creativity
& computation
lab

week 14 || intro to openFrameworks

review

WHERE WE HAVE BEEN

What we have done:

Midterm presentations!

// Woohoo!

agenda

WHERE WE ARE GOING

What’s on for today:

Finish presentations

// keeping strict time!

oF vs. Processing

// what happens behind the scenes

Install Xcode/Code::Blocks and openFrameworks

// smoothly, I hope

Structure of oF

// more files than we’re used to, but we will like this

Creating an app

// FUN!

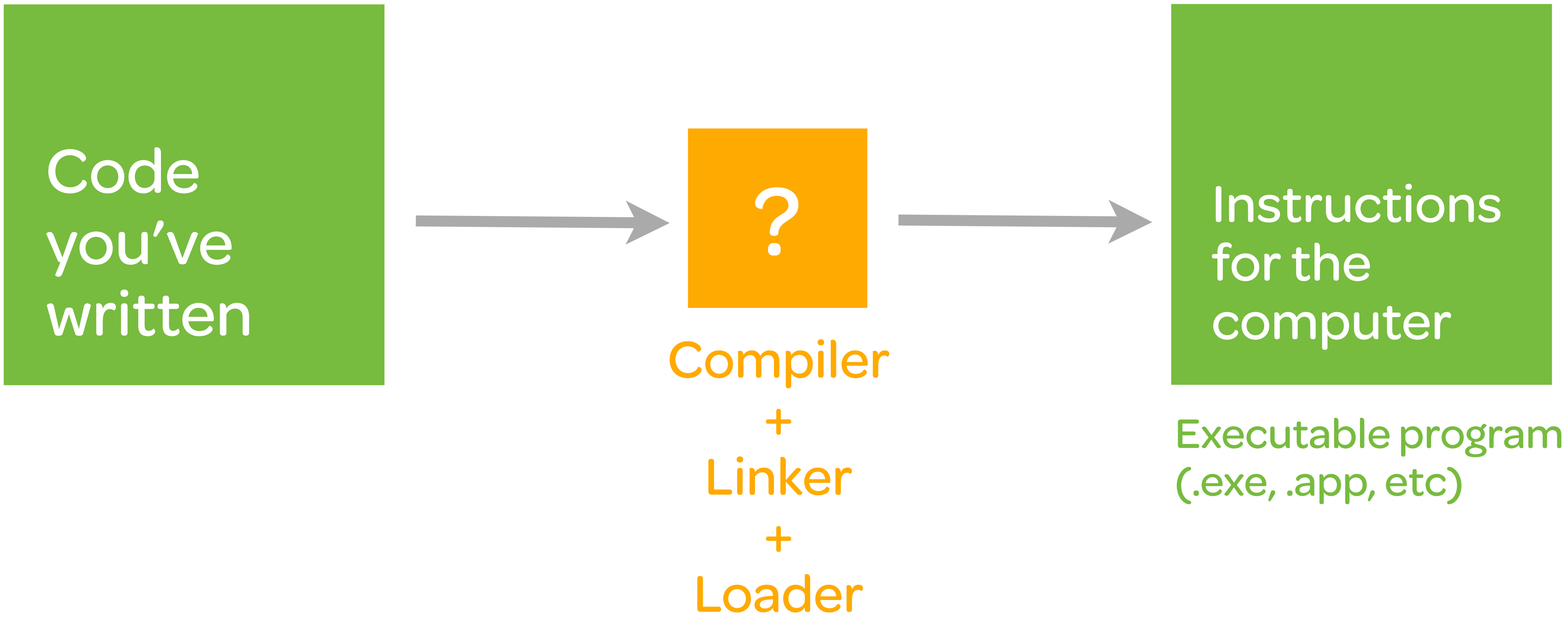
programming

WHAT IS IT AGAIN?

We all know that code is essentially a series of instructions we write to tell the computer what to do.
//Remember our tooth brushing example?

how it works

COMPILERS, LINKERS, AND LOADERS, O MY!



Code you've written

?

Compiler
+
Linker
+
Loader

Instructions for the computer

Executable program
(.exe, .app, etc)

processing

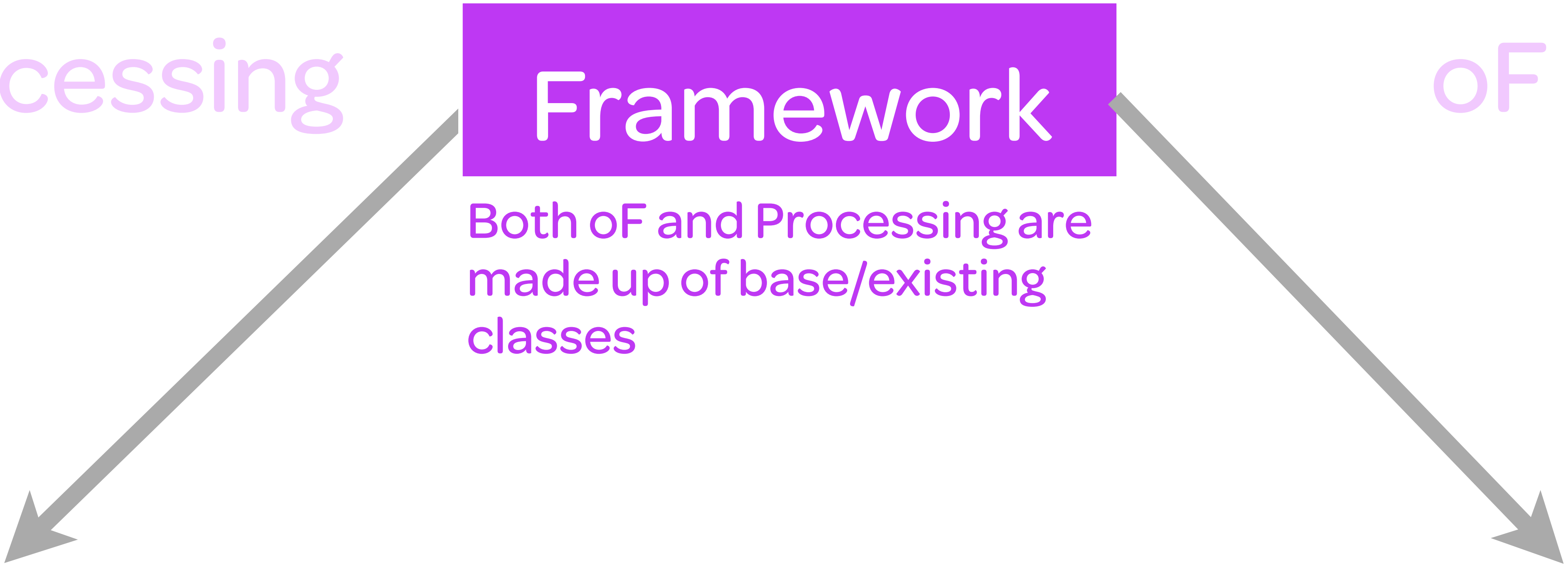
Framework

oF

Both oF and Processing are made up of base/existing classes

Processing is actually an engine running / extending a Base Class

When you write draw() and setup() functions, you are re-defining the draw() and setup() functions that are in the Processing base class.



processing

Framework

oF

Both oF and Processing are made up of base/existing classes

Processing is actually an engine running / extending a Base Class

When you write draw() and setup() functions, you are re-defining the draw() and setup() functions that are in the Processing base class.

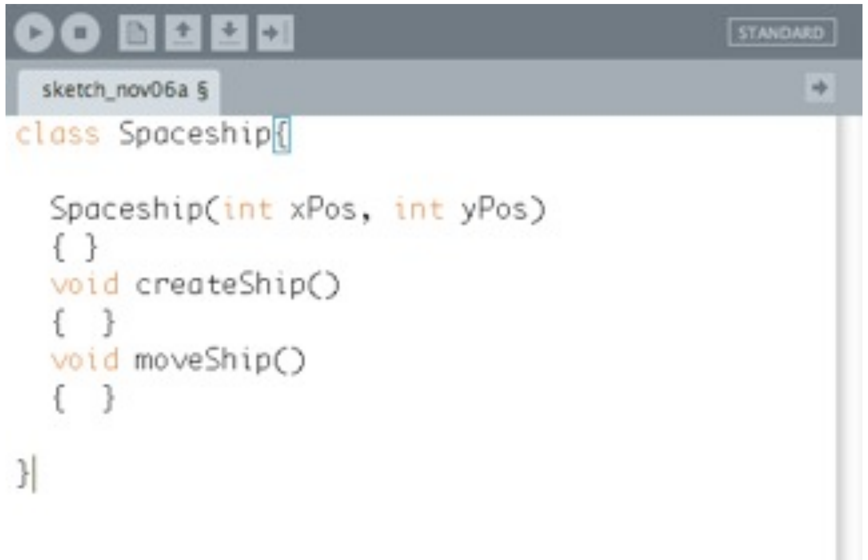
OpenFrameworks is also extending a series of existing classes, but it makes it more obvious that it's doing so.

