fa uso della libreria audio Minim[11] è la seguente:

```
    import ddf.minim.*;

AudioInput in;
int WIDTH=400, HEIGHT=200;
int x;
int level, prevLevel;
color colorLine=color(0,255,0);
boolean acceso=false;
int basso=80;
int alto=120;
Minim minim;

void setup()
{
  size(WIDTH, HEIGHT);
  minim = new Minim(this);
  // get a line-in from Minim: mono, 512 sample buffer
  // default sampling rate is 44100, default bit-depth is 16
  in = minim.getLineIn(Minim.MONO, 512);

}

void draw()
{
  if (x==0) {
    background(0);
    stroke(255,0,0); line(0,HEIGHT-alto,WIDTH,HEIGHT-alto);
    stroke(0,255,0); line(0,HEIGHT-basso,WIDTH,HEIGHT-basso);
  }
  stroke(colorLine);
  level = int(in.left.level()*HEIGHT*10);
  line(x, HEIGHT - prevLevel, (x+1)%WIDTH, HEIGHT - level);
  x = (x+1)%WIDTH;
  prevLevel = level;
  if (acceso) {
     if (level < basso) {
                colorLine = color(0, 255, 0);
        acceso = false;
     }
    }
  else //spento
            {
     if (level > alto) {
        colorLine = color(255, 0, 0);
        acceso = true;
     }
```

[11] http://code.compartmental.net/tools/minim

```
  }
}

void stop()
{
  // always stop Minim
  minim.stop();
  super.stop();
}
```

Solution to Exercise 7.2 (p. 68)

Nella soluzione seguente il segnale viene abbattuto, quando scende al di sotto della soglia inferiore, mediante moltiplicazioni successive per un coefficiente `gDown`, il cui valore (sempre minore di uno) viene impostato in base al tempo desiderato di smorzamento ed al frame rate. Lo smorzamento si considera esaurito quando il segnale di ingresso viene ad essere moltiplicato per 0.01. Analogamente, al superamento della soglia superiore, il moltiplicatore viene riportato a uno mediante moltiplicazioni successive per il coefficiente `gUp`, di valore sempre maggiore di uno.

```
    import ddf.minim.*;

AudioInput in;
int WIDTH=400, HEIGHT=200;
int x;
int level, prevLevel, levelG;
color colorLine=color(0,255,0);
boolean acceso=false;
int basso=30;
int alto=50;
float damp=1.0;
float gain = 1.0;
Minim minim;

float decayTime = 10.0; // in seconds
float raiseTime = 2.0;
float gDown = pow(10,-2/frameRate/decayTime);
float gUp = pow(10,2/frameRate/raiseTime);


void setup()
{
  println("gDown = " + gDown + "gUp = " + gUp);
  size(WIDTH, HEIGHT);
  minim = new Minim(this);
  // get a line-in from Minim: mono, 512 sample buffer
  // default sampling rate is 44100, default bit-depth is 16
  in = minim.getLineIn(Minim.MONO, 512);

}

void draw()
{
```

```
  if (x==0) {
    background(0);
    stroke(255,0,0); line(0,HEIGHT-alto,WIDTH,HEIGHT-alto);
    stroke(0,255,0); line(0,HEIGHT-basso,WIDTH,HEIGHT-basso);
  }
  stroke(colorLine);
  level = int(in.left.level()*HEIGHT);
  gain = gain*damp;
  //println(gain);
  if (gain > 1.0) gain = 1.0;
  if (gain < 0.01) gain = 0.01;
  levelG = int(level*gain);
  line(x, HEIGHT - prevLevel, (x+1)%WIDTH, HEIGHT - levelG);
  x = (x+1)%WIDTH;
  prevLevel = levelG;
  if (acceso) {
     if (level < basso) {
                colorLine = color(0, 255, 0);
                damp = gDown;
        acceso = false;
     }
    }
  else //spento
            {
     if (level > alto) {
        colorLine = color(255, 0, 0);
                damp = gUp;
        acceso = true;
     }
  }
}

void stop()
{
  // always stop Minim
  minim.stop();
  super.stop();
}
```

Solution to Exercise 7.3 (p. 68)

Nel codice qui proposto si possono sperimentare diversi livelli di smoothing, variando i valori di `a0` e `a1`. Più questi valori sono vicini, maggiore è l'entità dello smoothing. Per non alterare con il filtraggio l'ampiezza generale del segnale è opportuno che il doppio di `a0`, sommato ad `a1`, dia come risultato 1.

```
    import ddf.minim.*;

AudioInput in;
int WIDTH=400, HEIGHT=200;
int x;
int level, prevLevel;
color colorLine=color(0,255,0);
```

```
boolean acceso=false;
int basso=80;
int alto=120;
Minim minim;

float GAIN = 10;
int BUFLEN = 5;
float[] buffer = new float[BUFLEN];
int bufPoint=2, bufPoint1=1, bufPoint2=0;
float a0 = 0.3, a1 = 0.4; //smoothing
//float a0 = 0, a1 = 1; // no smoothing
void setup()
{
  size(WIDTH, HEIGHT);
  minim = new Minim(this);
  // get a line-in from Minim: mono, 512 sample buffer
  // default sampling rate is 44100, default bit-depth is 16
  in = minim.getLineIn(Minim.MONO, 512);

}

void draw()
{
  if (x==0) {
    background(0);
    stroke(255,0,0); line(0,HEIGHT-alto,WIDTH,HEIGHT-alto);
    stroke(0,255,0); line(0,HEIGHT-basso,WIDTH,HEIGHT-basso);
  }
  stroke(colorLine);
  buffer[bufPoint] = in.left.level();
  level = int((a0*buffer[bufPoint] + a1*buffer[bufPoint1] + a0*buffer[bufPoint2])*GAIN*HEIGHT);
  bufPoint = (bufPoint+1)%BUFLEN;   bufPoint1 = (bufPoint1+1)%BUFLEN;   bufPoint2 = (bufPoint2+1)%BUFLE
  line(x, HEIGHT - prevLevel, (x+1)%WIDTH, HEIGHT - level);
  x = (x+1)%WIDTH;
  prevLevel = level;
  if (acceso) {
     if (level < basso) {
                colorLine = color(0, 255, 0);
        acceso = false;
     }
    }
  else //spento
            {
     if (level > alto) {
        colorLine = color(255, 0, 0);
        acceso = true;
     }
  }
}

void stop()
```

```
{
  // always stop Minim
  minim.stop();
  super.stop();
}
```

Solution to Exercise 7.4 (p. 69)
Si provino diversi valori di a nel codice qui proposto, mantenendo il vincolo di stabilità.

```
    import ddf.minim.*;

AudioInput in;
int WIDTH=400, HEIGHT=200;
int x;
int level, prevLevel;
color colorLine=color(0,255,0);
boolean acceso=false;
int basso=80;
int alto=120;
Minim minim;

float GAIN = 10;
float currentLev;
float prevLev = 0;
float a = 0.95; // the closer to 1, the smoother the signal

void setup()
{
  size(WIDTH, HEIGHT);
  minim = new Minim(this);
  // get a line-in from Minim: mono, 512 sample buffer
  // default sampling rate is 44100, default bit-depth is 16
  in = minim.getLineIn(Minim.MONO, 512);

}

void draw()
{
  if (x==0) {
    background(0);
    stroke(255,0,0); line(0,HEIGHT-alto,WIDTH,HEIGHT-alto);
    stroke(0,255,0); line(0,HEIGHT-basso,WIDTH,HEIGHT-basso);
  }
  stroke(colorLine);
  currentLev = a*prevLev + (1 - a)*in.left.level();
  prevLev = currentLev;
  level = int(currentLev*GAIN*HEIGHT);
  line(x, HEIGHT - prevLevel, (x+1)%WIDTH, HEIGHT - level);
  x = (x+1)%WIDTH;
  prevLevel = level;
  if (acceso) {
     if (level < basso) {
```

```
                colorLine = color(0, 255, 0);
        acceso = false;
     }
    }
  else //spento
            {
     if (level > alto) {
        colorLine = color(255, 0, 0);
        acceso = true;
     }
  }
}

void stop()
{
  // always stop Minim
  minim.stop();
  super.stop();
}
```

Chapter 8

Programmazione Visuale[1]

8.1 Programmazione Visuale

Un linguaggio di programmazione visuale[2] permette di specificare programmi mediante manipolazione di elementi grafici. I linguaggi più usati in ambito artistico e di produzione multimediale (tra questi Pure Data[3] , MAX/MSP[4] , vvvv[5] , e Quartz Composer[6] sono basati sul paradigma di calcolo dataflow[7] , nel quale i dati (piuttosto che le sequenze di operazioni) sono al centro dell'attenzione. Le operazioni sono "scatole nere" che elaborano l'input, non appena questo è disponibile, producendo un output che può essere dato in input ad altre scatole di elaborazione. Un programma dataflow assomiglia ad una rete di catene di montaggio, esplicita in maniera intrinseca il parallelismo, e non nasconde uno stato. Oltre che prestarsi alla parallelizzazione automatica, i programmi dataflow sono in un certo senso "auto-documentati" e si prestano al debugging mediante sonde.

