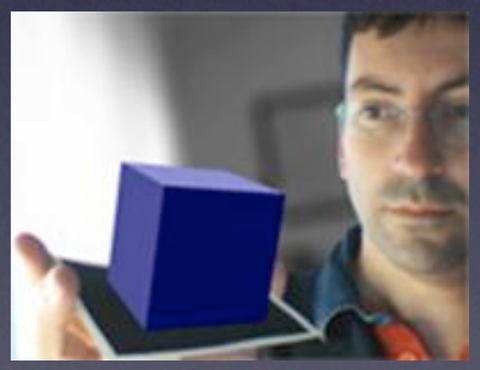
Computer Graphics

Introduction to *processing*



Jordi Linares i Pellicer
Escola Politècnica Superior d'Alcoi
Dep. de Sistemes Informàtics i Computació
jlinares@dsic.upv.es
http://www.dsic.upv.es/~jlinares

What is *processing* ?

- *Processing* is a Java dialect specifically designed for generating visual art, animation and any kind of graphic application
- Developed by artists to artists
- *Processing* is free software, available wherever Java is available (Mac OS, Linux, Windows etc.)
- Developed from 2001 at the MIT by Casey Reas and Ben Fry. Inspired in John Maeda's DBN (*Design By Numbers*)
- http://processing.org -> download centre and central point to everybody in the world of *processing*

Why *processing* ?

- Very, very easy to learn... first sketches in just a few minutes
- It has its own IDE (very easy)
- Faster learning than OpenGL + GLUT or others alternatives
- Powerful, very powerful. It allows to develop from very simple to very complex applications
- Scalable. It is possible to combine *processing* with pure Java and with Java libraries. A pure Java application can use *processing* libraries as well..

Why *processing* ?

- It allows 3 ways of programming: basic, procedural/ structured and object-oriented
- The first sketches can be developed in the basic mode (no functions, just direct code)
- More ambitious programs or sketches can use functions (C-like programming)
- Complex programs can be developed using object-oriented approach (Java classes)
- Eventually, it is possible to develop a pure Java application and use other environments (eclipse, netbeans) with the libraries of *processing*

Why *processing* ?

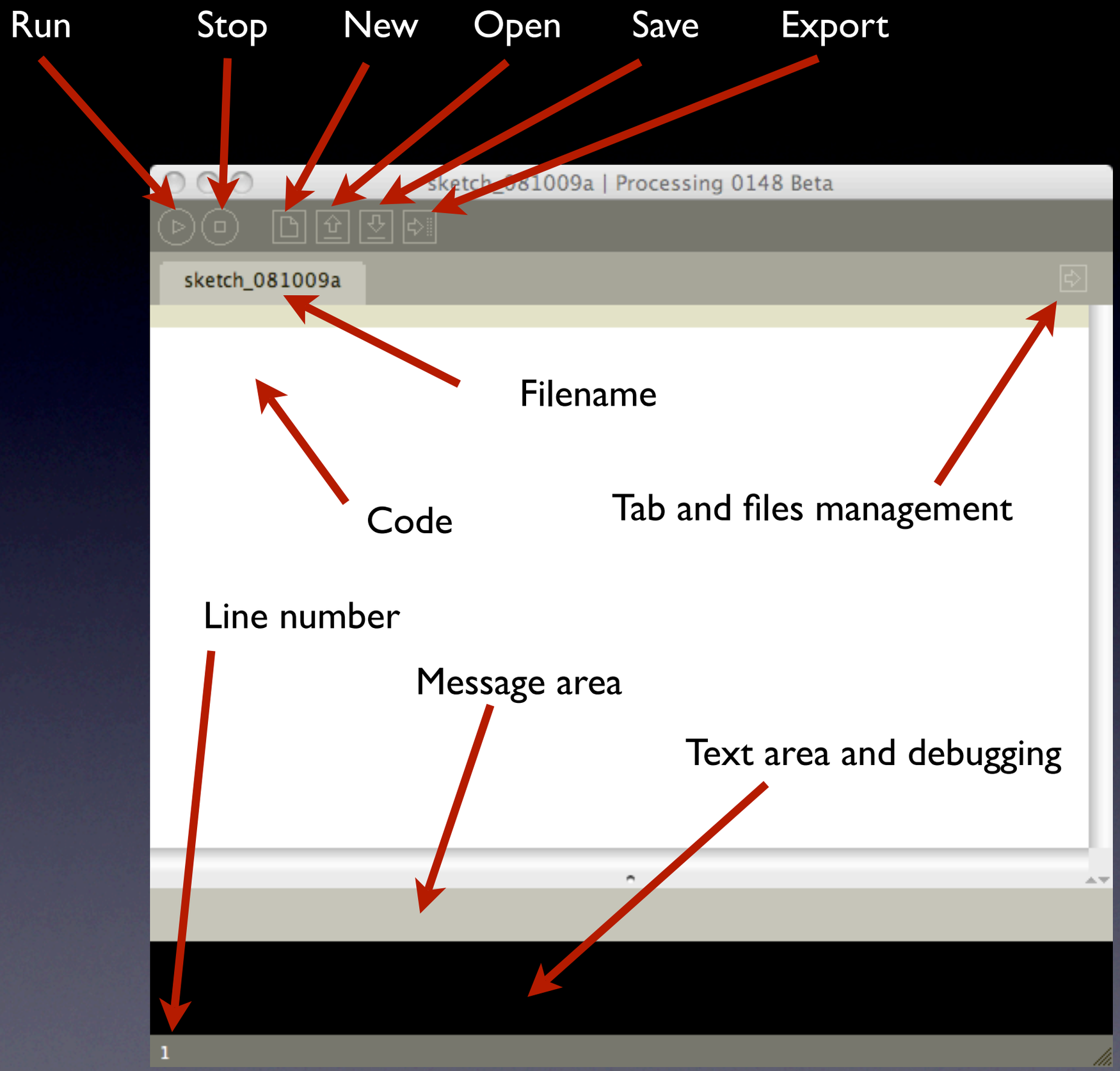
- Generates ready-to-run applications for the main platforms: Mac OS, Linux and Windows
- *processing* applications can also be executed in the Internet (as Java *applets*)
- It is possible to develop for mobile devices (http://mobile.processing.org)
- It allows connection with electronic devices and prototypes: Arduino and Wiring projects (http://hardware.processing.org)