processing

Framework

oF

Both oF and Processing are made up of base/existing classes



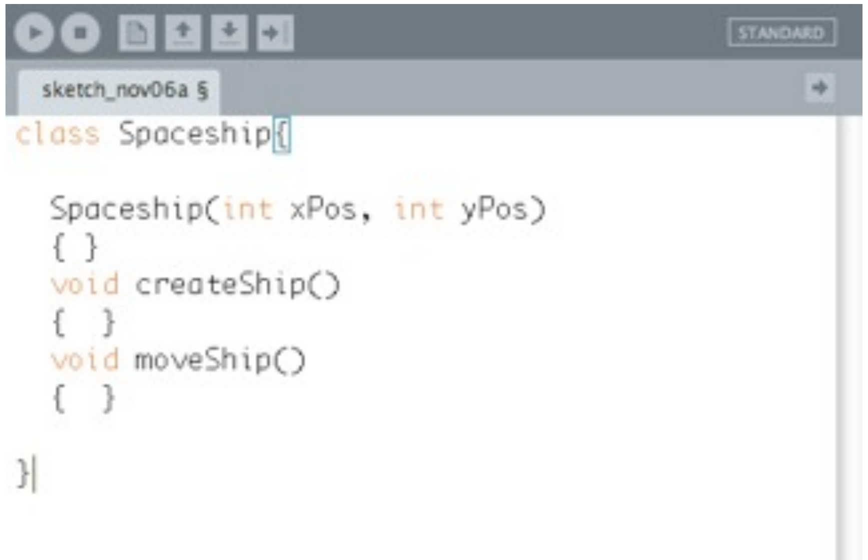
```
class Spaceship{

  Spaceship(int xPos, int yPos)
  { }
  void createShip()
  {  }
  void moveShip()
  {  }

}
```

processing

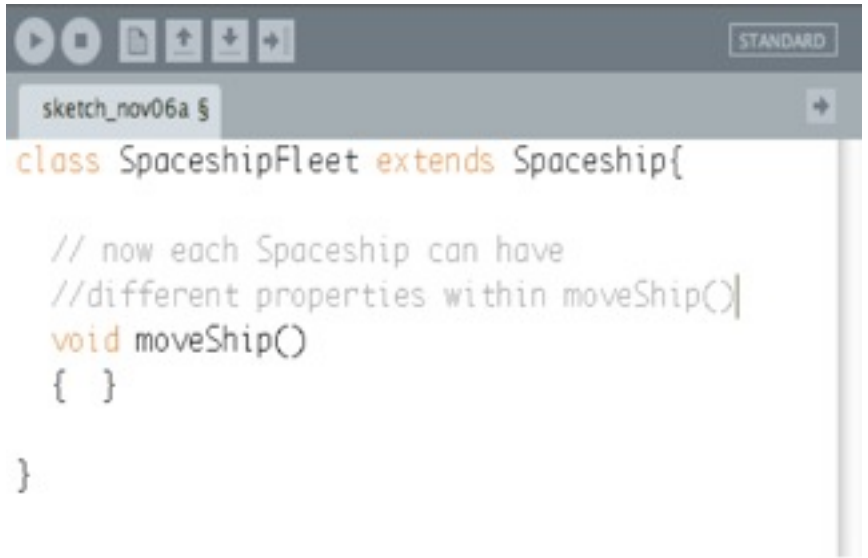
Framework

oF

Both oF and Processing are made up of base/existing classes



```
class Spaceship{

  Spaceship(int xPos, int yPos)
  { }
  void createShip()
  {  }
  void moveShip()
  {  }

}
```



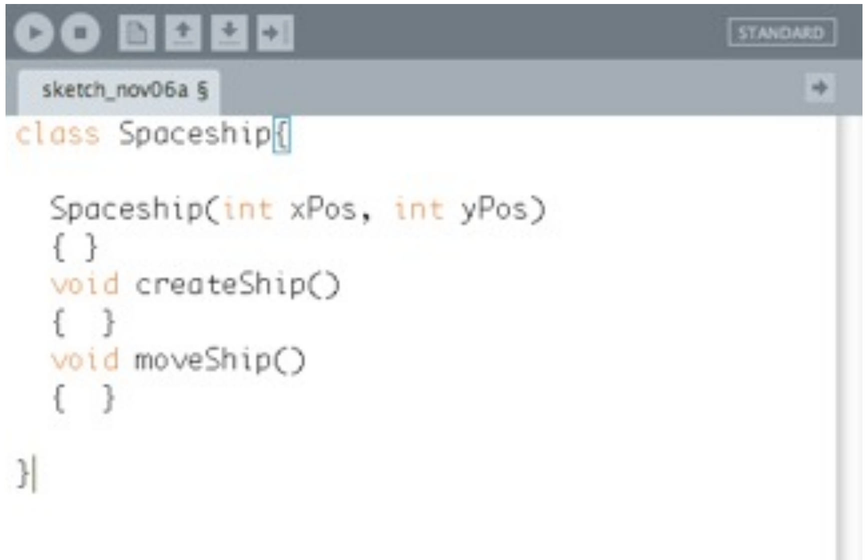
```
class SpaceshipFleet extends Spaceship{

  // now each Spaceship can have
  //different properties within moveShip()
  void moveShip()
  {  }

}
```

processing

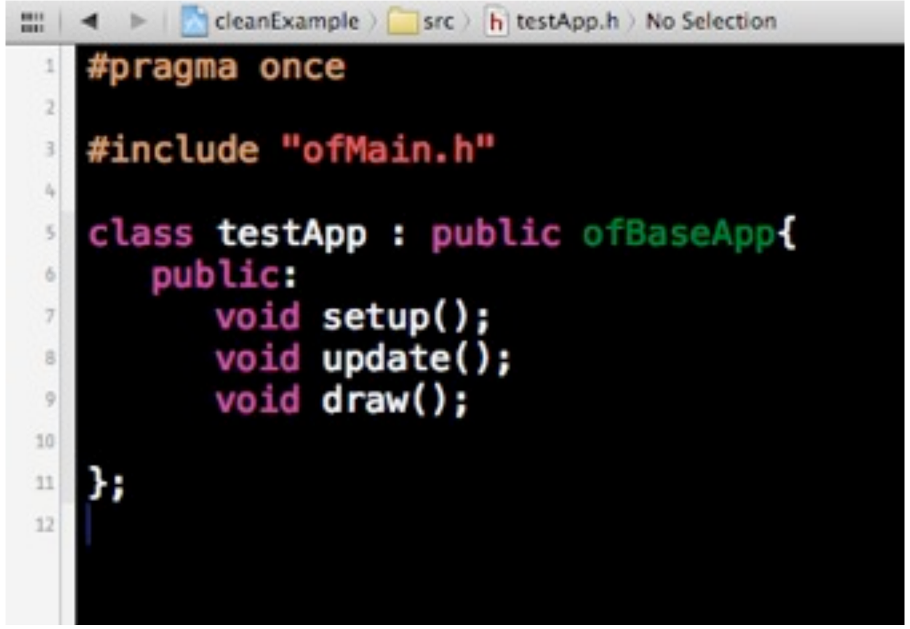
Framework

oF

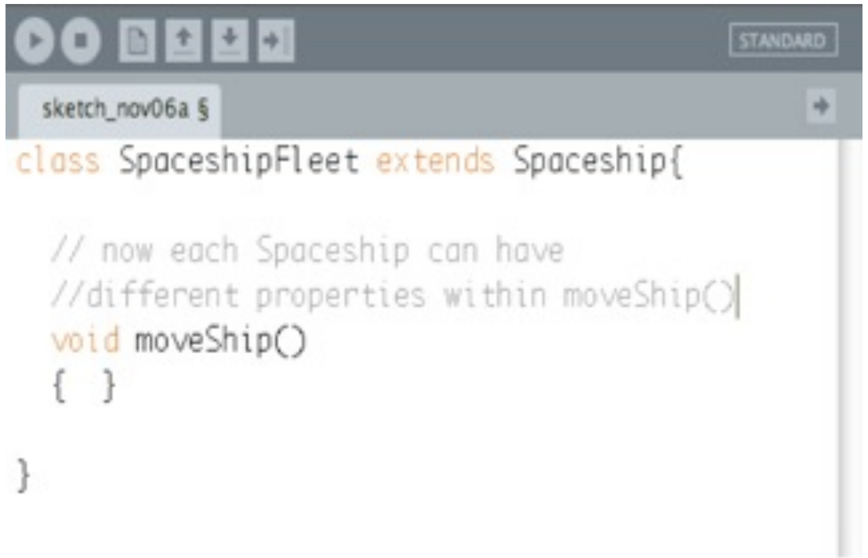
Both oF and Processing are made up of base/existing classes



```
class Spaceship{

  Spaceship(int xPos, int yPos)
  { }
  void createShip()
  {  }
  void moveShip()
  {  }

}
```



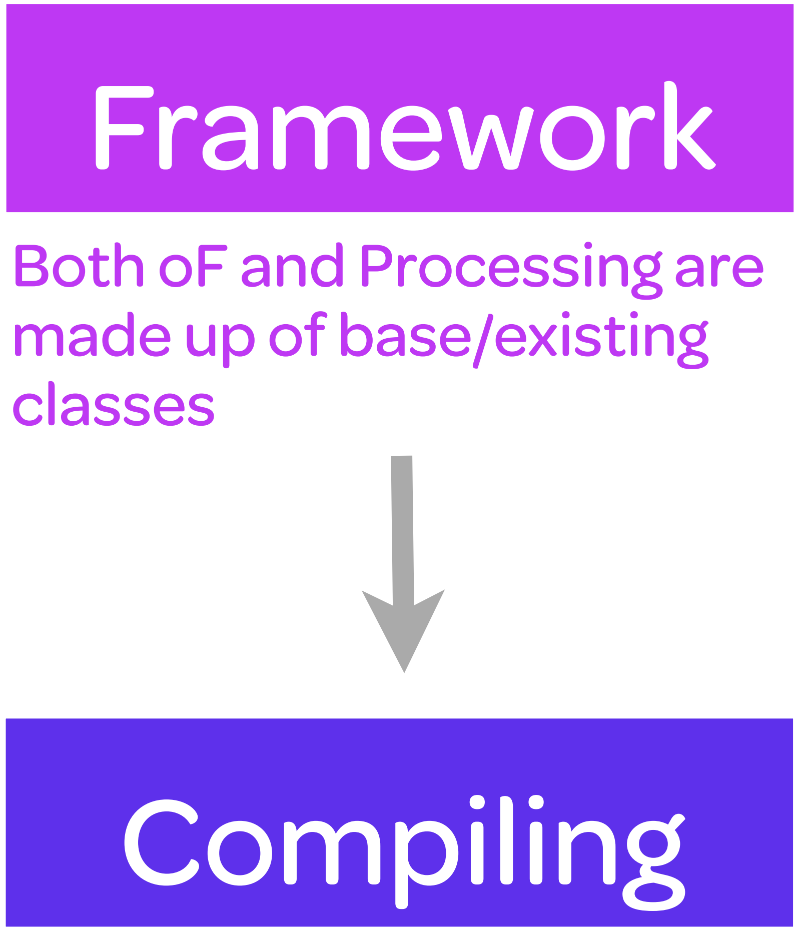
```
#pragma once

#include "ofMain.h"

class testApp : public ofBaseApp{
   public:
      void setup();
      void update();
      void draw();

};
```



```
class SpaceshipFleet extends Spaceship{

  // now each Spaceship can have
  //different properties within moveShip()
  void moveShip()
  {  }

}
```