Pure Data[8] è un linguaggio di programmazione visuale inizialmente concepito da Miller Puckette[9] per la computer music in tempo reale, poi esteso ad opera di una comunità di programmatori ed oggi diffuso anche tra artisti visuali, performer e interaction designer. Pure Data è un software libero, ed ha uno stretto grado di parentela con MAX/MSP[10] . Essendo nato per elaborare e controllare segnale audio, Pure Data supporta la comunicazione dei dati da un operatore all'altro a due rate: il sample rate che per default è di 44100 campioni al secondo, ed il control rate ad un sessantaquattresimo del sample rate. Più precisamente, gli operatori che elaborano segnale audio (caratterizzati dal suffisso ∼) prendono campioni di ingresso in maniera sincrona ad audio rate e allo stesso rate producono segnali di uscita. Tutti gli altri operatori elaborano dati non appena questi sono disponibili (dataflow) ad un rate massimo di circa 690 Hz (per default, un sessantaquattresimo del sample rate). Tutti i calcoli sono effettuati su numeri floating-point a 32 bit. La documentazione di Pure Data è disponibile dalla voce `Help` del suo menu, oppure è consultabile online[11] . Questo modulo è parzialmente basato su tale documentazione e sulle lezioni di Gary Scavone[12] .

I programmi Pure Data sono chiamati **patch** e si presentano come grafi di elaborazione di flussi di dati. Ogni programma ha una finestra principale e può avere un numero arbitrario di sottofinestre. Il software

[1] This content is available online at <http://cnx.org/content/m14602/1.7/>.
[2] http://en.wikipedia.org/wiki/Visual_programming
[3] http://en.wikipedia.org/wiki/Pure_Data
[4] http://en.wikipedia.org/wiki/Max/MSP
[5] http://vvvv.org/
[6] http://en.wikipedia.org/wiki/Quartz_Composer
[7] http://en.wikipedia.org/wiki/Dataflow_language
[8] http://en.wikipedia.org/wiki/Pure_Data
[9] http://crca.ucsd.edu/∼msp/
[10] http://en.wikipedia.org/wiki/Max_%28software%29
[11] http://crca.ucsd.edu/∼msp/Pd_documentation/
[12] http://www.music.mcgill.ca/∼gary/306/

Pure Data ha una interfaccia modale[13] , in quanto ci sono due modi distinti di operazione: run mode e edit mode. Si passa dall'uno all'altro con `command-E` e il feedback sul modo corrente è dato dalla forma del puntatore del mouse. I blocchi che si possono connettere in forma di grafo di dataflow sono di tipo: object, message, atom (number o symbol), GUI, e comment.

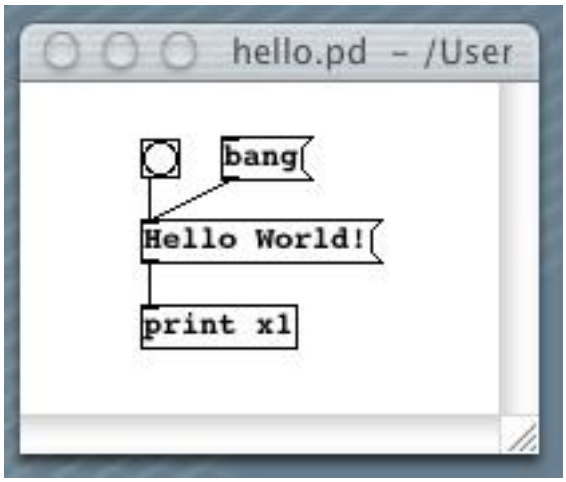

Figure 8.1

In Figure 8.1 è illustrato un semplicissimo patch[14] che stampa sulla consolle la scritta `x1: Hello World`. Esso contiene due oggetti message (lato destro concavo) ed un oggetto object (rettangolo con comando `print x1`). L'oggetto object rapresenta un comando con eventuali parametri di default. Il quadrato con cerchio all'interno è uno speciale messaggio senza contenuto, chiamato **bang**, che invia un evento ai blocchi ad esso collegati.

[13] http://en.wikipedia.org/wiki/Mode_%28computer_interface%29
[14] See the file at <http://cnx.org/content/m14602/latest/./hello.pd>

8.1.1 Idiosincrasie grafiche

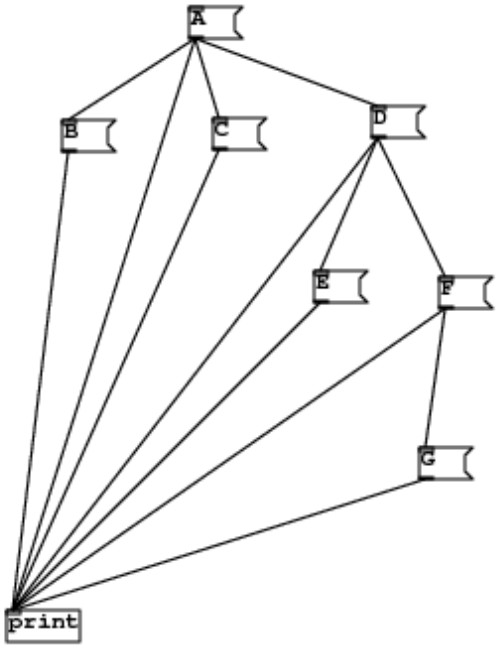

Figure 8.2

Ogniqualvolta un messaggio è inviato ad un oggetto, questi può diventare mittente a sua volta verso altri oggetti. In altri termini, si configura un albero di ricezioni di messaggi. Secondo il manuale di Pure Data [15] l'ordine di esecuzione utilizzato è depth-first[16] , cioè ogni ramo è esplorato fino alle foglie prima di procedere con gli altri rami. L'ordine di esecuzione tra nodi allo stesso livello dell'albero sarebbe invece da considerarsi indeterminato. In realtà, provando ad eseguire l'esempio[17] di Figure 8.2 e ridisponendo le connessioni tra i nodi dell'albero con ordini diversi ci si accorge che non si può nemmeno contare sull'ordine depth-first, in quanto esso interferisce con l'ordine di immissione delle connessioni. In Max/MSP la situazione è ancora peggiore perché muovendo messaggi e bang in posti diversi della finestra, a parità di configurazione topologica, si ottengono sequenze di attivazione diverse. L'elaborazione di tipo dataflow può dare luogo a loop non computabili, quale è quello riportato in Figure 8.3. Pure Data, in questo caso, riporta un messaggio di "stack overflow" sulla consolle. Per rendere il loop computabile è sufficiente inserire un elemento di disaccoppiamento quale è un `pipe` (si veda p. 85). Per quanto sia piccolo il parametro ritardo di questo pipe

[15] http://crca.ucsd.edu/~msp/Pd_documentation/x2.htm#s3.2
[16] http://en.wikipedia.org/wiki/Depth-first_search
[17] See the file at <http://cnx.org/content/m14602/latest/./depth-first.pd>

(al limite anche nullo), esso terrà memoria del dato in ingresso per consumarlo al ciclo successivo di calcolo.

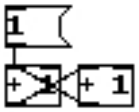

Figure 8.3

Nella maggior parte degli oggetti, la inlet di sinistra è **hot**, nel senso che messaggi che ad essa arrivano scatenano messaggi in output. E' spesso desiderabile inviare messaggi a due o più inlet di un oggetto, ma il risultato dell'operazione può dipendere dall'ordine con cui si sono fatte le connessioni. Ciò mina l'efficacia semantica del programma visuale, in quanto potremmo avere due patch apparentemente identiche che però si comportano diversamente perché costruite con un ordine diverso. Ad esempio, si provi a fare la somma di un numero con sè stesso secondo la Figure 8.4.

