Intro to Java with Art!

For SMASH CS Fundamentals III: Object-Oriented Programming

Goal: Write at least 3 methods that draw shapes, pictures, or patterns. Call these in setup() to see how they work.

Welcome to Java!

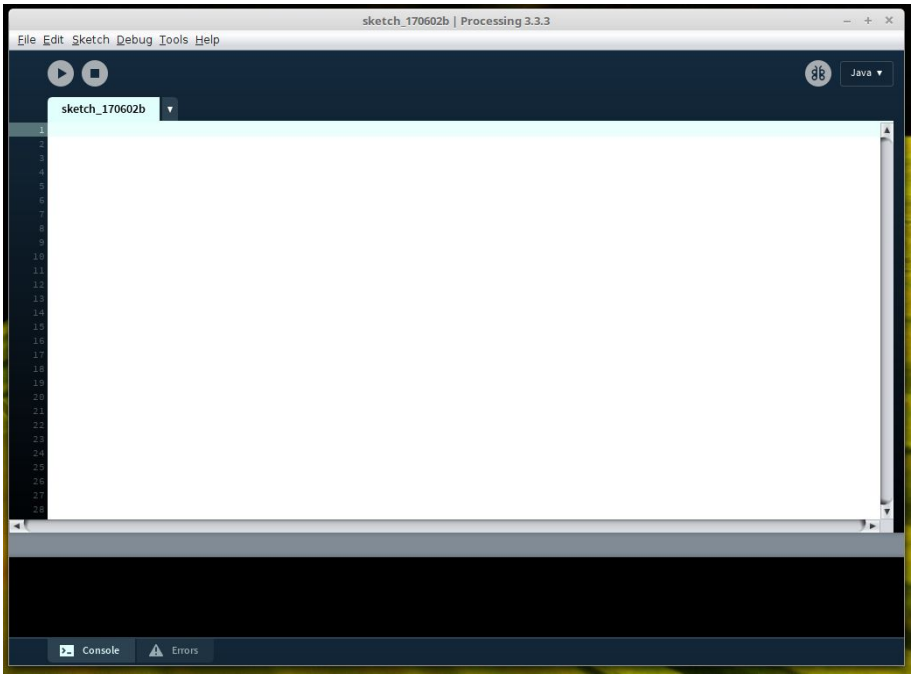
Java is a popular programming language that anyone can learn. It's a ton of fun! You will learn basic Java syntax through the visual arts implementation known as Processing.

Programming will be very confusing at first. But stick with it! Just like riding a bike, you're going to fall off quite a bit until you get the hang of it.

Be patient with yourself, stay focused, and ask your classmates and teachers for help if you get stuck! Most importantly, have fun!

Getting Started

Open up the Processing application on your computer. Once it's open, you should see this:



Take a look at the interface a bit. There are two windows:

- **Code Window: The window at the top is where all your coding will happen. Right above this are two buttons that should be recognizable: a "play/run" button and a "stop" button.**
- **Console Window: The window at the bottom is where you'll see your code's output along with details about any errors encountered.**

Feel free to resize any of these windows by clicking and dragging the dividers between them.

Now we're ready to start drawing!

1) Setup and API

The first step for creating any visual art is determining how big your canvas is going to be.

So let's do that. Type this into your code window exactly as it appears here, then click the run button.

```
size(400, 400);
```

What happened? A box popped up. Yeah, that's your canvas!

Replace your code with this:

```
size(200, 400);
```

See that? You now have a longer canvas.

```
size(400, 200);
```

Your canvas is wider!

I think you're getting the picture (pun intended). It looks like this piece of code takes two arguments or inputs. The first is represents the width, the second the height. So we might say in general it looks like this:

```
size(w, h);
```

This is an example of a method, or more generally a function. You'll hear both terms used throughout your time learning how to program, but with Java we'll usually use the term "method" for reasons you'll learn later on.

Methods are like commands. You write your method, click run, and the Processing IDE (Integrated Development Environment) is like, "Okay boss!" and does what you want it to do.

Well, it *should*, **anyway.**

Try this, exactly as it is written:

```
size(400, 400)
```

Got an error! Something about "null". We don't know what that is implying yet, so don't worry about it. What do you notice different about this code though? Something is missing…

The semicolon! That's right. You need semicolons after statements, which are essentially "code sentences". In this case the code sentence is simply the command "Change the size of my canvas.", only we left out the "period".