processing

oF



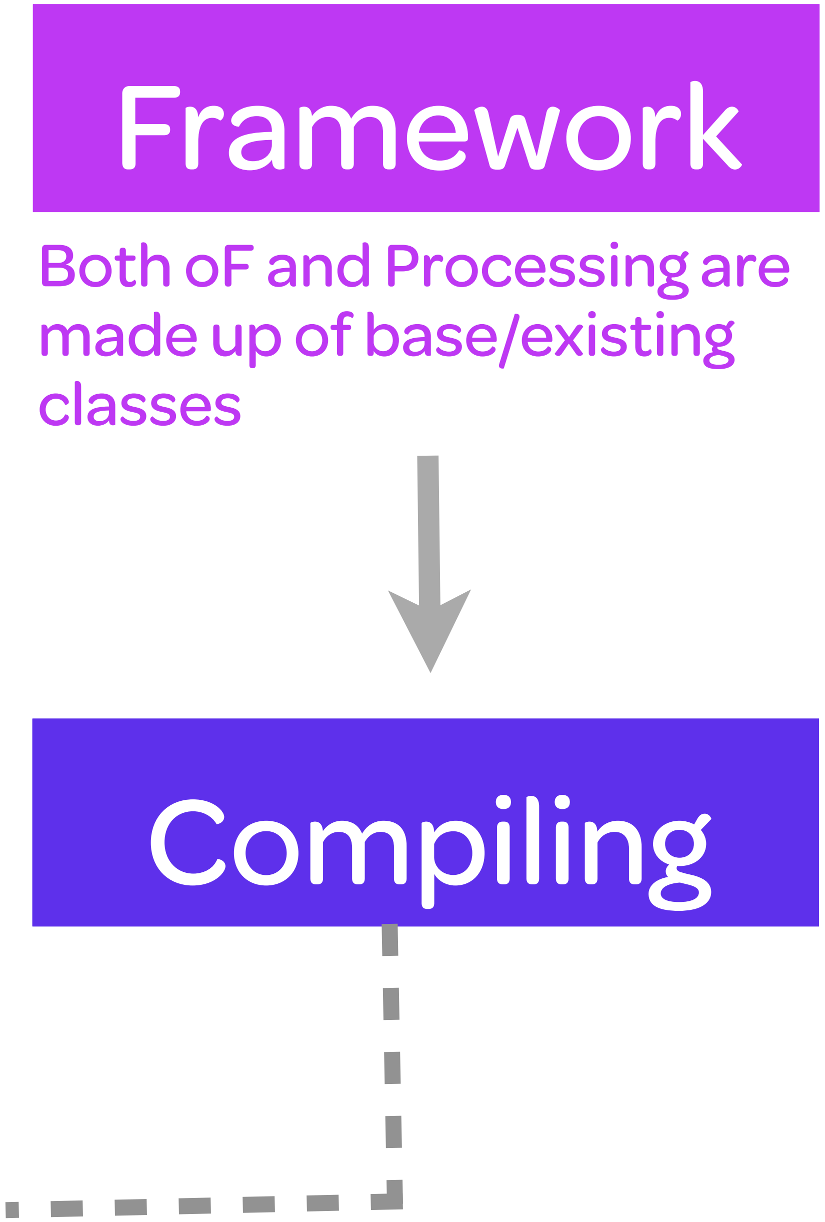
Framework

Both oF and Processing are made up of base/existing classes

Compiling

processing

oF



Framework

Both oF and Processing are made up of base/existing classes

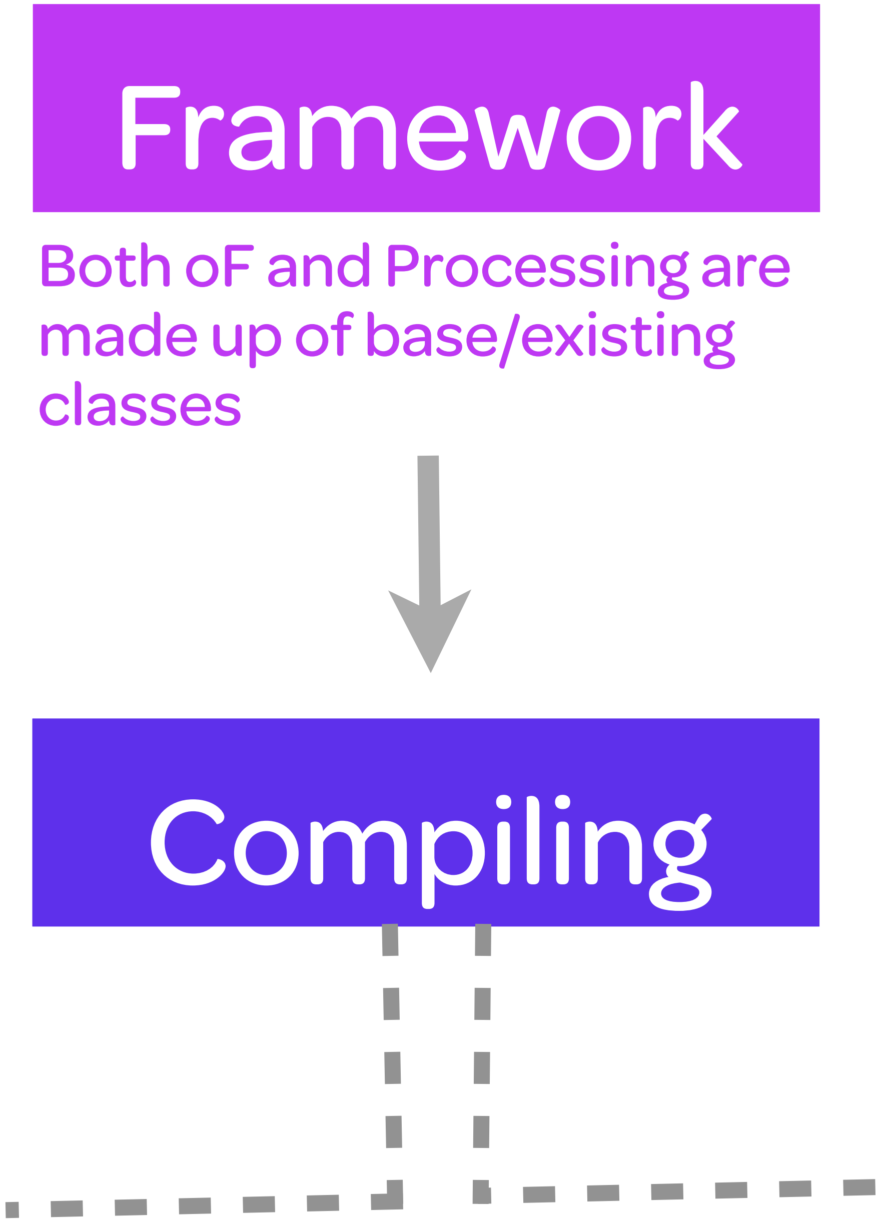
Compiling

In Java, each time you compile, your entire program is run through and changed into byte code.

Then when you run your program, a Java interpreter does runtime compilation.

processing

oF



Framework

Both oF and Processing are made up of base/existing classes

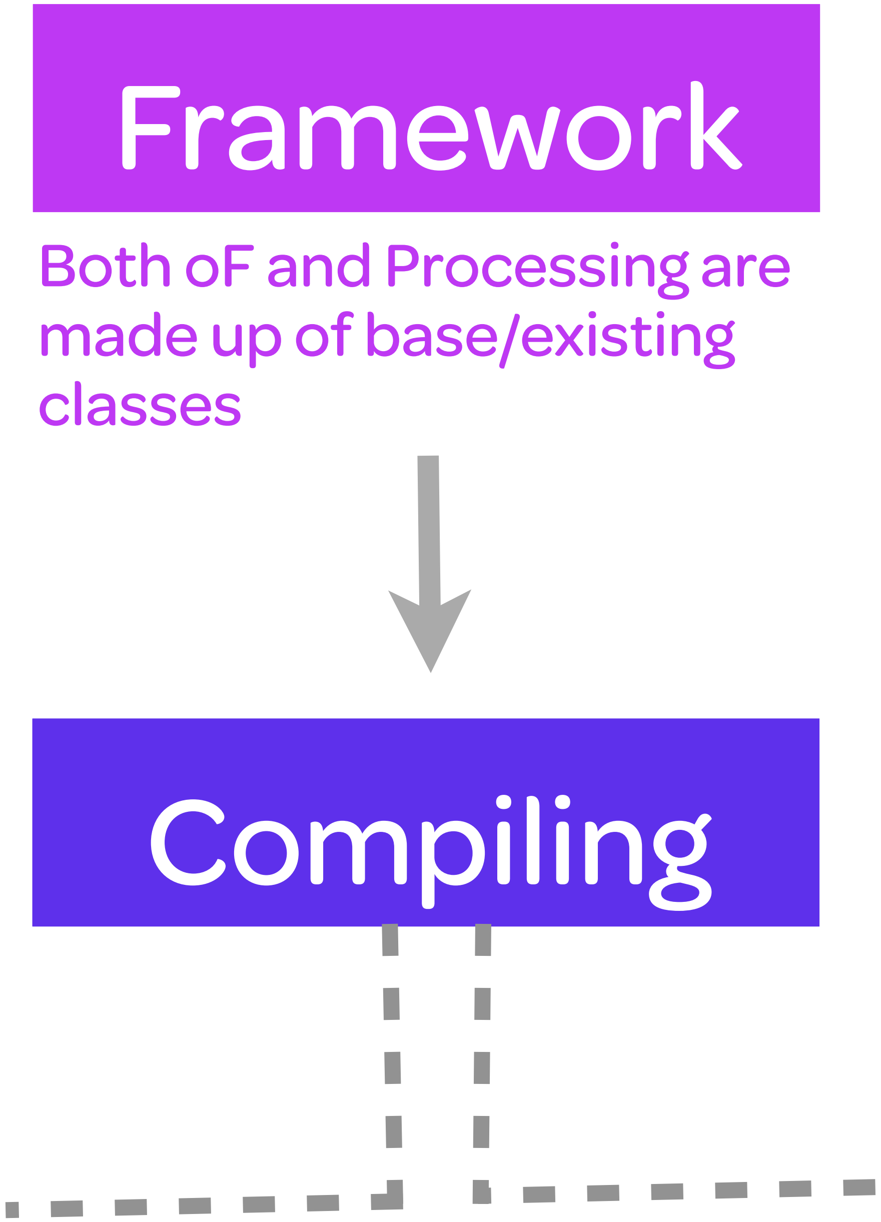
Compiling

In Java, each time you compile, your entire program is run through and changed into byte code.