Figure 8.4

La sequenza di creazione delle connessioni determina la correttezza o meno del risultato. Per imporre chiarezza semantica al patch è opportuno inserire un oggetto `trigger` che forza l'ordine di attivazione delle outlet da destra a sinistra, come indicato in Figure 8.5.

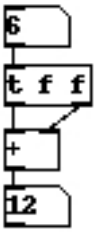

Figure 8.5

8.1.2 Elementi di GUI

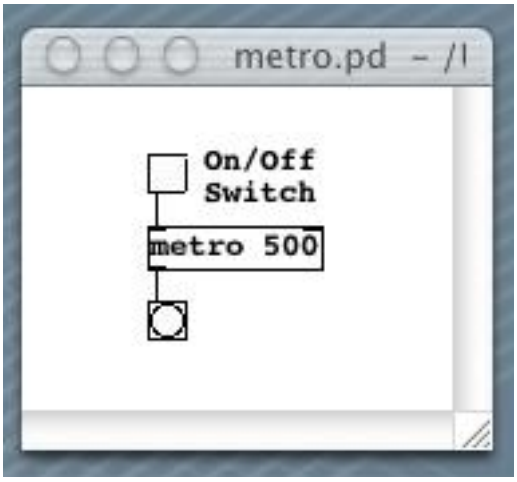

Figure 8.6

In Figure 8.6 è illustrata l'utilizzazione di un generatore periodico di bang[18]. Si noti il parametro di `metro` che stabilisce l'intervallo (di 500 millisecondi) tra due bang. Il quadrato in alto è un elemento di GUI detto toggle switch, in pratica un interruttore.

[18] See the file at <http://cnx.org/content/m14602/latest/./metro.pd>

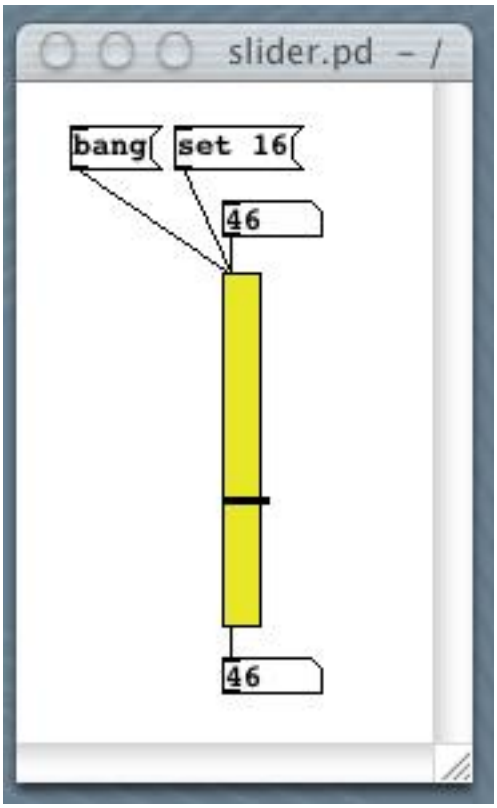

Figure 8.7

Un altro elemento di GUI, il vslider[19], è illustrato in Figure 8.7. In questo patch è anche visibile un oggetto number, il quale invia un numero in uscita (**outlet**) ogni volta che ne viene cambiato il valore, per dragging con il mouse o per immissione di numeri dal suo input.

[19]See the file at <http://cnx.org/content/m14602/latest/./slider.pd>

8.1.3 Array

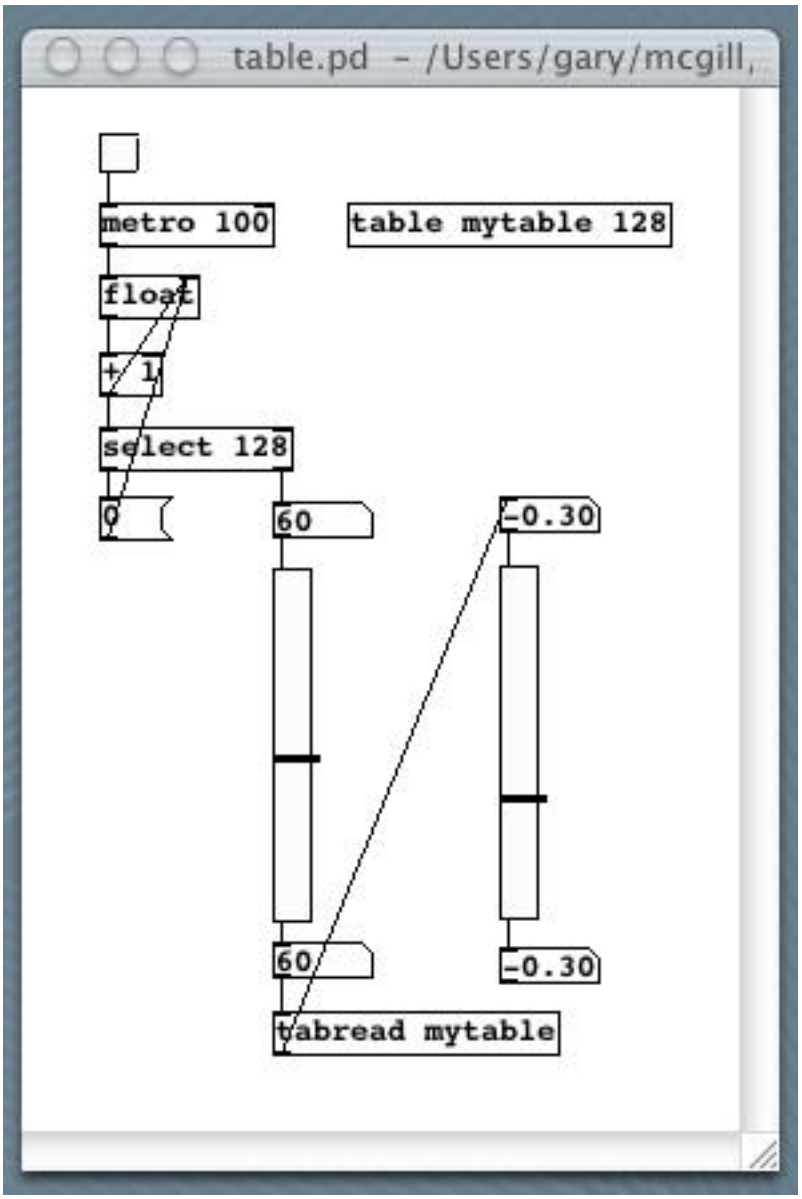

Figure 8.8

In Pure Data, un array è un contenitore di numeri ed un elemento di visualizzazione al tempo stesso. Infatti, l'oggetto `table` istanzia l'array e crea un subpatch che ne visualizza la rappresentazione grafica. Questa rappresentazione è anche uno strumento di input, in quanto i valori dell'array possono essere "disegnati" con il mouse. Per leggere e scrivere i singoli elementi dell'array si usano gli oggetti `tabread` e `tabwrite`. L'utilizzazione di `tabread` per scandire un array[20] è illustrata in Figure 8.8.

[20] See the file at <http://cnx.org/content/m14602/latest/./table.pd>

8.1.4 Selezioni, routing, multiplexing

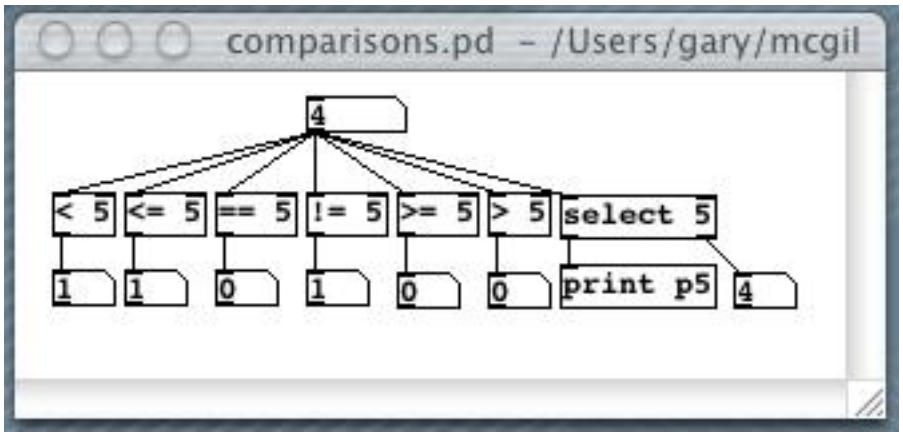

Figure 8.9

In Figure 8.9 sono illustrati blocchi di confronto e selezione[21]. In particolare, il blocco `select` attiva l'uscita di sinistra se l'ingresso (**inlet**) è uguale al suo parametro, altrimenti attiva l'uscita di destra.