Graphic packages. History

- Standards:
  - 3D Core Graphics System. ACM and SIGGRAPH (1977)
  - GKS (Graphical Kernel System). 2D. ISO-ANSI (1985)
  - GKS-3D (1988)
  - PHIGS (Programmer's Hierarchical Interactive Graphics System). 3D. (1988)
  - SRGP and SPHIGS (Foley)
- Others
  - OpenGL
  - XWindows, Microsoft Windows, Mac OS ...
  - VRML, X3D (Internet oriented)
  - POV-Ray, Renderman
  - Java2D, Java3D
  - DirectX (Direct-3D)
  - etc.

The *processing* IDE

- *Processing* has its own IDE (Integrated Development Environment) developed in Java
- Simple and easy. It is enough for most of the applications. Migration to eclipse, e.g., is possible in front of more complex applications
- It is known as PDE (*processing* development environment)


Run
Stop
New
Open
Save
Export
sketch_081009a | Processing 0148 Beta
sketch_081009a
Filename
Code
Tab and files management
Line number
Message area
Text area and debugging
1

The *processing* IDE

- When saving an application, a directory with the application name is generated. Inside this directory, the main file of the application is saved, with the same name and the extension *.pde*
- 64 characters is the maximum length of filename, no spaces allowed
- More code files can be generated for the same application (tab management). These files have by default the same extension *.pde*, but they can also have the *.java* extension (they will be treated as Java source files).
- Additional files are perfect for new functions or classes (in that case, it is compulsory the implementation of the built-in function *setup*)
- Visible tabs will be the ones considered when the application is generated
- In order to open a project we have to open the main *.pde* file of the directory (though any *.pde* inside will work)

The *processing* IDE

- *Run* and *Stop* allow to start and interrupt the application execution
- *Export* allows to generate an '*applet*' for the application. '*Export Application*' menu command allows to generate a ready-to-run application for Mac OS, Linux or Windows. Everything will be generated in specific directories inside the main directory of the application
- '*Skecthbook*' is a user's default directory but any directory can be used to save applications

The *processing* IDE

- Other interesting options:
  - *Import Library*. Facilitates the inclusion of *import* sentences of the most common libraries
  - *Add File*. Allows to add images, fonts an other multimedia information in a special directory inside the application: the directory 'data'. It is in this directory where we have to save any element we want to use inside our application (e.g., through the function '*loadImage*'). The directory 'data' will be created if it was not present previously.
  - *Create font*. Facilitates *applet* creation that use system fonts by ensuring that these font will appear correctly in any Internet browser.
  - *Help*. Help about *processing*, allowing even to consult the reference of the function we have selected in the code