Then when you run your program, a Java interpreter does runtime compilation.

The compiler:
(1) goes through all your #include statements and copy/pastes chunks of your code to create one file.

processing



Framework

oF

Both oF and Processing are made up of base/existing classes

Compiling

In Java, each time you compile, your entire program is run through and changed into byte code.

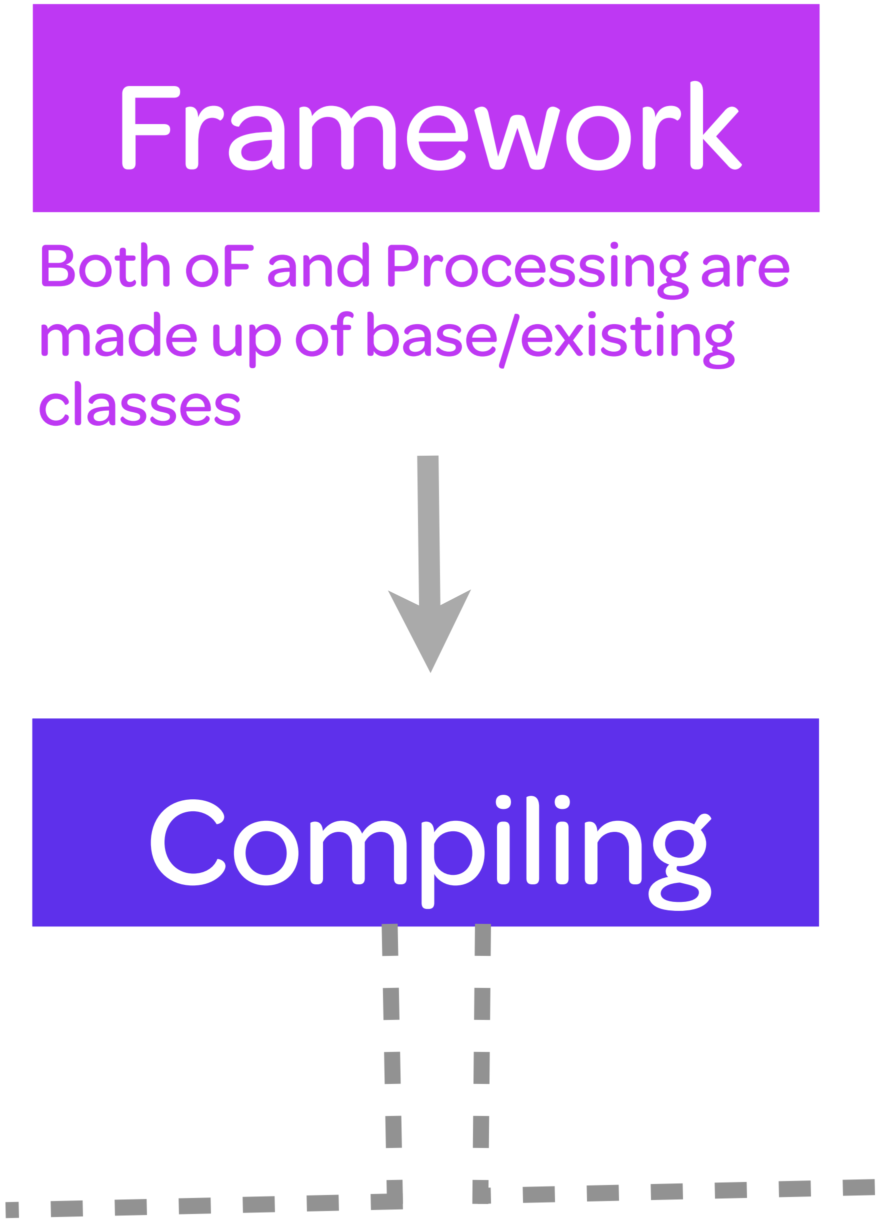
Then when you run your program, a Java interpreter does runtime compilation.

The compiler:
(1) goes through all your #include statements and copy/pastes chunks of your code to create one file.

(2) parses the code to make sure it makes sense

processing

oF



Framework

Both oF and Processing are made up of base/existing classes

Compiling

In Java, each time you compile, your entire program is run through and changed into byte code.

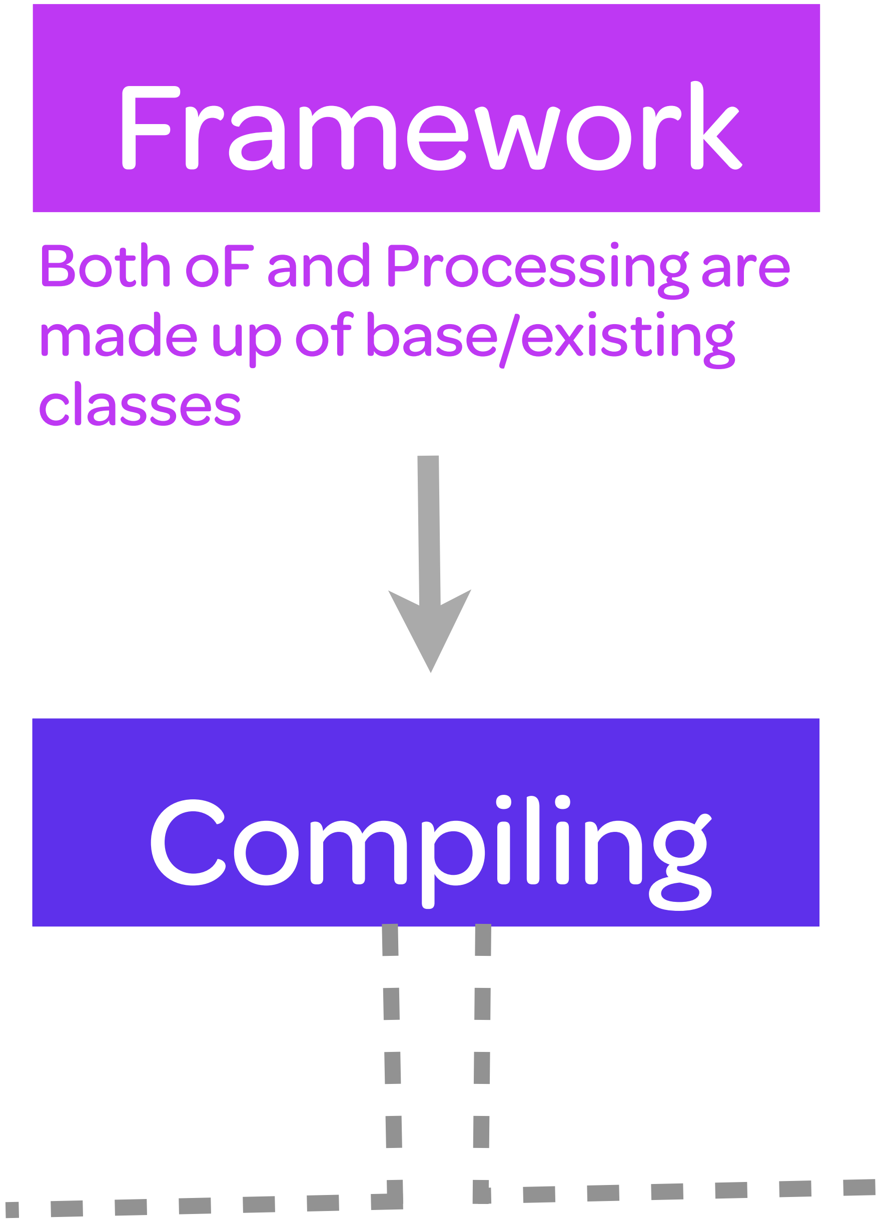
Then when you run your program, a Java interpreter does runtime compilation.

The compiler:
(1) goes through all your #include statements and copy/pastes chunks of your code to create one file.

(2) parses the code to make sure it makes sense

(3) translates the code into Assembly, a low-level language, and creates file objects from that

processing



Framework

oF

Both oF and Processing are made up of base/existing classes

Compiling

In Java, each time you compile, your entire program is run through and changed into byte code.

Then when you run your program, a Java interpreter does runtime compilation.

The compiler:
(1) goes through all your #include statements and copy/pastes chunks of your code to create one file.