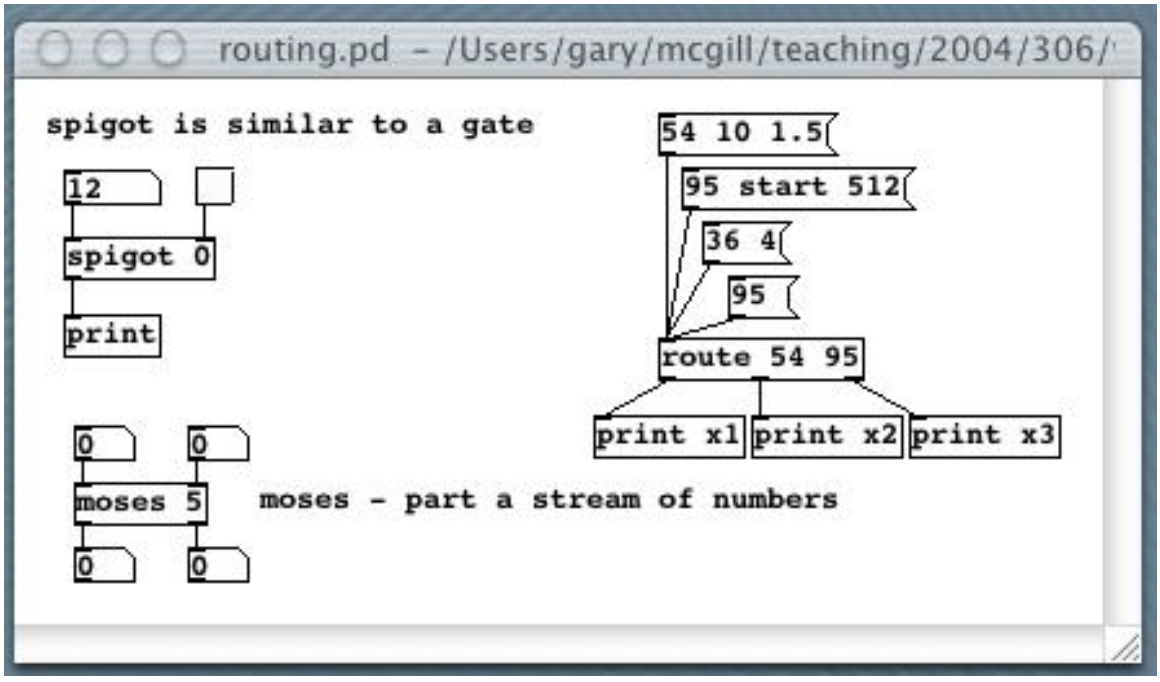

Figure 8.10

[21] See the file at <http://cnx.org/content/m14602/latest/./comparisons.pd>

In Figure 8.10 sono illustrati tre elementi di routing[22]. Il blocco `spigot` trasmette messaggi dall'inlet di sinistra all'outlet in dipendenza dallo stato dell'ingresso (di controllo) di destra. In sostanza si tratta di un **gate** con controllo di apertura e chiusura. Invece, `moses` consente di smistare alle uscite sinistra e destra i valori di ingresso che sono rispettivamente minori o maggiori (o uguali) del valore di controllo passato dalla inlet di destra. Infine, `route` smista dei messaggi ricevuti in ingresso sulla base del loro primo elemento, mettendolo a confronto con gli argomenti. Un multiplexer[23] si può ottenere combinando la `route` con la `pack`.

Exercise 8.1 *(Solution on p. 90.)*

Realizzare un multiplexer e un demultiplexer utilizzando gli oggetti `pack`, `route` e `spigot`.

8.1.5 Ritardi, code e gestione del tempo

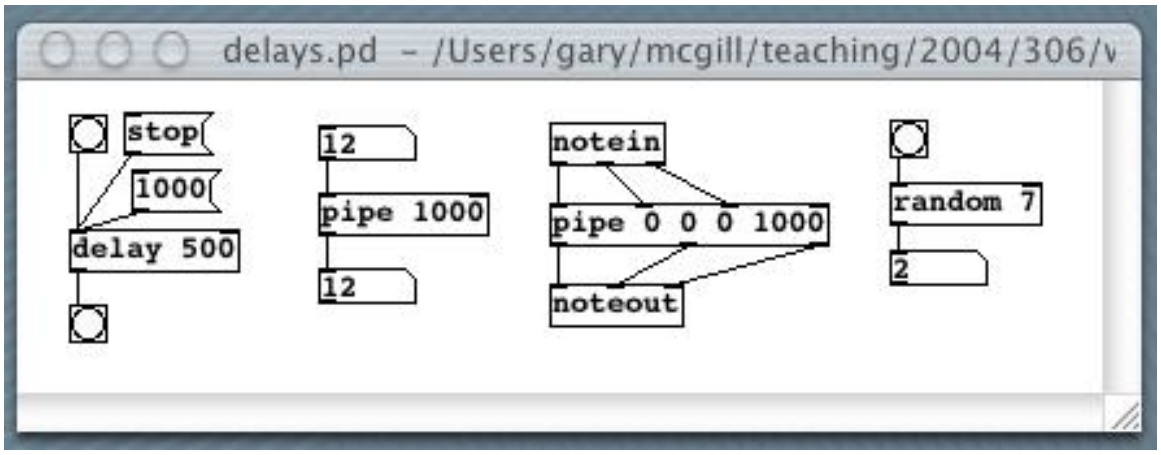

Figure 8.11

In Figure 8.11 è illustrata l'utilizzazione di ritardi e code[24]. La `delay` ritarda un bang in ingresso di un numero di millisecondi pari al suo argomento. Se si vuole ritardare un flusso di dati bisogna usare la `pipe`, la quale è realizzata come coda a buffer circolare (p. 52).

[22] See the file at <http://cnx.org/content/m14602/latest/./routing.pd>
[23] http://en.wikipedia.org/wiki/Multiplexer
[24] See the file at <http://cnx.org/content/m14602/latest/./delays.pd>

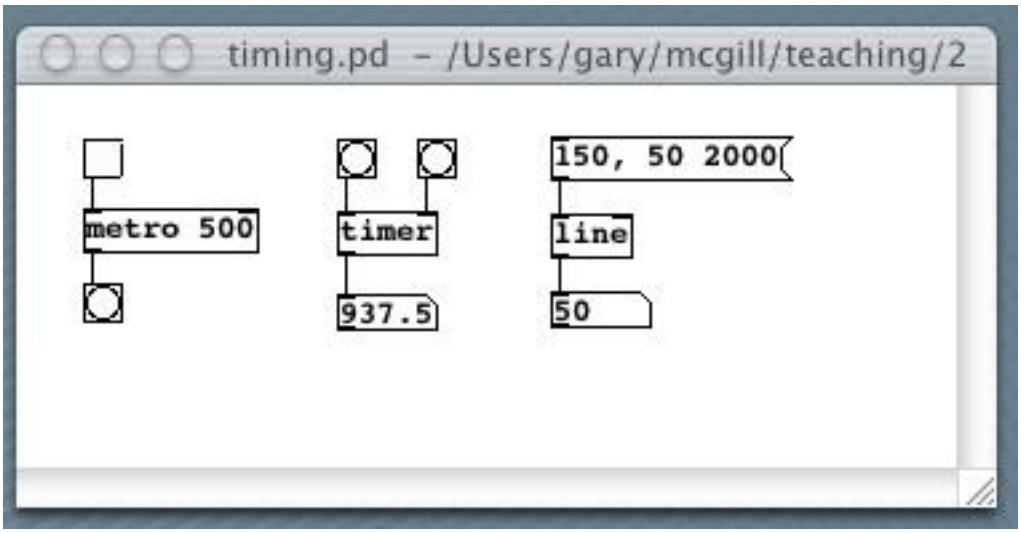

Figure 8.12

In Figure 8.12 si vedono alcuni oggetti che consentono di scandire, misurare, e avanzare[25] il tempo. Rispettivamente, ciò si ottiene con `metro`, `timer`, e `line`. Quest'ultimo oggetto genera una rampa lineare, da un valore iniziale a uno finale, in un certo tempo.

