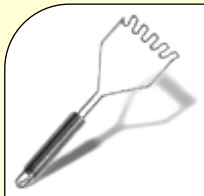
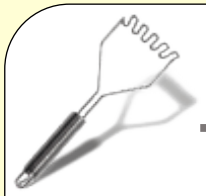
Mashups: Remixing the Web

Lecture 8: Mashups with Processing



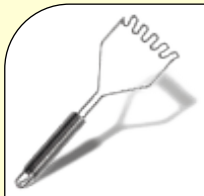
Reminders

- Assignment #3 due today
- Assignment #4 is now available
- The list of readings has been updated, and now includes all readings for the remainder of the class.
- Protected readings: login/password on the board
- "Very special" class next week:
  - Project proposals due
  - In-class project proposal presentations
  - Remember to sign up your team: http://webremix.org/teamsignup.php



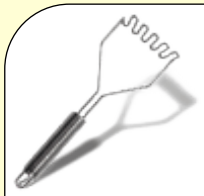
Today's Goals

- Review the basics of the Processing environment
- Understand how to produce and publish Processing applets
- Learn how to download and process data from Web APIs from within the Processing framework
- Experiment with creating some visualizations of data from Web APIs



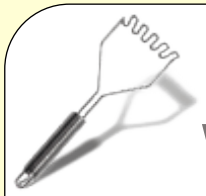
Preparing for Today's Exercises

- Download the latest version of Processing for your platform:
  http://processing.org/download/
- Download and install some Processing libraries that we will use for the lab exercises:
  - JSON library
    http://webremix.org/labs/lab8/json.zip
  - Switchboard
    http://www.realtimeart.com/switchboard/
- Other useful libraries you may be interested in:
  - proHTML
  - Yahoo! Search Library



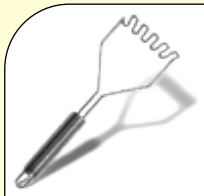
Outline

- Processing review
- Work through some Processing examples
- JSON in processing example
- Look at how to connect to a few web APIs
- Dress up our examples and add interactivity



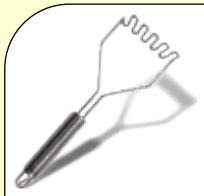
What is Processing?

- An easy-to-use Java compiler
- A development environment
- Focused on interactive graphics, sound, and animation
- Produces both locally-run programs and web-embeddable applets
- Can be used together with “real” Java



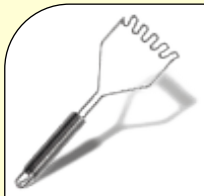
Processing Perspective

- A development tool for exploring multimedia programming
- An educational tool for learning programming fundamentals
- An ideation tool or “electronic sketchbook” for trying out ideas
- Targeted for designers, artists, beginning programmers



Nice Things about Processing

- Takes care of a lot of the annoying setup logistics for doing video and graphics in Java
- Easy to create interesting dynamic visuals programmatically
- Allows quick experimentation
- Strong focus on graphics, sound, and simple interactivity (unlike traditional Java programming with a text console)



Getting Help on Processing

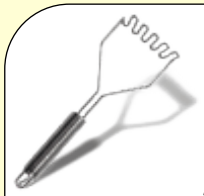
- Look at the built-in examples
- More examples:

  `http://www.processing.org/learning/`
- Function reference:

  `http://www.processing.org/reference/`
- Discussion forums:

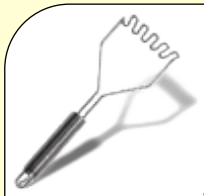
  `http://www.processing.org/discourse/`
- User-contributed code samples:

  `http://www.processinghacks.com/`



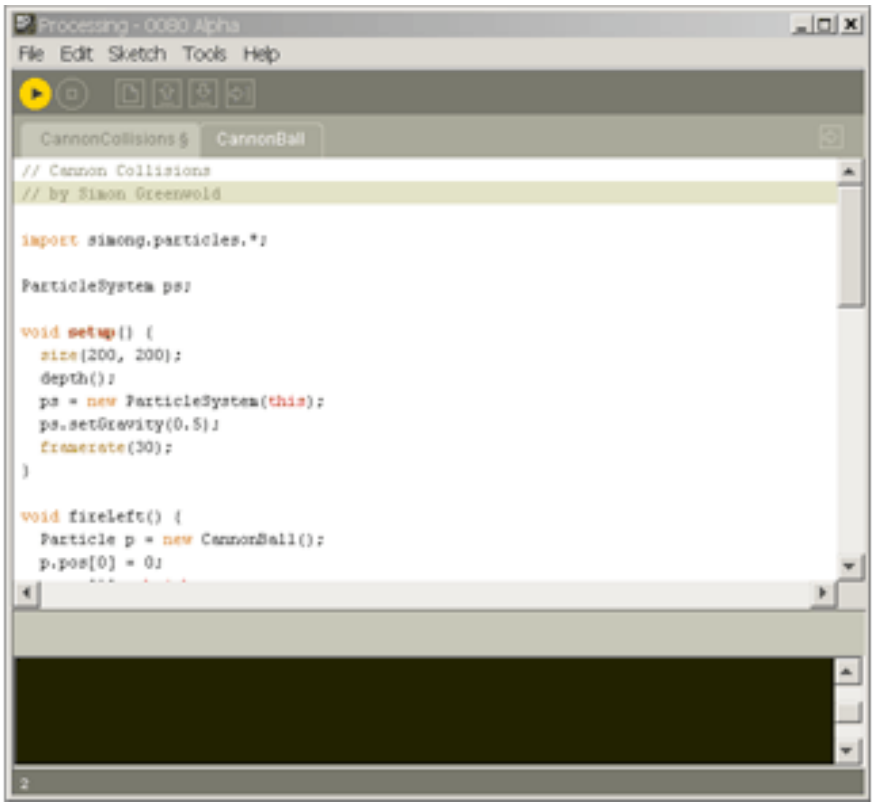
Available Libraries

- Built-In
  - Video
  - Networking
  - Serial Communication
  - Importing XML, SVG
  - Exporting PDF, DXF, etc.
- External Contributions
  - Sound: Ess, Sonia
  - Computer Vision: JMyron, ReacTIVision, BlobDetection
  - Interface: proCONTROLL, Interfascia
  - Many others