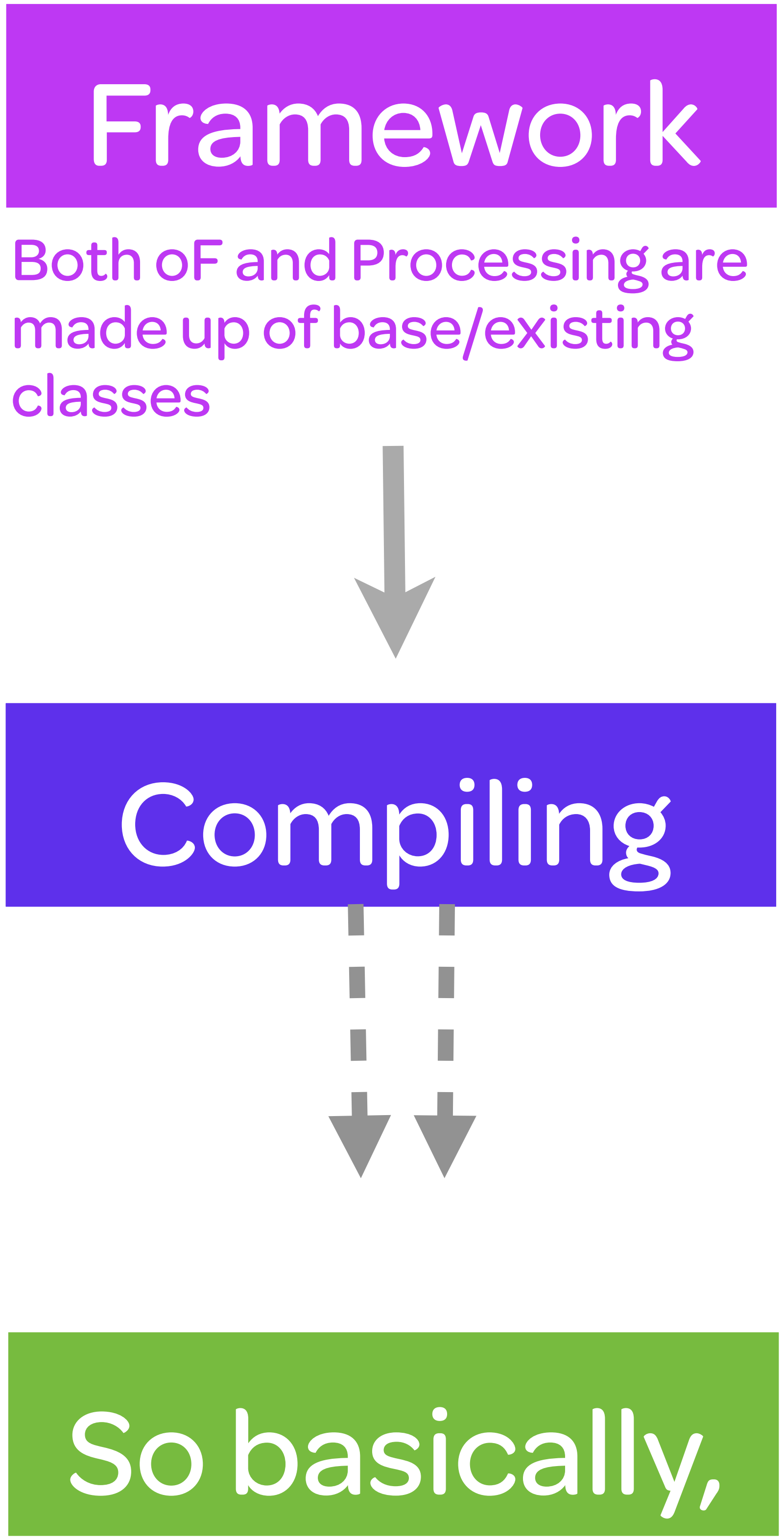
(2) parses the code to make sure it makes sense

(3) translates the code into Assembly, a low-level language, and creates file objects from that

(4) links Assembly objects together into a .app file

processing

oF



Framework

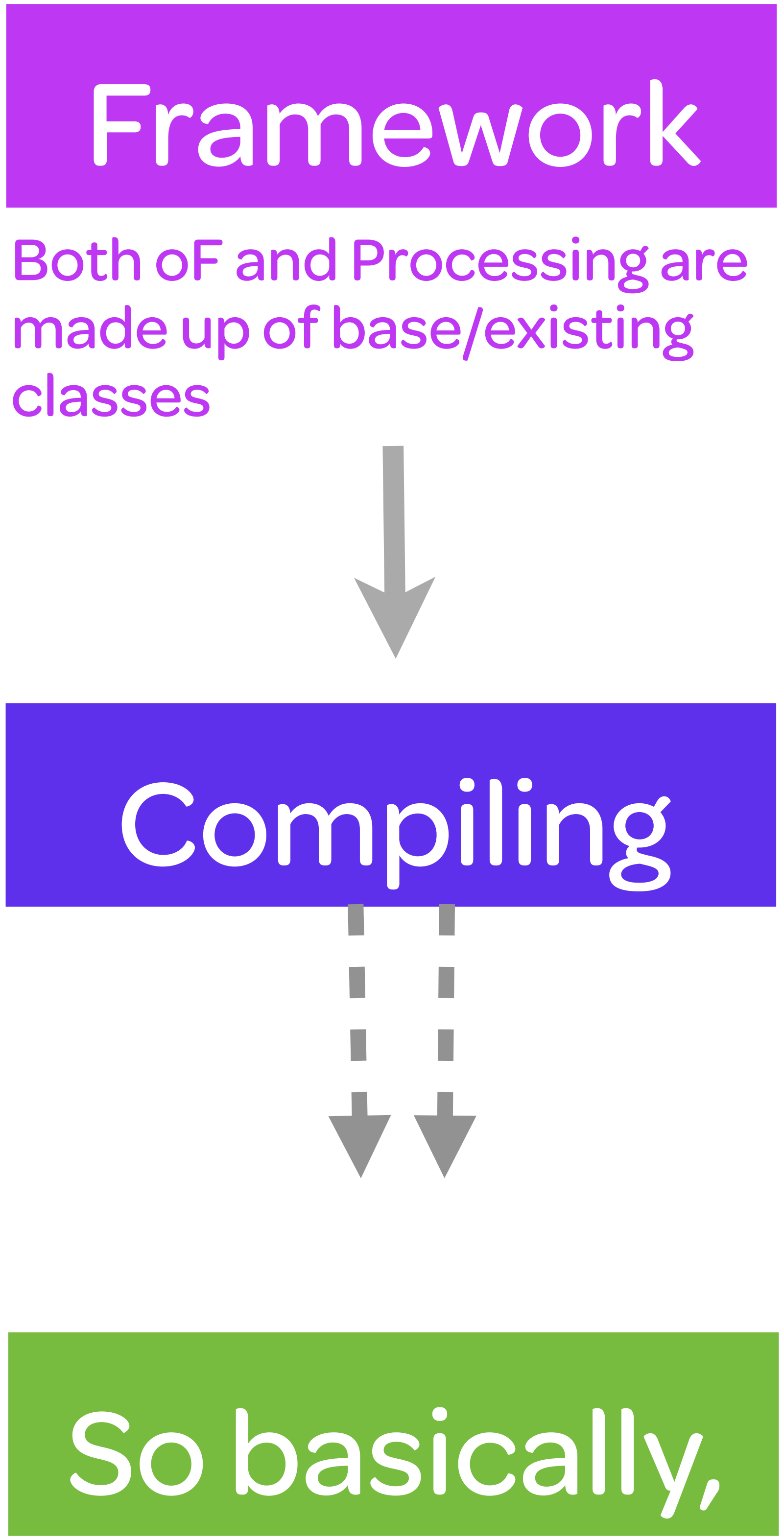
Both oF and Processing are made up of base/existing classes

Compiling

So basically,

processing

oF



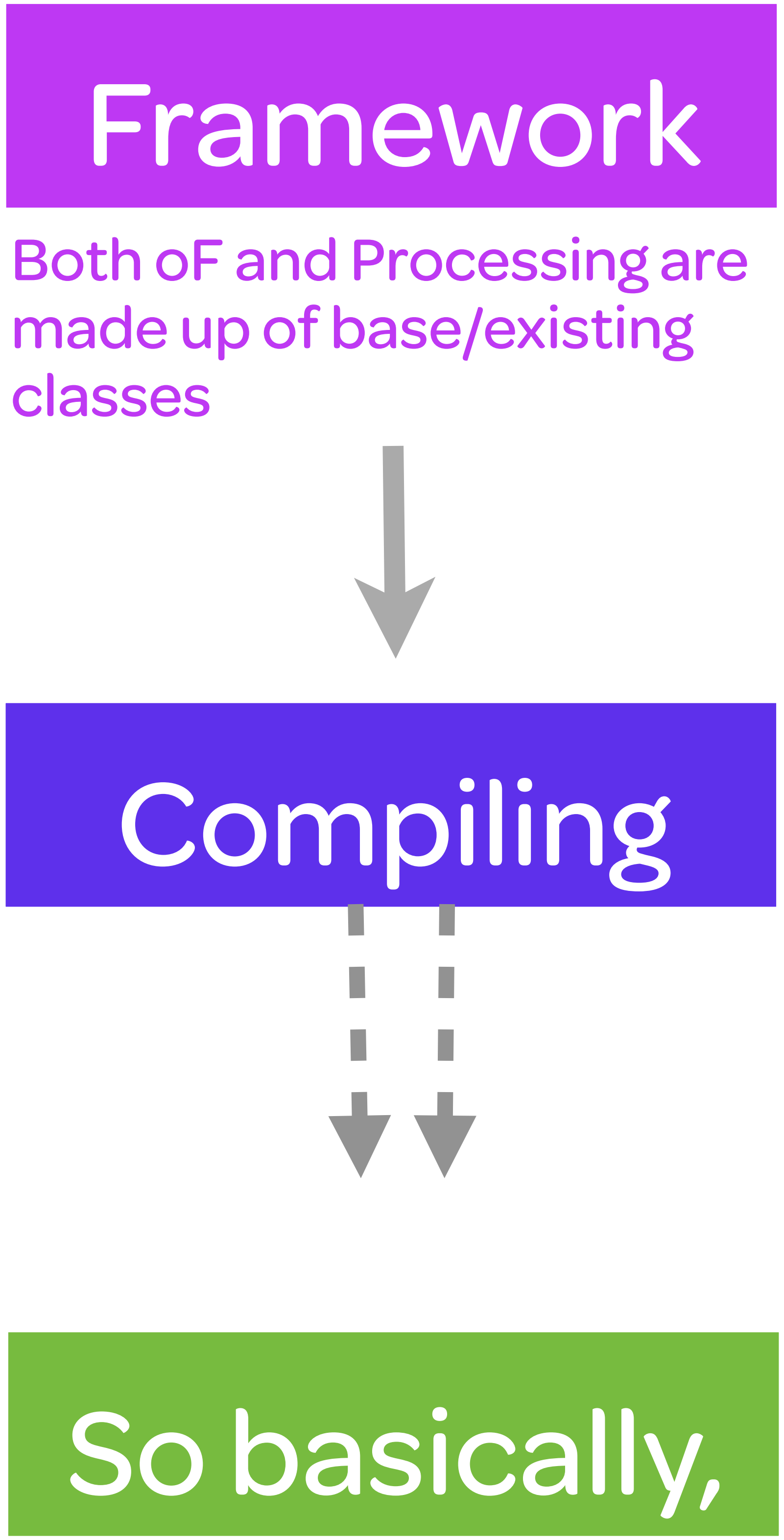
Framework

Both oF and Processing are made up of base/existing classes

Compiling

So basically,

In Java, the compiler rebuilds everything from scratch each time it runs.