Okay, so I'll add my semicolon. Now let me try this:

```
size(300);
```

What the… you got an error! Something about an array… and an index… hmm… maybe you'll learn about that stuff later.

But for now, we just need to get rid of that error! What did we do wrong? Well, the `size()` method expects two arguments. The parameters or "placeholder variables" `w` and `h` are there to help us know what goes in. In this case, we've only supplied one argument. So a simple fix would be to just place two comma-separated values inside, just like we did before.

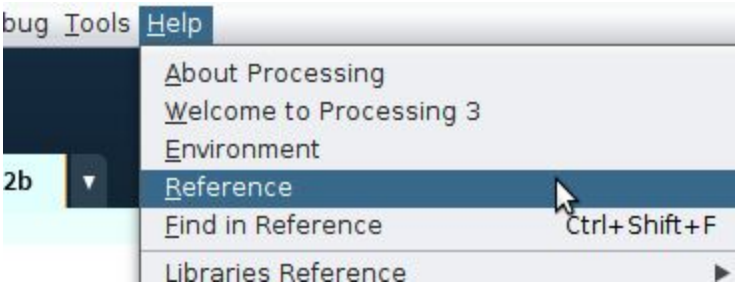
But wait… how would we have known this? When we opened our Processing IDE all we had was a blank text editor!

That's where the documentation comes in. Programming languages like Java are composed of a bajillion libraries, with tons of methods that have already been written for programmers to use. That way programmers don't need to go reinvent the wheel each time they wish to, say, resize a canvas! They just need to go read the documentation, call the method, and voila, the canvas is resized!

This documentation is essentially the instruction manual for the library or language, and is otherwise known as the Application Programming Interface or API.

So where can I find the Java API? Settle down, young one. Those waters are deep and the currents are strong. For now, let's just look at the Processing API, which is much more simplified (and the reason we're starting here instead of jumping into full blown Java right away!).

The Processing API is conveniently located right on your computer!



1. **Go there, press ctrl+f on your keyboard, type in "`size()`", and press Enter.**
2. **Click on the `size()` method and you'll see everything you need to know about this method!**

Even the Processing API is rather large, so here is a "cheat sheet" version, courtesy of Dr. David Matuszek at the University of Pennsylvania (he was my professor!). These are some of the more important methods you'll need to draw simple pictures:

https://github.com/ahob85/smash/blob/master/cs3/draw/cheat_sheet.txt

You should bookmark or download that so you can have it as a reference!

2) Draw a line

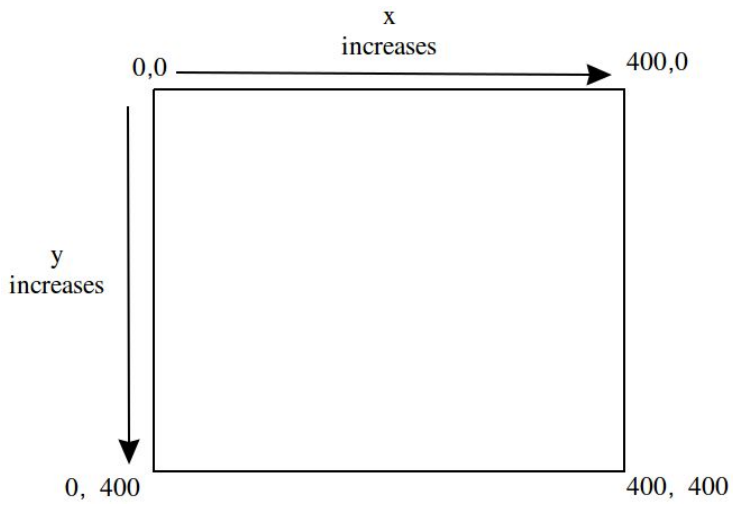
Try this:

```
size(400, 400);
line(100, 100, 200, 200);
```

We drew a line! Specifically, we drew a line from (x1, y1) to (x2, y2). Just like geometry! Only with some important differences:

1. **There are no negative axes. Hence, no negative values. At least, if a point is positioned "negatively" then you simply won't be able to see it.**
2. **The origin (0, 0) begins at the top-left corner, not the bottom-left. Which means...**
3. **The y-axis is flipped upside down. That's right: y-values begin at 0 and grow as you go down the y-axis!**

To give you a visual, here's your canvas, annotated with the coordinate system:



3) Draw a square

Try this:

```
size(400, 400);
line(100, 100, 200, 100);
line(200, 100, 200, 200);
```

```
line(200, 200, 100, 200);
line(100, 200, 100, 100);
```

Cool huh?