A Quick Tour

Display Window



Menu

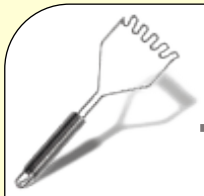
Toolbar

Tabs

Text Editor

Message Area

Text Area



Toolbar Buttons

 Run

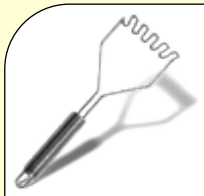
 Stop

 New

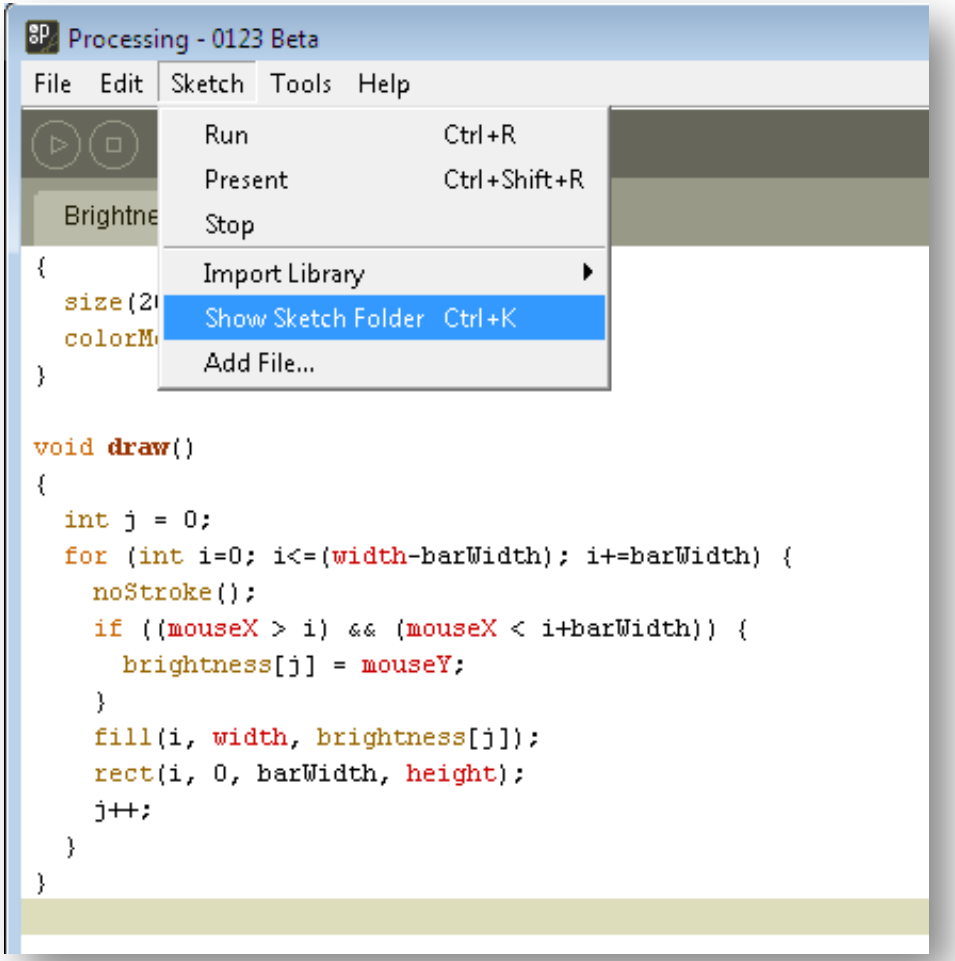
 Open

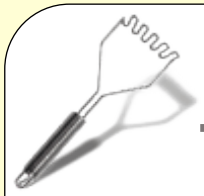
 Save

 Export



Sketches


Processing - 0123 Beta
File Edit Sketch Tools Help
Run Ctrl+R
Present Ctrl+Shift+R
Stop
Import Library
Show Sketch Folder Ctrl+K
Add File...
void draw()
{
int j = 0;
for (int i=0; i<=(width-barWidth); i+=barWidth) {
noStroke();
if ((mouseX > i) && (mouseX < i+barWidth)) {
brightness[j] = mouseY;
}
fill(i, width, brightness[j]);
rect(i, 0, barWidth, height);
j++;
}
}