Exercise 8.2 ***(Solution on p. 90.)***

Nel patch di Figure 8.8, si provi a rimuovere e ripristinare i collegamenti che arrivano alla inlet di destra dell'oggetto `float`, e si verifichi che il reset dell'indice una volta raggiunto il limite di 128 può non avere luogo. Perché? Come si può introdurre in maniera deterministica il reset dell'indice?

8.1.6 Message Passing

Il funzionamento dataflow di Pure Data avviene mediante **pipe** che corrispondono alle connessioni tra oggetti. E' anche possibile operare in regime di message passing[26] , cioè ricevere (receive[27]) e spedire (send[28]) dati tra diverse parti di un patch o tra patch diversi. Ciò è illustrato in Figure 8.13. Invece, l'oggetto `value` realizza una sorta di variabile globale.

[25] See the file at <http://cnx.org/content/m14602/latest/./timing.pd>
[26] http://en.wikipedia.org/wiki/Message_passing
[27] See the file at <http://cnx.org/content/m14602/latest/./receive.pd>
[28] See the file at <http://cnx.org/content/m14602/latest/./send.pd>

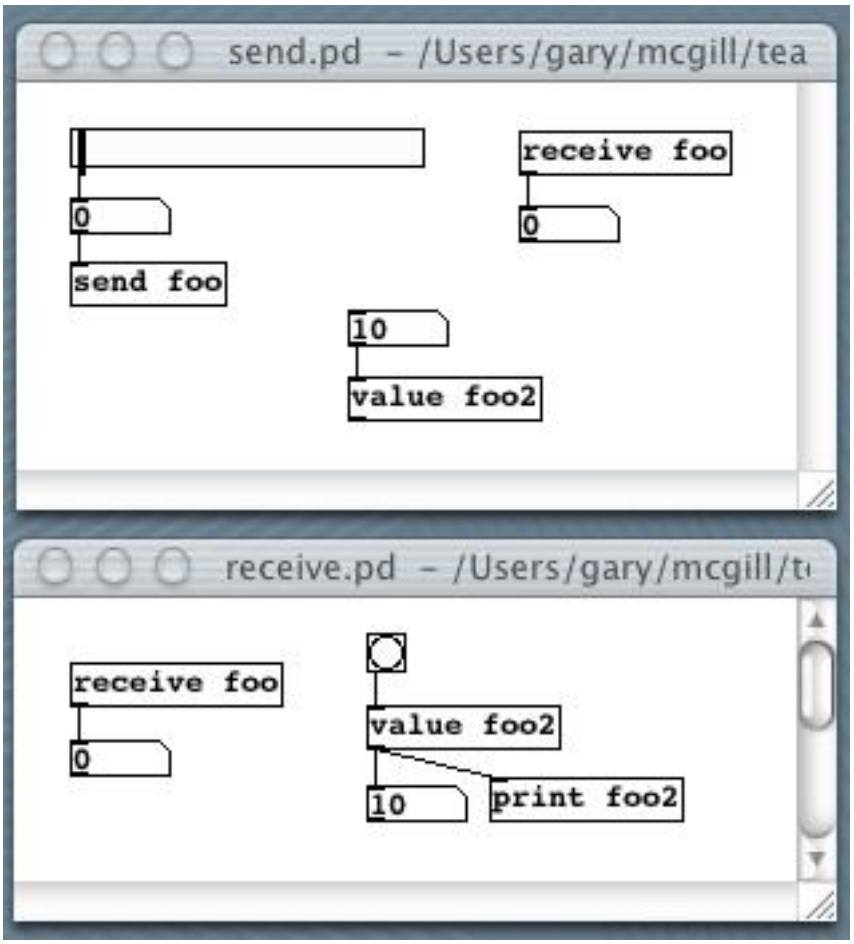

Figure 8.13

8.1.7 Modularità

Nella programmazione visuale la modularità si ottiene con i meccanismi di **subpatching** che, in Pure Data, sono di due tipi:

- Si utilizza l'oggetto `pd` per dichiarare un subpatch. In quest'ultimo, si specificano le `inlet` e `outlet`. Si veda Figure 8.14.
- Si costruisce un patch come file separato, che a questo punto diventa uno degli oggetti a disposizione di Pure Data. Si veda Figure 8.15.

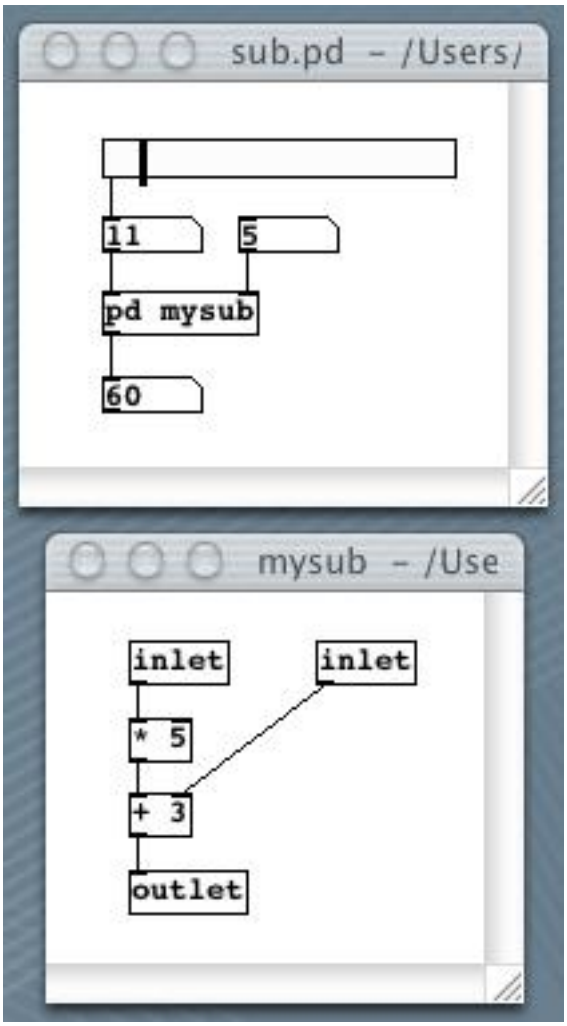

Figure 8.14

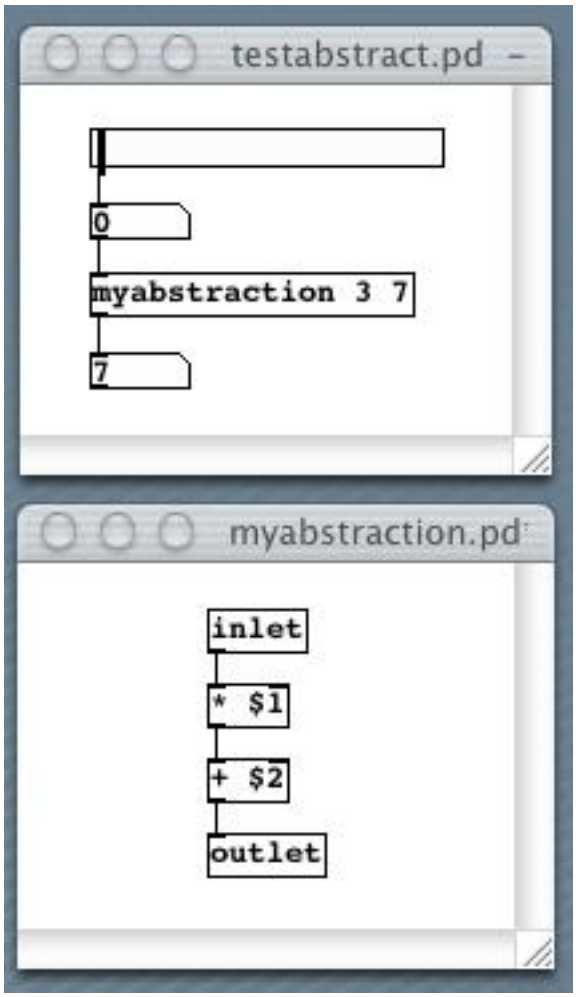
testabstract.pd
0
myabstraction 3 7
7
myabstraction.pd
inlet
* $1
+ $2
outlet

Figure 8.15

Solutions to Exercises in Chapter 8

Solution to Exercise 8.1 (p. 85)
Pure Data patch[29].
Solution to Exercise 8.2 (p. 86)
E' sufficiente introdurre un elemento di disaccoppiamento (ad esempio, una `pipe 0`) che imponga un ordine tra l'immissione del numero incrementato e l'immissione di 0 nella inlet di destra.