By the way, what we did was create an algorithm, or series of steps to accomplish a goal. The goal in this case was to draw a square.

Wouldn't it be kind of annoying to have to type all that each time we wanted to draw a square though? What if we could create our own method to do it for us?

4) Write a method

Let's write a method to draw squares for us. Brace yourself, there's a lot to unpack here:

```
void drawSquare(int x, int y, int size) {
  line(x, y, x + size, y);
  line(x + size, y, x + size, y + size);
  line(x + size, y + size, x, y + size);
  line(x, y + size, x, y);
}
```

Don't run this yet. Let me explain:

- **"`void`" is the return type. We use "void" here because this method doesn't return anything. You'll learn what "return" means later in the course.**
- **"`drawSquare`" is the method name. Just like `size()` and `line()` have names that make sense, ours does too.**
- **The stuff in the `parentheses` is the parameter list. Just as with `size()`, we have certain values we are expecting to be passed in as arguments. In this case, we expect three `integer` values `x, y,` and `size`.**
- **The rest of the stuff inside the curly braces {} is the method body. This is the "guts" if you will, or what happens each time the method is called.**

Are you ready to call this method? Well, make sure entire code window looks just like this. Notice I've added something:

sketch_170602b

```
1  void drawSquare(int x, int y, int size) {
2    line(x, y, x + size, y);
3    line(x + size, y, x + size, y + size);
4    line(x + size, y + size, x, y + size);
5    line(x, y + size, x, y);
6  }
7
8  void setup(){
9    size(400, 400);
10   drawSquare(100, 100, 100);
11 }
```

The `setup()` method is called automatically by Processing when you press "play". From now on, if we want stuff to happen only once, we'll place it within that method.

However, we define our own methods outside of `setup()`, just as we did above.

Run that and watch the magic unfold. Notice how I'm using math to figure out where each point of the square should be. For example, I know my top-right point should be at `(x + size, y)`. It's just simple algebra.

5) Draw a bunch of squares.

Methods let us be lazy, and that is what programming is all about. Working hard now so you can be lazy later.

Behold the power of methods. You can draw a bunch of squares now very ~~lazily~~ *conveniently*.

```
void setup(){
  size(400, 400);
  drawSquare(100, 100, 100);
  drawSquare(50, 50, 200);
  drawSquare(150, 150, 50);
}
```

Remarkable.

6) Draw a bunch of triangles.

Let's try it the hard way first. Don't delete your drawSquare() method, but instead delete everything inside setup and replace it with this:

```
void setup(){
  size(400, 400);
  line(100, 100, 50, 150);
  line(50, 150, 150, 150);
  line(150, 150, 100, 100);
}
```

See that? You wrote the same number of lines of code as when you drew three squares… only here you got one triangle. Kinda lame!

So we need to do it the lazy way: write a `drawTriangle()` method!

I'm not going to help you on this one though. Let's see if you can figure it out on your own, based on how we wrote the `drawSquare()` method.

Give it a shot! Then try it out like this. Note that the size is still how long each line should be (so the triangle is equilateral).

```
void setup(){
  size(400, 400);
  drawTriangle(100, 100, 100);
  drawTriangle(200, 200, 200);
}
```

7) Reinventing the wheel sometimes teaches you stuff.

Oh snap! The API already has methods for creating shapes.

Taken straight from the cheat sheet:
rect(x, y, width, height); // (x, y) is top left corner
ellipse(x, y, width, height); // (x, y) is the center of the ellipse

triangle(x1, y1, x2, y2, x3, y3); // triangle

So why did I have you do some of that from scratch?

To teach you stuff. Sometimes reinventing the wheel is a good thing because you can learn from it. Don't you feel smart? You did almost the same thing the folks who invented Processing did.

Pat yourself on the back.

But now we'll just use the API. ^_^

8) Draw a few squares using a loop

We're really getting into it now. Let's draw some squares using a loop. And we'll use the API for drawing the squares, so don't worry.