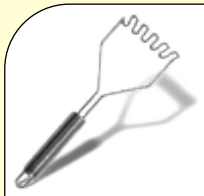
Tabs

Processing - 0123 Beta

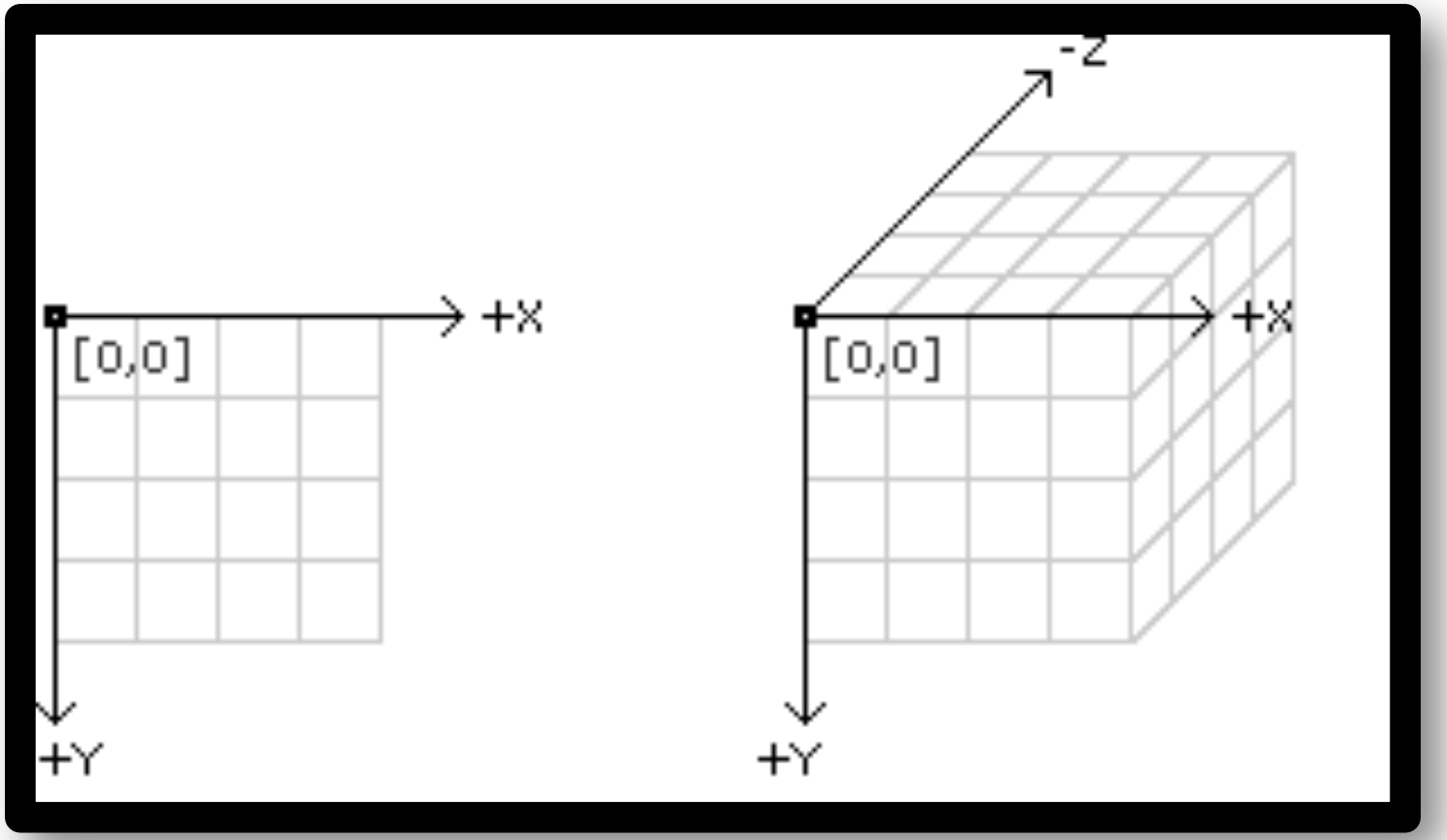
File Edit Sketch Tools Help

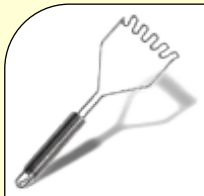
BlobDetection_JMyron § MyronSetup §

```
import JMyron.*;
import

JMyron m;//a camera object
BlobDetection theBlobDetection;
PImage img;
boolean newFrame=false;

void setup()
{
  size(320,240);
  m = new JMyron();//make a new instance of the object
  m.start(width,height);//start a capture at 320x240
  smallImg = new PImage(80, 60);
  theBlobDetection = new BlobDetection(img.width, img.height);
  theBlobDetection.setPosDiscrimination(true);
  theBlobDetection.setThreshold(0.2f); // will detect bright areas
```



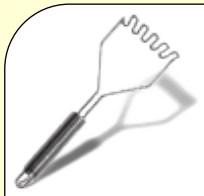
Coordinates





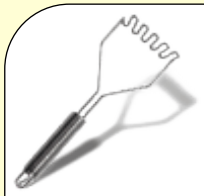
Programming Modes

- Basic
  - For drawing static images and learning programming fundamentals
- Continuous
  - Provides a setup() and draw() structures and allows writing custom functions and classes and using keyboard and mouse events
- Java
  - Most flexible mode, giving access to the full Java programming language



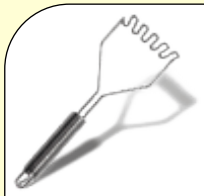
Basic Mode

```
size(200, 200);
background(255);
noStroke();
fill(255, 204, 0);
rect(30, 20, 50, 50);
```



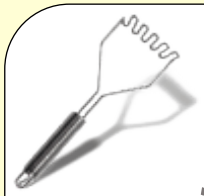
Continuous Mode

```
void setup() {
  size(200, 200);
  noStroke();
  background(255);
  fill(0, 102, 153, 204);
  smooth();
  noLoop();
}
void draw() {
  circles(40, 80);
  circles(90, 70);
}
void circles(int x, int y) {
  ellipse(x, y, 50, 50);
  ellipse(x+20, y+20, 60, 60);
}
```



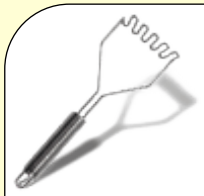
Continuous Mode

```
void setup() {
  size(200, 200);
  rectMode(CENTER);
  noStroke();
  fill(0, 102, 153, 204);
}

void draw() {
  background(255);
  rect(width-mouseX, height-mouseY, 50, 50);
  rect(mouseX, mouseY, 50, 50);
}
```



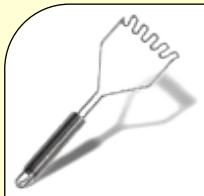
Java Mode

```
public class MyDemo extends PApplet {
  void setup() {
    size(200, 200);
    rectMode(CENTER);
    noStroke();
    fill(0, 102, 153, 204);
  }
  void draw() {
    background(255);
    rect(width-mouseX, height-mouseY, 50, 50);
    rect(mouseX, mouseY, 50, 50);
  }
}
```