[29] See the file at <http://cnx.org/content/m14602/latest/./muxdemux.pd>

Chapter 9

Comunicazione tra Applicazioni[1]

9.1 Socket

Il protocollo TCP/IP
La comunicazione tra macchine diverse è strutturata secondo cinque strati:

- Application layer
- Transport layer
- Internet layer
- Network access layer
- Physical layer

Il protocollo IP (Internet Protocol) viene usato nel layer 3 per instradare i messaggi tra reti diverse. Esso è presente anche nei router, cioè in quelle macchine che fungono da snodo per la comunicazione tra le reti. Il protocollo TCP (Transmission Control Protocol) viene usato nel layer 2 per fornire una modalità di comunicazione che sia affidabile e che garantisca in ricezione lo stesso ordine del flusso di dati (**stream**) imposto in trasmissione. TCP deve essere presente nei due terminali (host) della comunicazione.
Indirizzi
Ogni host ha un indirizzo internet univoco in senso globale, usato da IP per instradare i messaggi. Esso è di 32 bit (in IPv4, e di 128 bit in IPv6, diventato standard nel 1996) e normalmente fornito in notazione "dotted-quad" (e.g. 157.138.204.250) o come nome di host (e.g. www.iuav.it). La traduzione da nomi a numeri puo' avvenire mediante il file /etc/hosts o mediante il database distribuito chiamato Domain Name Service (DNS). Ogni applicazione deve avere un numero di port (di 16 bit) univoco all'interno dell'host, usato da TCP per gestire la comunicazione da e verso l'applicazione. I numeri fino a IPPORT_RESERVED (tip. 1024) sono riservati per un accesso ai server di sistema (HTTP, FTP, etc.).
Dati
Un blocco di dati inviato da una applicazione si arricchisce di informazioni aggiuntive (header) mentre scende verso gli strati piu' bassi dei protocolli di comunicazione.

- A livello di trasporto viene aggiunto il TCP header (in modo da formare un TCP segment), il quale contiene le seguenti informazioni:
  - Destination Port
  - Sequence Number (per poter riordinare i blocchi di dati)
  - Checksum (per poter rilevare errori di trasmissione)
- A livello di internet viene aggiunto l'IP header, in modo da formare un IP datagram. Tra le informazioni contenute in questo header vi è l'indirizzo dell'host di destinazione.

[1] This content is available online at <http://cnx.org/content/m14632/1.4/>.

Affidabilità della trasmissione
All'interno del TCP header, il sequence number tiene traccia dell'ordine dei pacchetti inviati (non piu' grandi di 64KB), in modo che in ricezione si possano individuare i pacchetti persi e ricostruire l'ordine corretto. Mediante il checksum il ricevente è in grado di rilevare errori di trasmissione. Il ricevente notifica la corretta ricezione mediante messaggi di risposta. Essendo questo protocollo basato sulla connessione di due nodi, tale connessione (virtuale) deve essere stabilita prima di iniziare la comunicazione. Il protocollo UDP (User Datagram Protocol) consente la comunicazione priva dell'overhead dovuto ai controlli di integrità e consistenza dei messaggi. Essenzialmente, UDP aggiunge ad IP soltanto la capacita' di indirizzare dei port. Nell'UDP header c'è anche un campo checksum, ma non è previsto l'acknowledgement di ricezione corretta. Il protocollo UDP non ha bisogno di una connessione stabilita prima della comunicazione. Il protocollo UDP, oltre ad avere minore overhead rispetto a TCP, è anche essenziale al supporto di certe attività. Si pensi ad esempio al programma ping, il quale verifica la qualità di una connessione. Esso può funzionare solo se ha la possibilita' di verificare la perdita eventuale di pacchetti.

Il modello client-server e i socket
Un processo server attende le eventuali connessioni di processi client. Quando la connessione viene stabilita, il server esegue dei compiti in base a quanto ricevuto dal client e poi, di solito, la connessione viene interrotta. La comunicazione tra client e server deve essere affidabile. La programmazione della comunicazione mediante TCP/IP viene di solito effettuata mediante l'interfaccia socket BSD, introdotta da UNIX 4.2BSD. E', di fatto, una forma di Inter-Process Communication che si aggiunge alle altre forme di comunicazione tra processi di UNIX (pipe, shared memory, signals, message queues, semaphores) con la peculiarità di consentire la comunicazione tra macchine diverse fornite di indirizzo IP.

Definition 9.1: Socket
è uno dei due terminali di una connessione bidirezionale tra due processi in esecuzione su macchine collegate ad una rete.

Java fornisce una suite di classi che consentono di stabilire delle connessioni mediante socket in modo indipendente dalla particolare realizzazione dei socket da parte del sistema operativo sottostante. In Java, le classi Socket e ServerSocket vengono usate per stabilire la connessione rispettivamente dal lato client e dal lato server. La comunicazione client-server mediante socket avviene come segue:

CLIENT	-	SERVER
conosce hostname e port number del server	-	ascolta il socket
richiede la connessione al server	→	accetta la connessione
crea un socket e lo usa per comunicare con il server	←	ottiene un nuovo socket su un port differente
	-	continua ad ascoltare il socket originario

Table 9.1

Il modello message passing
Il protocollo di trasmissione UDP realizza di fatto un modello di comunicazione di tipo message passing con ricezione bloccante e invio non bloccante. Quindi i due partner comunicano con una forma di rendez vous "lasco". E' demandato al programmatore il compito di assicurare la corretta ricezione dei messaggi (**datagram**) se essa è importante per l'applicazione.

Definition 9.2: Datagram
è un messaggio la cui trasmissione in rete non assicura l'effettiva ricezione, il tempo di arrivo, e l'integrita' del contenuto.

La comunicazione in UDP avviene mediante indirizzamento indiretto di tipo molti a uno: la primitiva send indirizza il proprio messaggio ad uno specifico identificatore del port (che gioca il ruolo di una mailbox) di accesso al processo destinazione (hostname e port). Il processo destinazione conosce l'identità del processo sorgente mediante hostname e port specificati in testa al messaggio. Mediante UDP, è anche possibile che

molti client si mettano in ascolto di messaggi inviati (broadcast) da un processo server. Questa è una comunicazione di tipo message passing uno a molti. In Java, le classi DatagramSocket e DatagramPacket consentono di confezionare dei messaggi e di spedirli e riceverli attraverso socket di tipo UDP. La classe MulticastSocket consente ad un client di associarsi al gruppo di client che possono ricevere il broadcast e di ricevere in modo passivo i messaggi inviati dal server.

Comunicazione client-server in Processing

Tra le core libraries di Processing, la Network[2] consente di creare client e server.

Example 9.1: Controller come client

In questo esempio la posizione del mouse sulla finestra determina valori di ampiezza e frequenza che controllano i parametri di un oscillatore server. Più correttamente, si assume che ci sia un oscillatore con due server già attivi: uno in ascolto sulla porta `5214` (frequenza) e uno in ascolto sulla porta `5215` (ampiezza). Il numero IP `127.0.0.1` indica localhost, o la medesima macchina su cui sta girando il codice Processing.

```
import processing.net.*;
int portf = 5214;
int porta = 5215;
Client frequency, amplitude;
int val = 0;

void setup() {
  size(200, 200);
  frequency = new Client(this, "127.0.0.1", portf);
  amplitude = new Client(this, "127.0.0.1", porta);
}

void draw() {
  background(val);
  frequency.write(str(mouseX));
  frequency.write(';');
  amplitude.write(str(height - mouseY));
  amplitude.write(';');
}
```

Comunicazione client-server in Pure Data

Pure Data mette a disposizione gli oggetti `netsend` e `netreceive` per realizzare la comunicazione di tipo client-server. L'oggetto `netreceive` apre un socket di ricezione TCP (stream) o UDP (datagram) su un port specificato come argomento. Quando si usa TCP, l'outlet di destra restituisce il numero di client che hanno aperto su questo socket la connessione. La comunicazione via UDP si sceglie aggiungendo un secondo argomento `1` a `netsend` e `netreceive`.

Dispositivi connessi alla rete

E' frequente oggigiorno l'utilizzazione di protocolli di rete per l'invio di dati raccolti con sensori. Esempi di dispositivi che fanno questo sono le macchine Kroonde e Toaster di La Kitchen, azienda francese che ha purtroppo chiuso le proprie attività di recente. Queste macchine si appoggiano a UDP per inviare i dati raccolti dai sensori, e sono a disposizione dei patch per Pure Data, basati su `netreceive`, sono a disposizione.