processing

Framework

oF

Both oF and Processing are made up of base/existing classes

Compiling

So basically,

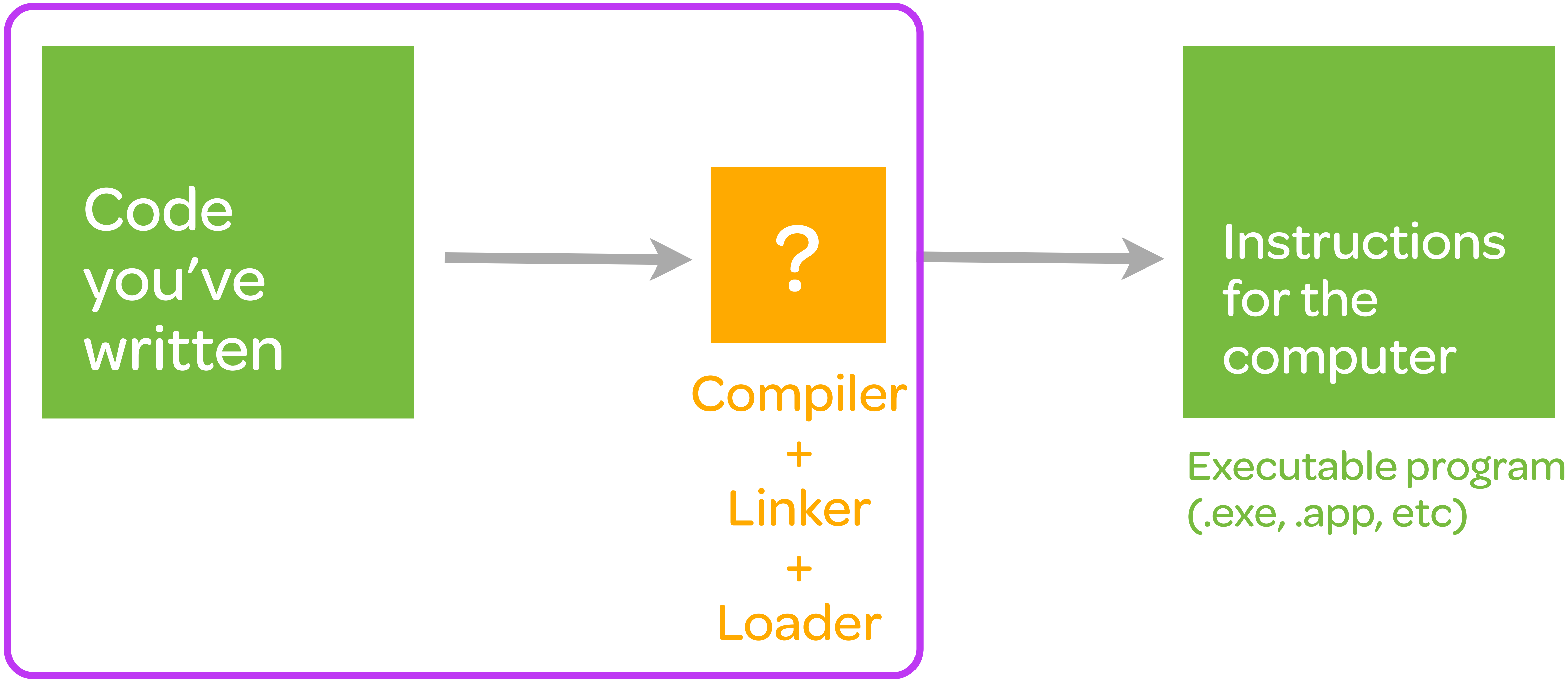
In Java, the compiler rebuilds everything from scratch each time it runs.

In oF, the compiler only needs to read /link things that have changed from build to build

how it works

SO WHAT DOES THIS MEAN FOR US?

Processing and Arduino have their own IDE
(Interactive Development Environment)



Code you've written

?

Compiler
+
Linker
+
Loader

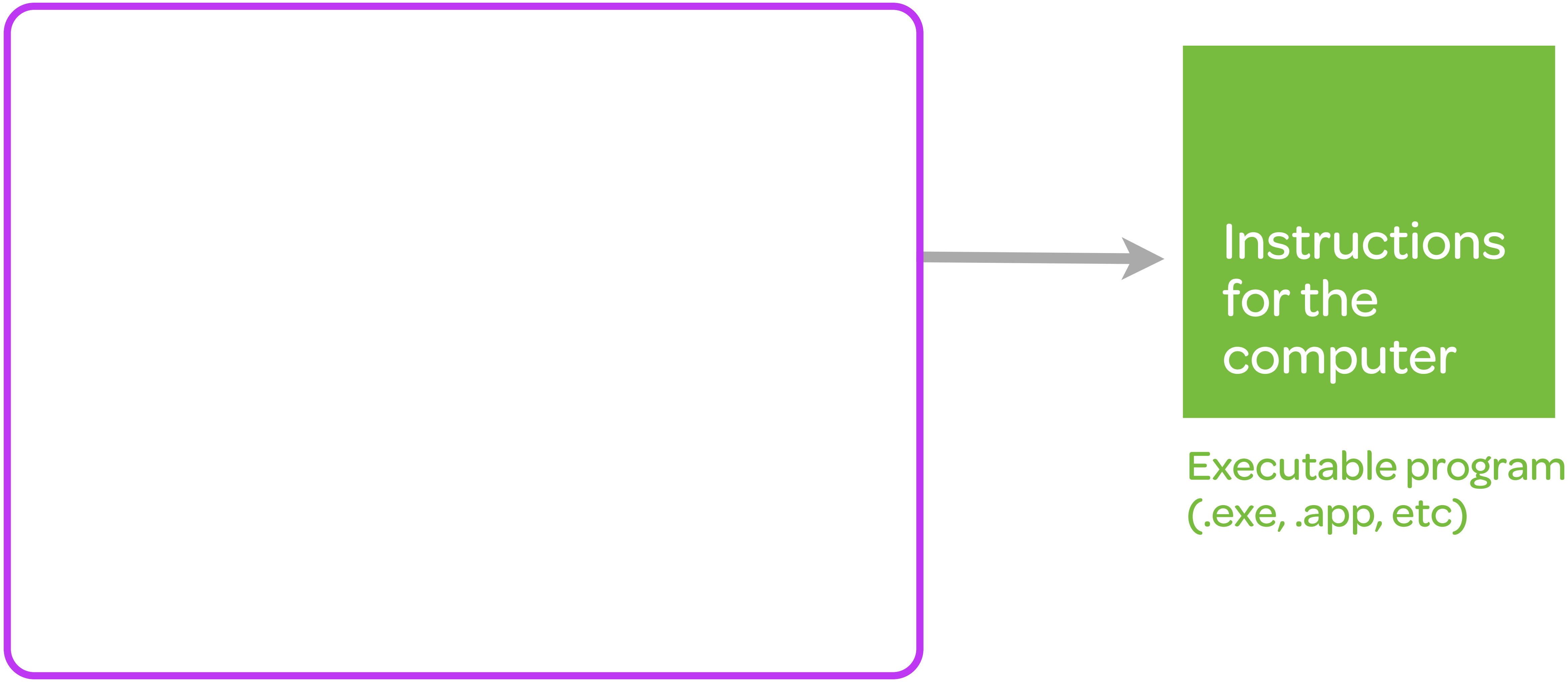
Instructions for the computer

Executable program
(.exe, .app, etc)

how it works

SO WHAT DOES THIS MEAN FOR US?

openFrameworks…not so much



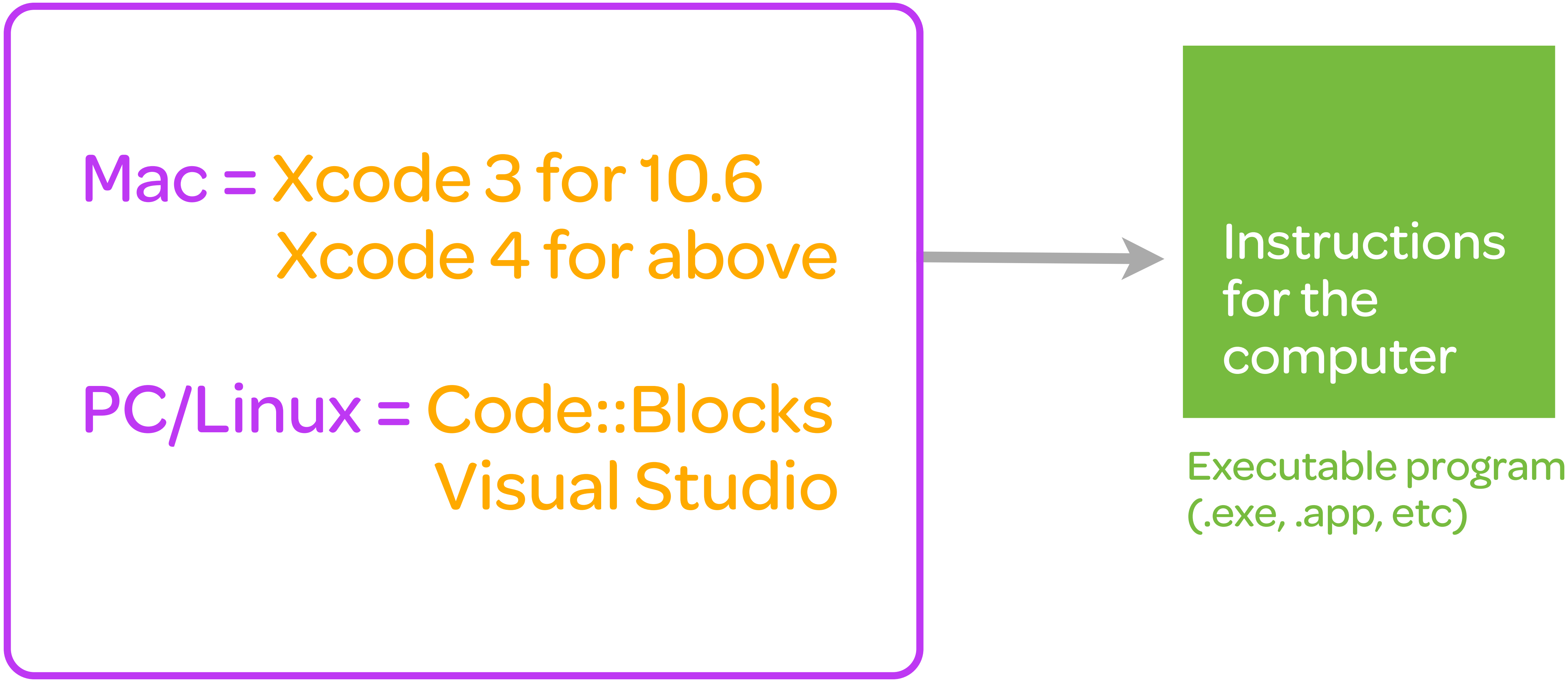
Instructions for the computer

Executable program (.exe, .app, etc)

how it works

SO WHAT DOES THIS MEAN FOR US?

