The Processing language

- Developed by Ben Fry and Casey Reas at MIT in 2001
- Related to the languages Logo and Java
- Free, open-source software
- processing.org contains many programming resources
- www.openprocessing.org is an online community platform devoted to sharing and discussing Processing sketches in a collaborative, open-source environment.

Basic Processing graphics functions

- `point(x,y)` — draws a point at location (x, y)
- `line(x1,y1,x2,y2)` — draws a line from (x_1, y_1) to (x_2, y_2)
- `rect(x,y,w,h)` — draws a rectangle at (x, y) that is w wide and h tall
- `ellipse(x,y,w,h)` — draws an ellipse circumscribed inside the rectangle at (x, y) that is w wide and h tall
- `triangle(x1,y1,x2,y2,x3,y3)` — draws a triangle with vertices at (x_1, y_1), (x_2, y_2), and (x_3, y_3)
- `quad(x1,y1,x2,y2,x3,y3,x4,y4)` — draws a quadrilateral with vertices at (x_1, y_1), (x_2, y_2), (x_3, y_3), and (x_4, y_4)
- `arc(x,y,w,h,`θ_1`,` θ_2`)` — draws a segment of the ellipse circumscribed inside the rectangle at (x, y) that is w wide and h tall from θ_1 to θ_2

Changing attributes in Processing

- `smooth()` — uses antialiasing to draw lines with fewer "jaggies"
- `rectMode(v)` — sets the rectangle drawign mode to v (either CENTER or CORNER).
- `ellipeMode(v)` — sets the rectangle drawign mode to v (either CENTER or CORNER).
- CENTER uses the (x, y) coordinate as the center of the rectangle
- CORNER uses the (x, y) coordinate as the upper left corner of the rectangle

Changing attributes in Processing

- `size(w,h)` — sets the output window size to be w pixels wide and h pixels tall (should always be first)
- `background(n)` — sets the background color to be gray with intensity $0 \leq n < 256 = 2^8$, same as `background(n,n,n)`
- `background(r,g,b)` — sets the background color to be the color represented by the (r, g, b) triple
- `stroke(n)` — sets the stroke color to be gray with intensity $0 \leq n < 256 = 2^8$, same as `background(n,n,n)`
- `stroke(r,g,b)` — sets the stroke color to be the color represented by the (r, g, b) triple
- `fill(n)` — sets the fill color to be gray with intensity $0 \leq n < 256 = 2^8$, same as `background(n,n,n)`
- `fill(r,g,b)` — sets the fill color to be the color represented by the (r, g, b) triple
- `noFill()` — sets the fill to be transparent
- `noStroke()` — no boundaries drawn on shapes

Transparency in Processing

- `background(r,g,b,a)` — sets the background color to be the color represented by the (r, g, b) triple
- `stroke(r,g,b,a)` — sets the stroke color to be the color represented by the (r, g, b) triple
- `fill(r,g,b,a)` — sets the fill color to be the color represented by the (r, g, b) triple

- The value of $0 \leq a < 256$ represents the opacity of the color, where 0 is

Quirks of the Processing Language

- a sketch is a Processing program
- a sketch name cannot contain spaces, and must start with a letter
- a sketch is stored in a folder named *sketch* as *name.pde*
- a reserved word is colored special in Processing
- Processing is case sensitive: Line and line are different!

- println() prints information to the message window, followed by a newline character

Comments

- Comments are very important in improving code readability
- Processing use two types of comments
  - // is an end of line comment
  - /* . . . */ is a multi-line comment

Running a Processing program

- Play button executes code
  - code is converted from Processign to Java
  - Java is converted from text to a Java bytecode
  - The Java bytecode is executed on the Java virtual machine

Blocks of code

- Processing uses curly braces, { and }, to group sections of code into an entity called a block
- We can nest block inside of blocks to an arbitrary level
- Convention dictaces we indent inner nested blocks with respect to their outer blocks

```
{
// A block of code
    {
    // A nested block of code
    }
}
```

`setup()` and `draw()`

- Processing allows us to write a simple program without blocks
- More complex programs will use multiple named blocks, called functions
- Two standard functions are `setup()` and `draw()`

```
void setup() {
// Initialiaztion goes here
}

void draw() {
// dynamic drawing code goes here
}
```

- Parentheses indicate we have a function, in these cases the functions have no parameters
- `void` indicates the function returns nothing
- More about functions shortly

Execution of `setup()` and `draw()`

```
void setup() {
// Step 1a
// Step 1b
}

void draw() {
// Step 2a
// Step 2b
}
```

- Executed as: 1a, 1b, 2a, 2b, 2a, 2b, 2a, 2b, ...

Mouse positions

- Mouse x and y position is indicated by the keywords `mouseX` and `mouseY`

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

// What needs to be here vs. in setup?
void draw() {
background(255); // erases
stroke(0);
fill(175);
rectMode(CENTER);
rect(mouseX, mouseY, 50, 50);
}
```

Timing of Processing's graphics display

- Processing writes the graphics your request in the `draw()` function to a buffer
- After `draw` has been called, the newly updated graphics buffer is displayed.
- This technique is called double-buffering

Previous mouse positions

- The previous mouse x and y position is indicated by the keywords `pmouseX` and `pmouseY`

Handling mouse and key events

- When a mouse button is pressed, Processing calls a `mousePressed()` method, if it exists in your program
- When a keyboard key is pressed, Processing calls a `keyPressed()` method, if it exists in your program