The *processing* language

- *processing* is based on Java 1.4.2 (just some modifications in order to simplify programming)
- It is possible to use 1.5 and beyond but by using other development environment (a pure Java application and using the graphic library of *processing*)
- Complete documentation at:
  - http://java.sun.com/j2se/1.4.2/docs/api/index.html
- *processing* allow to program in a 'C'-like style (structured programming, as a set of functions). But it is convenient to take most of its object-oriented approach (slight modification to Java classes)

The *processing* language

processing code (.pde)

Preprocessor (processing)

Java code (.java)

Compiler (javac)

bytecode (.class)

Execution

JVM (Java Virtual Machine)

The *processing* language

- Variables

  `type name;`
- Basic types

  `byte`, `short`, `int` (178), `long` (8864L), `float` (37.266F), `double` (37.266/26.77e3), `char` (‘c’), `boolean` (true/false)
- Arrays

  `byte[ ] array;`
- Inicialization

  `type name = value;`
- Constants

  `final type variable = value;`

The *processing* language

- Arithmetic: `+, -, *, /, %, ++, --`
- Relationals: `>, >=, <, <=, ==, !=`
- Conditionals (logic): `&&, ||, !, &, |, ^`
- Bitwise operations: `<<, >>, &, |, ^, ~`
- Assignment: `=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=`
- Others: `?:, [], ., (type), new, instanceof`
- Precedence:

1. `expr++, expr--`
2. `++expr, --expr, ~, !`
3. `*, /, %`
4. `+, -`
5. `<<, >>`
6. `<, >, <=, >=, instanceof`
7. `==, !=`
8. `&`
9. `^`
10. `|`
11. `&&`
12. `||`
13. `?:`
14. `=, +=, -, =, *=, /=, %=, &=, ^=, |=, <<=, >>=`

The *processing* language

- Block delimiters: { instructions; }
- while:

```
while ( expr ) { instructions }
do { instructions } while ( expr )
```

- for

```
for ( begin; end; inc ) {instructions }
```

- if/else

```
if ( expr ) {instructions }
if ( expr ) {instructions } else { instructions }
```

- switch

```
switch ( var ) { case val: instructions default: }
```

- Jumps: `break`, `continue`, `return`

The *processing* language

- Comments:

```
// This comment goes until the end of line
/* This is a multiline
comment */
```

- Functions:

```
returned_type functionName(list of parameters)
```

If we define our own functions in *processing*, it is compulsory to implement the built-in function `setup()`

The *processing* language

- Example:

```
void setup() // It is executed the first
{
   size(100, 100);
   drawLine(5);
}
void drawLine(int x)  // Our own function
{
   line(x, 0, x, 99);
}
```

The *processing* language

- Strings:
  - `String message = “hello”;`
  - `String message = “hello” + “bye”;`
- A lot of methods:
  - `length()`
  - Access to a character: `charAt()`
  - `substring()`
  - Modification: `concat()`,`replace()`,`trim()`,`toLowerCase()`,etc.
  - Search: `indexOf()`, `lastIndexOf()`, etc.
  - Comparison: `startsWith()`, `endsWith()`, `compareTo()`, etc.
  - etc.
- Examples:

```
String message = “hello”+ “bye”;
int len = message.length();
int len = “hello”.length();
```

The *processing* language

- In *processing* (Java) an array is an object:
  - Its length can be known with the attribute *length*
  - Examples:

```
int[] vector; // vector is ‘null’
vector = new int[3]; // 3 components created
int len = vector.length; // length = 3
int item = vector[2]; // access to a component

int [][] matrix = new int[4][4];
matrix[0][0] = 12;  // Matrices
```

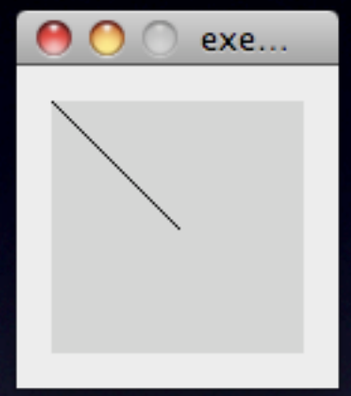
The *processing* language

- Which are the differences between the *processing* language and Java ?
  - The preprocessor of *processing* usually allows us a more relaxed syntax
  - It allows the basic mode: without own functions, only global variables and only direct lines of code
  - It allows to program following a 'C'-like style, defining our own functions, without having to define classes and methods
  - It allows to define and use classes, with a more simple syntax for the classes that then are transformed to Java classes
  - The basic and C-like style, global variables etc. are possible in *processing* thanks to the preprocessor; all these elements are integrated in a class (transparently to the programmer)