openFrameworks...not so much



Mac = Xcode 3 for 10.6
Xcode 4 for above

PC/Linux = Code::Blocks
Visual Studio

Instructions for the computer

Executable program
(.exe, .app, etc)

install

ALL TOGETHER NOW

This can be a sticky process, so we are gonna do it together.

Install Xcode or Code::Blocks first
//Link
//Link

Download openFrameworks
//Link to download page

the need to knows

STRUCTURE OF OPENFRAMEWORKS

There are three main things you need to know to learn oF:

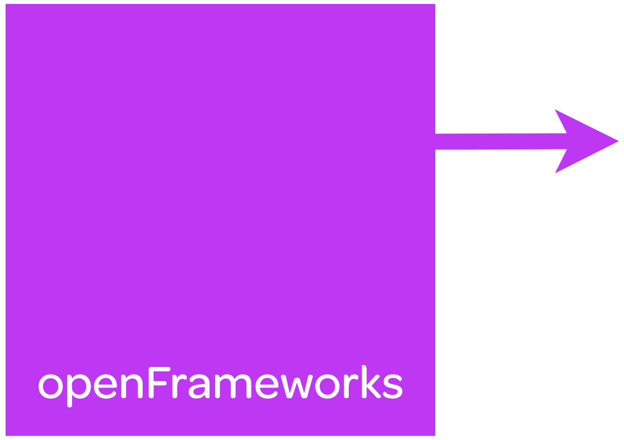
1) How to use an IDE
//RE: file structure

2) How to write C++ code
//RE: how to adapt others' code and reference the interwebs

3) How to use oF libraries
//Been there, done that. Twice. But more on this next class.

folders

STRUCTURE OF OPENFRAMEWORKS

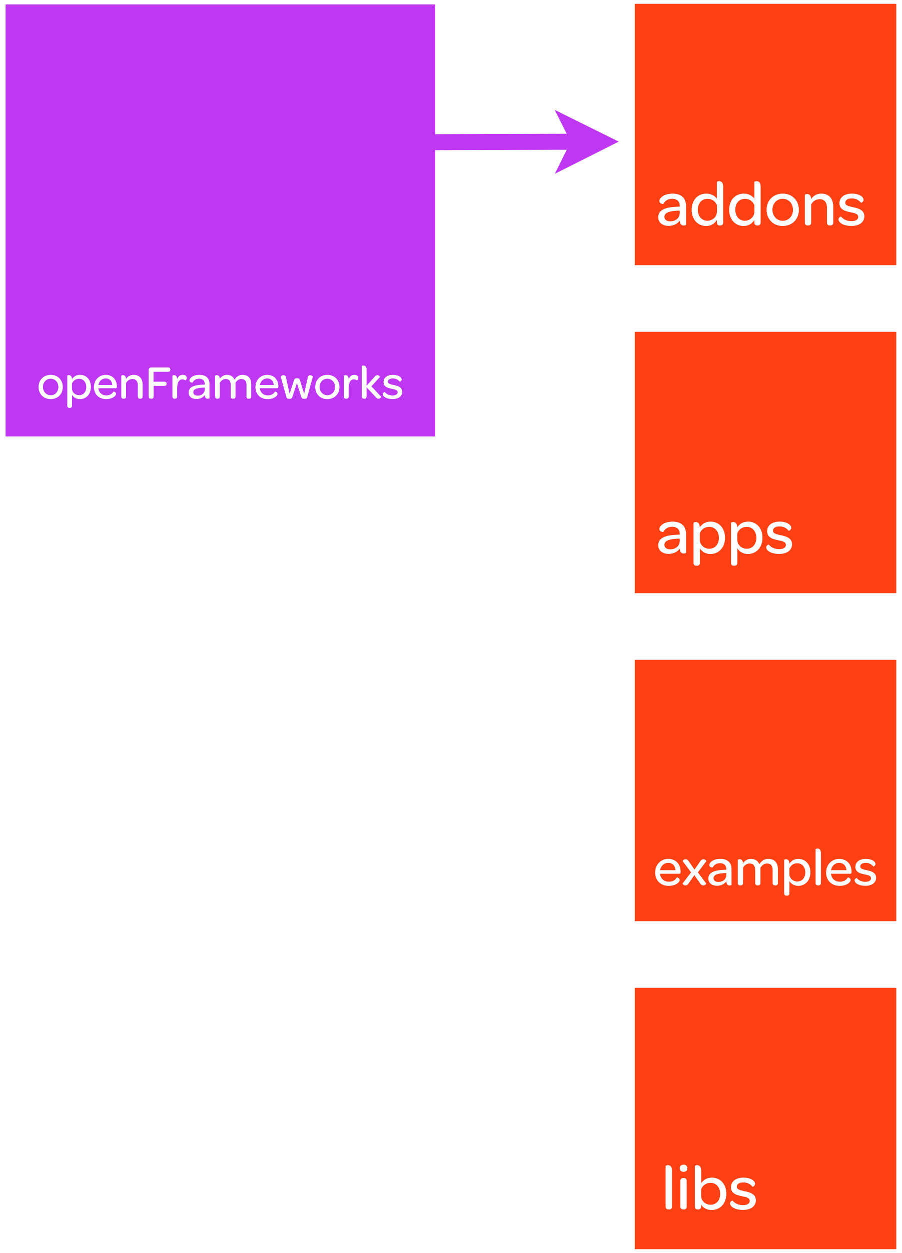


openFrameworks

folders

STRUCTURE OF OPENFRAMEWORKS

ROOT FOLDER



openFrameworks

addons

Adding piecemeal functionality and fun stuff to an app.

apps

Where you store your apps. More on this in a sec.

examples

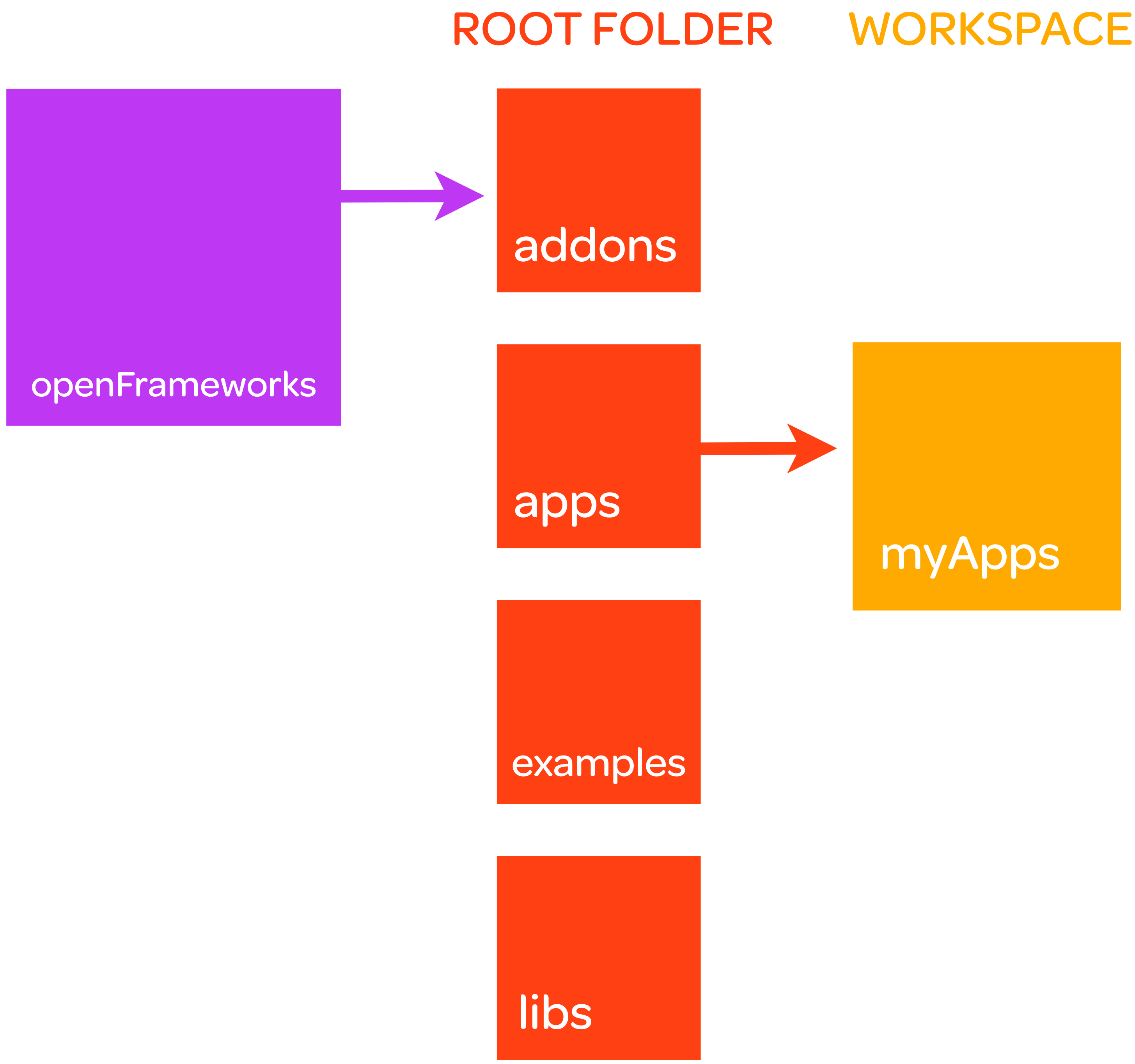
Self explanatory.