[2] http://processing.org/reference/libraries/net/index.html

Example 9.2: Oscillatore come server
Un oscillatore che risponde a richieste quali quelle riportate in Example 9.1 (Controller come client) è rappresentato in Figure 9.1. Si noti come in Example 9.1 (Controller come client) il valore numerico rappresentante la posizione del mouse viene convertito in stringa nel momento in cui esso viene inviato. Inoltre, viene inserito il separatore `';'` per consentire l'estrazione dei numeri da parte di Pure Data.

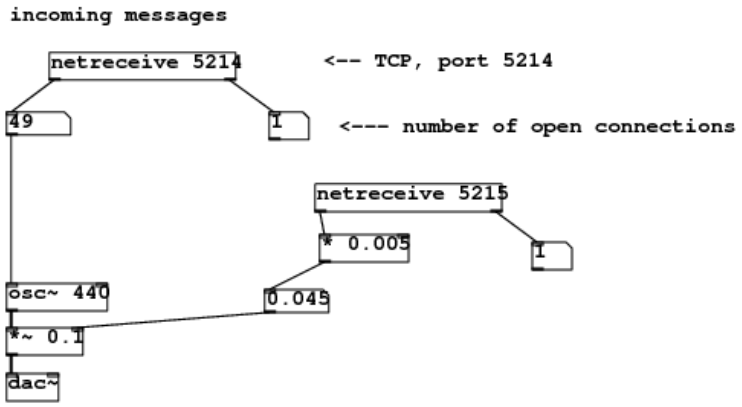

Figure 9.1

Exercise 9.1 *(Solution on p. 97.)*
Si modifichino i client e server di Example 9.1 (Controller come client) e Example 9.2 (Oscillatore come server) in maniera da realizzare una comunicazione bidirezionale. Per esempio, fare in modo che le consolle di Processing e Pure Data scrivano entrambe i valori scambiati di frequenza.

9.2 MIDI

Il protocollo MIDI[3] (Musical Instrument Digital Interface; IPA) è un protocollo di comunicazione tra dispositivi elettronici, introdotto dalle industrie degli strumenti musicali all'inizio degli anni '80 per garantire l'interoperabilità dei propri apparati. Il fatto che esso non si occupa di trasmettere segnali ma messaggi relativi ad eventi ha fatto sì che la sua diffusione sia stata ampia anche in ambito non strettamente musicale. Lo

[3] http://en.wikipedia.org/wiki/Midi

standard MIDI comprende sia il protocollo di comunicazione sia la definizione dell'interfaccia fisica. Questa trasmette alla velocità di 31,250 bit al secondo dei pacchetti costituiti da un bit di start (0), otto bit (un byte) di dati, un bit di stop (1). I messaggi scambiati sono relativi a eventi discreti (**note-on**, **note-off**, etc.) o processi che inviano flussi di valori (**pitch bend**, **aftertouch**, **control change**, etc.). Tutti questi messaggi appartengono a uno (o tutti) di 16 canali. Un byte di **status**, contenente canale e codice relativo al tipo di messaggio, precede uno o due byte di dati. Per le interazioni continue che ci si trova a dover trattare nell'interaction design, sono particolarmente utili i control change[4] . Il MIDI ha avuto un notevole successo come standard industriale e, per molte applicazioni, funziona egregiamente ed ha un punto di forza nella sua semplicità. Tuttavia, ha gli svantaggi di una comunicazione seriale e a bassa velocità, nonché le limitazioni legate alla sua bassa espressività e all'origine nell'ambito degli strumenti musicali a tastiera.

In Pure Data, è disponibile una gamma di oggetti per la gestione dei messaggi MIDI. La pagina di documentazione di riferimento è riportata in Figure 9.2. Si vede, ad esempio, che l'oggetto `ctlin` ha tre outlet che forniscono, rispettivamente, il valore, il numero, e il canale di un control change.

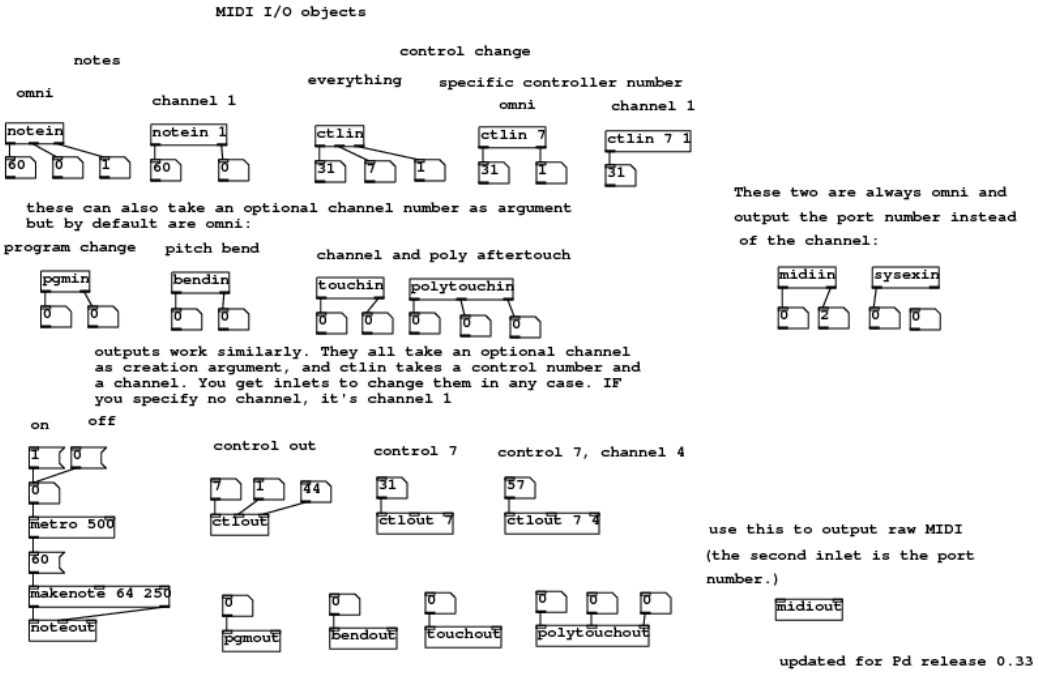

Figure 9.2

Per Processing e per Java esistono librerie[5] che includono funzioni di sequencer[6] . Il MIDI può essere usato anche per far dialogare tra loro applicazioni che girano in uno stesso computer o su computer diversi all'interno di una rete. Il sistema operativo del Macintosh, ad esempio, ha una applicazione chiamata `Audio MIDI Setup` che consente di impostare uno Inter-Application Communication - `IAC Driver` o un MIDI `Network` driver. Se abilitati, questi device driver saranno poi selezionabili come input e output device dalle preferences di Pure Data.

[4] http://www.harmony-central.com/MIDI/Doc/table3.html
[5] http://processing.org/reference/libraries/
[6] http://en.wikipedia.org/wiki/Music_sequencer

9.3 OSC

OpenSound Control - OSC[7] è un protocollo di comunicazione che consente a diversi dispositivi elettronici e applicazioni di scambiarsi dati in tempo reale su un supporto di rete. E' stato sviluppato per superare le limitazioni di MIDI e per sfruttare le possibilità di TCP e UDP, consentendo al tempo stesso una semantica raffinata dei messaggi. Le caratteristiche principali di OSC sono:

- indirizzamento simbolico "tipo-URL" (e.g., `/voices/synth1/osc1/modfreq`)
- indipendenza dal tipo di trasporto (tipicamente UDP socket)
- argomenti numerici e simbolici ai messaggi
- espressioni regolari[8] per la specificazione di più destinatari (e.g., `/voices/synth1/osc[1-4]/modfreq`)
- temporizzazione fine
- accorpamento di messaggi in **bundle** che richiedono azione simultanea.
- possibilità di interrogare un OSC server sulle sue capacità

L'unità fondamentale di OSC è il messaggio, il quale consiste di un pattern di indirizzo, una stringa di tipo (**type tag**), e un certo numero di argomenti. Ad esempio, un messaggio può essere indirizzato a `/voice/3/freq`, specificare come type tag un singolo numero floating point, e il suo argomento può essere 261.62558. L'indirizzamento aperto di OSC consente a ogni server di definire il proprio spazio di indirizzamento in relazione alla propria organizzazione di servizi. Un'ottima introduzione a OSC[9] si può scaricare, insieme a dettagliata documentazione e a link alle realizzazioni software presso il sito ufficiale di OSC[10] .