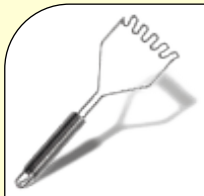
Some Basic Setup Statements

```
// specifies window size
size(200, 200);
// specifies background color
background(102);
// disables filling in shapes
noFill();
// disables drawing lines
noStroke();
// set fill color
fill(255,100,100);
// set stroke color
stroke(100,255,100);
```



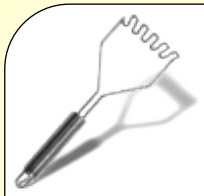
Some Basic Drawing Functions

```
// draw a point in the middle
// width and height store the
// window size
point(width/2, height/2);
// draw a 20x20 rectangle
rect(10,10,20,20);
// draw an ellipse
ellipse(50,50,30,30);
// draw an irregular shape
beginShape();
vertex(60, 40); vertex(160, 10);
vertex(170, 150); vertex(60, 150);
endShape();
```



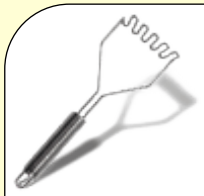
Setup and Draw

```
void setup()  {
  size(200, 200);
  stroke(255);
  frameRate(30);
}
float y = 100;
void draw() {
  background(0);
  y = (y+1) % height;
  line(0, y, width, y);
}
```



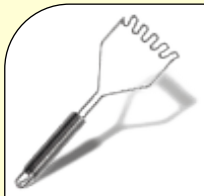
noLoop

```
void setup()  {
  size(200, 200);
  stroke(255);
  frameRate(30);
  noLoop();
}
float y = 100;
void draw() {
  background(0);
  y = (y+1) % height;
  line(0, y, width, y);
}
```



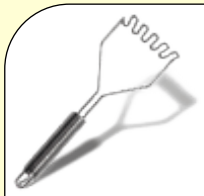
Loop

```
void mousePressed() {
  loop();
}
```



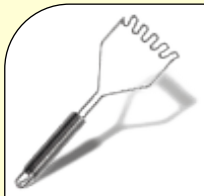
Redraw

```
void mousePressed() {
  redraw();
}
```



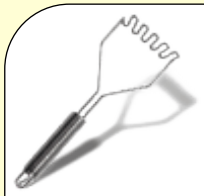
Event Handlers

```
mouseDragged()
mouseMoved()
mousePressed()
mouseReleased()
...
keyReleased()
keyPressed()
```



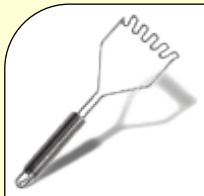
Mouse Drawing

```
void setup() {
  size(200, 200);
  background(50);
}
void draw() {
  stroke(255);
  if(mousePressed) {
    line(mouseX, mouseY,
         pmouseX, pmouseY);
  }
}
```



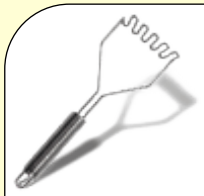
Functions

```
void draw_target(int xloc,
     int yloc, int size, int num) {
  float grayvalues = 255/num;
  float steps = size/num;
  for(int i=0; i<num; i++) {
    fill(i*grayvalues);
    ellipse(xloc, yloc,
     size-i*steps, size-i*steps);
  }
}
```



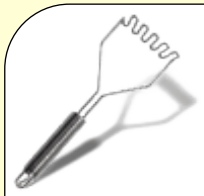
Other Basic Concepts

- These behave how you would expect (exactly as they do in Java)
  - Data types (int, float, boolean)
  - Arrays
  - Loops
  - Conditionals and Logical Operators
  - Strings
  - Variables and Scoping



Images

```
size(200, 200);
PImage img;
img = loadImage("tennis.jpg");
image(img, 0, 0);
image(img, 0, 0, img.width/10,
  img.height/10);
```



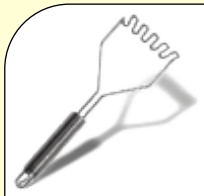
Reading JSON Data

- Load JSON by creating a JSONObject from a string (which can be loaded from a file or URL):

```
JSONObject jsonData =
  new JSONObject(jsonString);
```

- Then traverse the JSON tree using these functions:

```
data.getJSONArray("name");
array.getJSONObject(i);
obj.getInt()
obj.getDouble()
obj.getString()
```



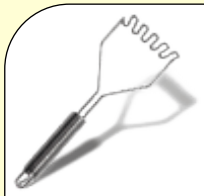
Reading XML Data

- Load XML by creating an XMLElement object from a file or a URL:

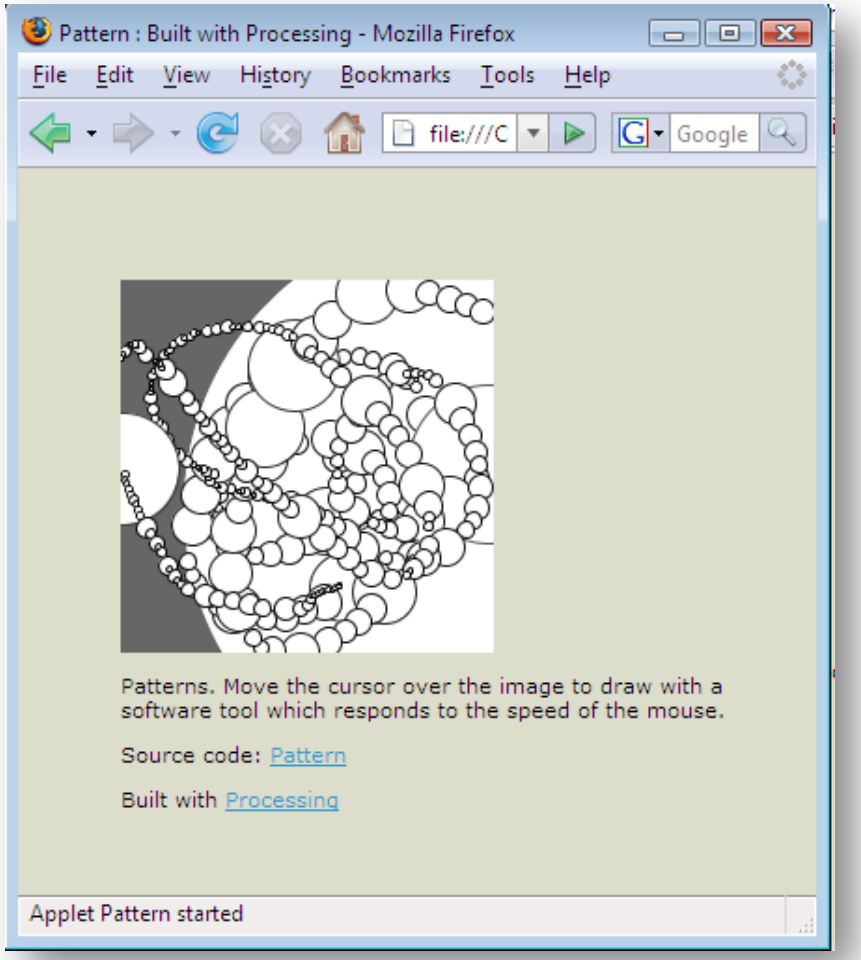
```
xml = new XMLElement(this, "sites.xml");
```

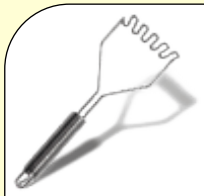
- Then traverse the XML tree using these functions:

```
getChildCount()
element.getChild()
getChildren()
XMLElement array.getContent()
getIntAttribute()
element.getFloatAttribute()
element.getStringAttribute()
element.getName()
```



Exporting an Applet





Signing an Applet

- Generate a keystore:

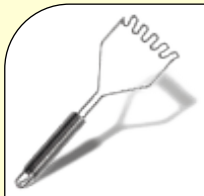
```
$ keytool -genkey -alias signFiles
  -keystore mystore
  -keypass thepassword
  -dname "CN=projname, OU=name,
   O=company, L=location, S=state,
   C=country" -storepass thepassword
```

- Export a certificate file (optional):

```
$ keytool -export -keystore mystore
  -storepass thepassword
  –alias signFiles
  -file mycertificate.cer
```

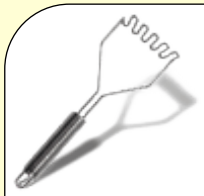
- Sign your jar file:

```
$ jarsigner -keystore mystore
  -storepass thepassword -keypass thepassword
  -signedjar output.jar input.jar signFiles
```



Lab Exercise

- Choose a web API that provides output in JSON or XML format.
- Make a query to the API in Processing, and load and parse the data.
- Display the data in an interesting graphical way, using color, animation, or interactivity.



Summary

- Processing provides a fun, easy, visual way to program interactive graphics
- Processing has built-in support for XML parsing, and there are libraries to handle JSON data and interfacing with various web APIs
- Check out the examples and take a look at the various external libraries