A loop is basically a bunch of code you want to repeat a certain number of times. There are a few types of loops in Java. Here's what's called a "while loop". You'll go into far more detail later in the course about how this thing works. So don't feel bad if you don't understand it completely. For now let's just get our hands dirty and try it!

```
void setup(){
  size(400, 400);
  int counter = 0;
  int x = 0;
  while(counter < 6){
    rect(x, 100, 50, 50);
    counter++;
    x += 50;
  }
}
```

Whoa nelly. What's going on here? Let's walk through the algorithm:

1. **We set an integer variable called `counter` to 0. We also set another integer variable called `x` to 0. The "=" sign accomplishes this "setting" stuff for us.**

2. **The while loop will continue so long as the statement within the parentheses is true. So ask yourself… is counter less than 6?**
3. **It is! So we go inside the loop and do what it says.**
4. **We draw a rectangle with those specifications. See the cheat sheet entry for details.**
5. **We increase the counter variable by 1. That's what the `++` means. Now counter holds the value 1.**
6. **We increase the x variable by 50. That's what the `x += 50` means. Now x holds the value 50.**
7. **Go back up to the "while" part and ask again, is counter less than 6? Well, counter holds the value 1. Is 1 less than 6? Yep! So…**
8. **We draw a rectangle with those specifications… (i.e., go back to #4).**
9. **And so on and so forth!**

Hence, the loop continues until `counter` is NOT less than 6 (i.e., counter is greater than or equal to 6).

Using loops, you can do all kinds of cool patterns. As a matter of fact...

9) Write some pattern-drawing methods.

Here's one for you:

```
void squarePattern(int x, int y, int w, int h, int numSquares){
  int counter = 0;
  while(counter < numSquares){
    rect(x + counter * 2, y + counter * 2, w + counter * 2, h + counter * 2);
    counter++;
  }
}

void setup(){
  size(400, 400);
  squarePattern(50, 50, 100, 100, 10);
}
```

Your turn!

10) Draw shapes, patterns, and pictures with functions and loops

Your goal is to write at least 3 more methods that draw shapes, pictures, or patterns. Call these in setup() to see how they work.

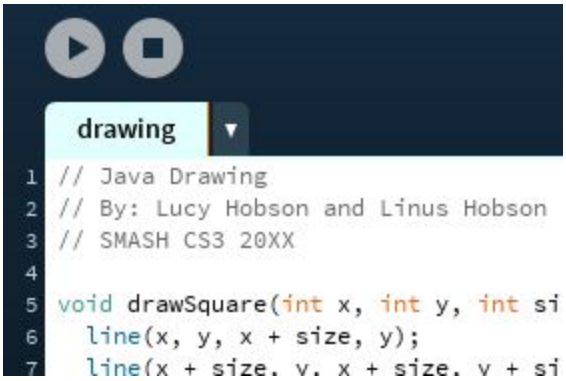
If you still can't figure out how to do loops, don't sweat it. Just do non-loopy methods instead. You'll have more time to learn loops later.

Play around and get dirty with your program. You won't learn anything by giving up or blindly copying others.

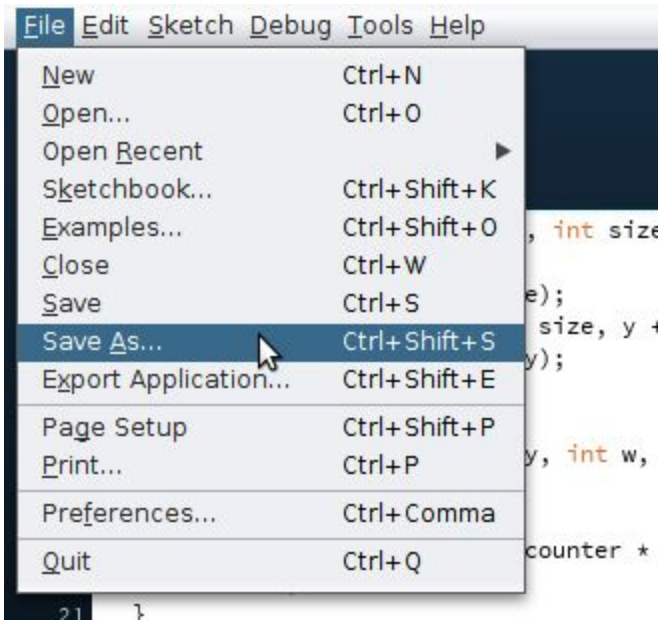
That being said, your teacher and classmates are here to help you. And don't forget the splendid "cheat sheet" API! Why not try different colors using fill(), for example?

11) Name, Save, and Submit

When you're satisfied, place the program's title, your name (or if you have a partner, both of your names), and the name of your course/year at the top of the file this:



Then save your program as "Java Drawing" sketchbook file and ask your teacher how/if you should turn it in:



That's all! Yeeeeeeah!!!