The *processing* language

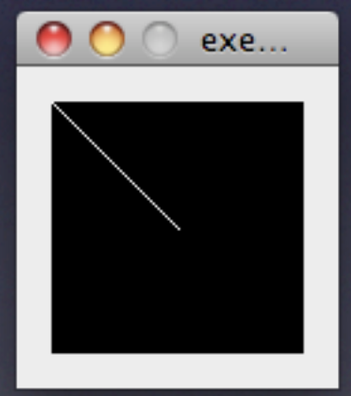
- Basic mode example (I)

```
// Minimum program in processing
line(0,0,50,50);
```



- Basic mode example (II)

```
// Variables and other elements ...
int i;
background(0); // black screen
stroke(255);   // white stroke
for (i = 0; i <= 50; i++)
      point(i, i);
```



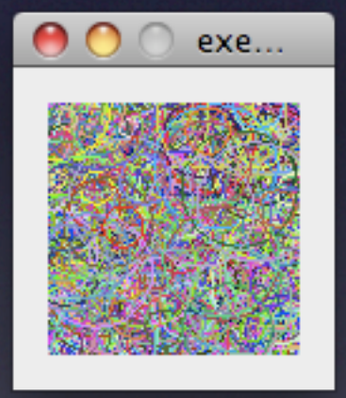
The *processing* language

- C-like style example:

```
// If own functions or classes are defined, an
// implementation for setup() must be defined
void setup()
{
    size(100,100); // Screen size
    noFill(); // Do not fill the circles
}

void draw() // Called each frame
{
    drawRandomCircle();
}

void drawRandomCircle()
{
    int r = int(random(50));
    // Stroke color
    stroke(random(255), random(255), random(255));
    // Circles with random circles (position, radius, color, sizes)
    ellipse(random(100), random(100), r, r);
}
```



The *processing* language

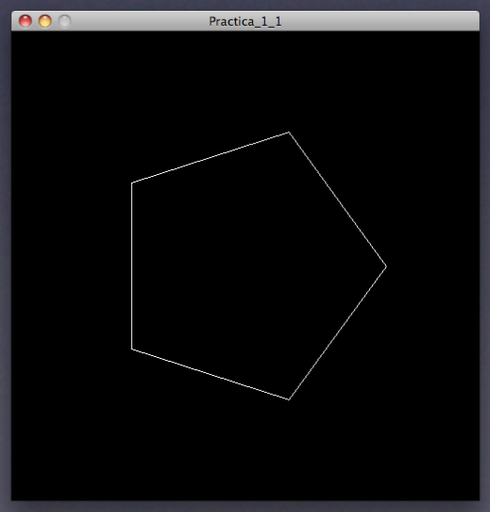
- Classes

```
void setup()
{
    size(100,100); // Screen size
    noFill(); }

void draw() // Called each frame
{
    Circle c = new Circle();
    c.draw();
}
// Circle class
class Circle
{
    // Attributes
    int r = int(random(50));
    int x = int(random(100));
    int y = int(random(100));
    color c = color(random(255), random(255), random(255));
    // Methods
    void draw()
    {
        stroke(c);
        ellipse(x, y, r, r);
    }
}
```

Practice 1-1

- Develop a processing program able to draw a polygon of n sides.
- To achieve it, implement a function `poligon` with the following parameters: center (two integers), radius (integer) and number of sides of the polygon (integer)
- Interesting additional information:
  - First, implement `setup` function, where the window size can be defined, e.g. `size(500,500)`, and then invoke the `poligon` function
  - `poligon` function can use:
    - Trigonometric functions `sin()` and `cos()` which arguments are radians
    - Function `line(x1, y1, x2, y2)` that traces a line from (x1, y1) to (x2, y2)
    - The (0,0) is placed top-left conner, positive x towards the right, positive y towards down.
    - Built-in constants such `PI` and `TWO_PI`



Practice 1-2

- Develop a *processing* program able to draw a color gradient, such that the first raw (line from side to side) will be drawn of an initial color, and the last one of a final color. Intermediate horizontal lines will be drawn following a linear graduation
- To achieve it, implement the function `gradient` with the following parameters: 6 integers; the first 3 are the RGB of the initial color (from 0 to 255), the other three are the RGB values of the final color
- Interesting additional information:
  - First, implement `setup` function, where the window size can be defined, e.g. `size(500,500)`, and then invoke the `gradient` function
  - Function `gradient` can use:
    - Linear interpolation of each component of the colors
    - Function `line(x1, y1, x2, y2)` that traces a line from (x1, y1) to (x2, y2)
    - The variables `width` and `height` return us at any time the width and height of the current application window
    - The function `stroke` that allows to change the color of the lines (any stroke), where their 3 parameters are the RGB values of the color of the line