Per Processing esiste la libreria oscP5[11] che consente di produrre ed interpretare messaggi e bundle di OSC.

Exercise 9.2

Si installi oscP5 per Processing e si provino gli esempi della documentazione[12] . In particolare, si cerchi di comprendere la composizione e decomposizione dei messaggi. Nell'esempio `oscP5plug` è illustrato un meccanismo di event-based programming[13] , secondo il quale è possibile instradare messaggi con un determinato pattern di indirizzo a un metodo specifico di una classe (**event handler**).

[7] http://en.wikipedia.org/wiki/Open_Sound_Control
[8] http://en.wikipedia.org/wiki/Regular_expression
[9] http://opensoundcontrol.org/files/Open+Sound+Control-state+of+the+art.pdf
[10] http://opensoundcontrol.org/
[11] http://www.sojamo.de/iv/index.php?n=11
[12] http://www.sojamo.de/content/p5/oscP5/documentation/
[13] http://en.wikipedia.org/wiki/Event-based_programming

Solutions to Exercises in Chapter 9

Solution to Exercise 9.1 (p. 94)

```
import processing.net.*;
int portf = 5214;
int porta = 5215;
int ports = 5204;
Client frequency, amplitude;
Server myServer;
int val = 0;

void setup() {
  size(200, 200);
  frequency = new Client(this, "127.0.0.1", portf);
  amplitude = new Client(this, "127.0.0.1", porta);
  myServer = new Server(this, ports);
}

void draw() {
  background(val);
  frequency.write(str(mouseX));
  frequency.write(';');
  amplitude.write(str(height - mouseY));
  amplitude.write(';');
  Client thisClient = myServer.available();
  if (thisClient !=null) {
    String whatClientSaid = thisClient.readString();
    if (whatClientSaid != null) {
      print(whatClientSaid);
    }
  }
}
```

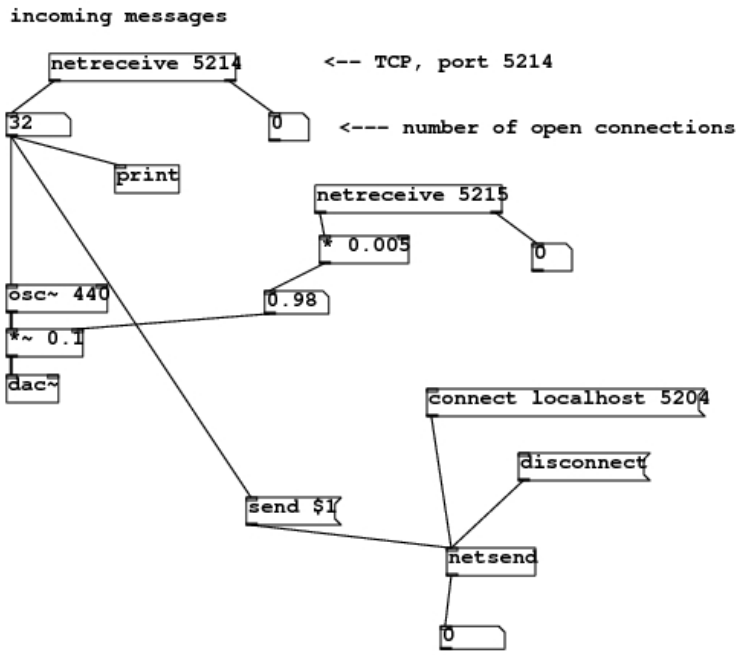
incoming messages
netreceive 5214
<-- TCP, port 5214
32
0
<--- number of open connections
print
netreceive 5215
* 0.005
0
osc~ 440
0.98
*~ 0.1
dac~
connect localhost 5204
disconnect
send $1
netsend
0

Figure 9.3

Glossary

D **Datagram**

è un messaggio la cui trasmissione in rete non assicura l'effettiva ricezione, il tempo di arrivo, e l'integrita' del contenuto.

G **global scope**

definita fuori da setup() e draw(), la variabile è visibile e usabile ovunque nel programma

L **local scope**

definita all'interno di un blocco di codice o di una funzione, la variabile assume valori locali al blocco o alla funzione, ed eventuali valori assunti da una variabile globale omonima vengono ignorati

S **scope**

all'interno di un programma, regione in cui si può accedere ad una variabile e modificarne il valore

Socket

è uno dei due terminali di una connessione bidirezionale tra due processi in esecuzione su macchine collegate ad una rete.

Index of Keywords and Terms

Keywords are listed by the section with that keyword (page numbers are in parentheses). Keywords do not necessarily appear in the text of the page. They are merely associated with that section. *Ex.* apples, § 1.1 (1) **Terms** are referenced by the page they appear on. *Ex.* apples, 1

A accelerazioni e decelerazioni, § 4(37)
array, § 3(25)

B buffer circolare, § 4(37)

D Datagram, 92

E Elaborazione di Media in Processing, 9
events, § 7(67)

F filtering, § 7(67)

G global scope, 12

H hysteresis, § 7(67)

I image processing, § 2(9)
interaction design, 9
Interaction Design Patterns, § 1(1)

L List, § 5(45)
local scope, 12

M mapping, § 3(25)
MIDI, § 9(91)

O object-oriented programming, § 2(9)
OSC, § 9(91)
oscillatore tabellare, § 4(37)

P procedural programming, § 2(9)
Processing, 9
processing language and environment, § 2(9)
Pure Data, § 8(77)

Q Queue, § 5(45)

S scope, 12
Socket, 92
Sockets, § 9(91)
sound processing, § 2(9)
Stack, § 5(45)

T Threads. Non-blocking I/O. Synchronization., § 6(57)
Thresholds, § 7(67)
timers, § 7(67)
type casting, 19

V Visual programming, § 8(77)

“ “Intersezione cerchio-cerchio”, 18

Attributions

Collection: *Programmazione di Artefatti Interattivi*
Edited by: Davide Rocchesso
URL: http://cnx.org/content/col10417/1.9/
License: http://creativecommons.org/licenses/by/3.0/

Module: "Pattern emergenti dal design dell'interazione"
By: Davide Rocchesso
URL: http://cnx.org/content/m14501/1.11/
Pages: 1-8
Copyright: Davide Rocchesso
License: http://creativecommons.org/licenses/by/3.0/

Module: "Programmare in Processing"
By: Davide Rocchesso
URL: http://cnx.org/content/m12614/1.16/
Pages: 9-24
Copyright: Davide Rocchesso
License: http://creativecommons.org/licenses/by/2.0/

Module: "Array e Mappe"
By: Davide Rocchesso
URL: http://cnx.org/content/m14505/1.10/
Pages: 25-35
Copyright: Davide Rocchesso
License: http://creativecommons.org/licenses/by/2.0/

Module: "Oscillazioni, ritardi, e fluttuazioni del tempo"
By: Davide Rocchesso
URL: http://cnx.org/content/m14532/1.4/
Pages: 37-44
Copyright: Davide Rocchesso
License: http://creativecommons.org/licenses/by/2.0/

Module: "Liste, pile e code"
By: Davide Rocchesso
URL: http://cnx.org/content/m14556/1.5/
Pages: 45-55
Copyright: Davide Rocchesso
License: http://creativecommons.org/licenses/by/2.0/

Module: "Thread e I/O non bloccante"
By: Davide Rocchesso
URL: http://cnx.org/content/m14565/1.7/
Pages: 57-66
Copyright: Davide Rocchesso
License: http://creativecommons.org/licenses/by/2.0/

Module: "Segnali nel tempo: soglie e filtri"
By: Davide Rocchesso
URL: http://cnx.org/content/m14574/1.3/
Pages: 67-75
Copyright: Davide Rocchesso
License: http://creativecommons.org/licenses/by/2.0/

Module: "Programmazione Visuale"
By: Davide Rocchesso
URL: http://cnx.org/content/m14602/1.7/
Pages: 77-90
Copyright: Davide Rocchesso
License: http://creativecommons.org/licenses/by/3.0/

Module: "Comunicazione tra Applicazioni"
By: Davide Rocchesso
URL: http://cnx.org/content/m14632/1.4/
Pages: 91-98
Copyright: Davide Rocchesso
License: http://creativecommons.org/licenses/by/2.0/

Programmazione di Artefatti Interattivi
Nel design degli artefatti interattivi, cosi' come in altre aree della progettazione, si puo' procedere alla risoluzione di un problema mediante identificazione di pattern, cioe' di configurazioni che ricorrono sovente, e per le quali esistono soluzioni di efficacia consolidata.