libs

Where your libraries live, as well as the oF core.

folders

STRUCTURE OF OPENFRAMEWORKS



ROOT FOLDER

WORKSPACES

openFrameworks

addons

apps

examples

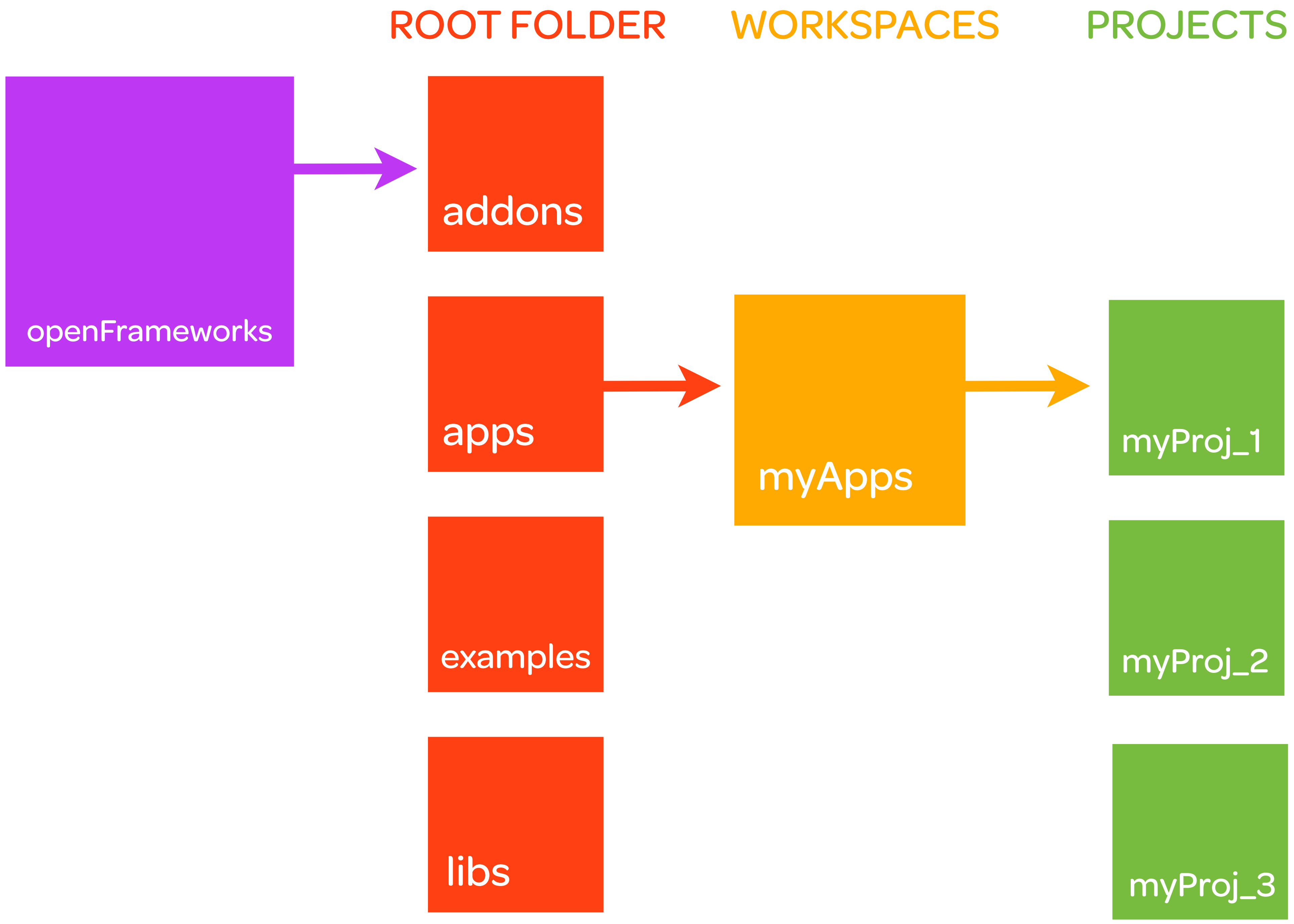
libs

myApps

Contains your projects

folders

STRUCTURE OF OPENFRAMEWORKS

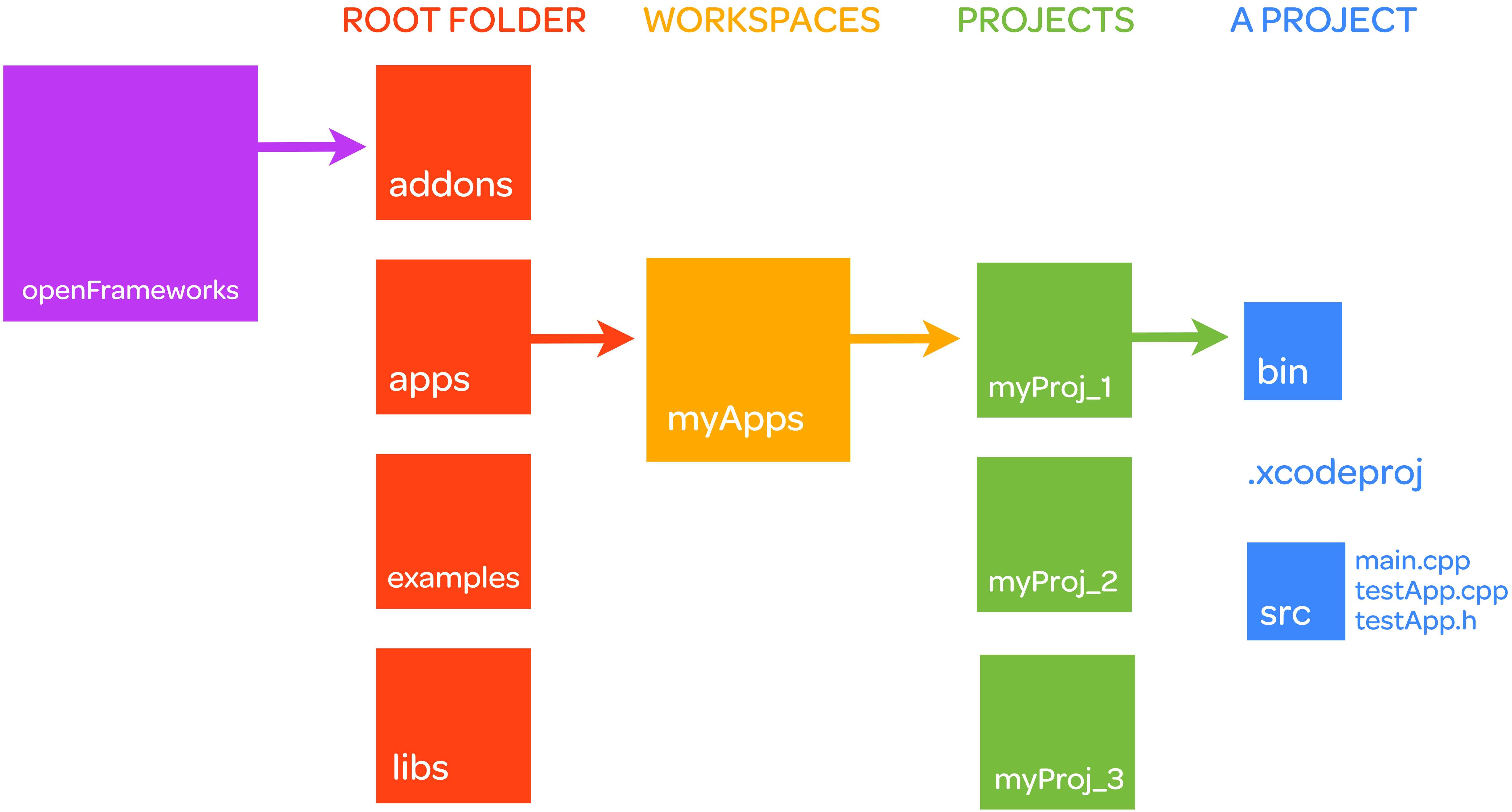


Contains folders for individual projects.

MUST be inside a workspace in order to compile!!

folders

STRUCTURE OF OPENFRAMEWORKS



ROOT FOLDER

WORKSPACES

PROJECTS

A PROJECT

openFrameworks

addons

apps

examples

libs

myApps

myProj_1

myProj_2

myProj_3

bin

.xcodeproj

src

main.cpp
testApp.cpp
testApp.h

example time

STRUCTURE OF OPENFRAMEWORKS

Open the graphicsExample.

Click Build and Run.

Ta daaaaa!!

files

STRUCTURE OF OPENFRAMEWORKS

In oF you have 3 files instead of one, as in Processing or Arduino.

main.cpp

testApp.h

testApp.cpp

Let’s use the old recipe analogy to understand why.

pumpkin pie

FILE STRUCTURE

helloWorld.pde

Ingredients
1 (8-ounce) package cream cheese, softened
2 cups canned pumpkin, mashed
1 cup sugar
1/4 teaspoon salt
1 egg plus 2 egg yolks, slightly beaten
1 cup half-and-half
1/4 cup (1/2 stick) melted butter
1 teaspoon vanilla extract
1/2 teaspoon ground cinnamon
1/4 teaspoon ground ginger, optional
1 piece pre-made pie dough
Whipped cream, for topping

Directions
Preheat the oven to 350 degrees F.
Place 1 piece of pre-made pie dough down into a (9-inch) pie pan and press down along the bottom and all sides. Pinch and crimp the edges together to make a pretty pattern. Put the pie shell back into the freezer for 1 hour to firm up. Fit a piece of aluminum foil to cover the inside of the shell completely. Fill the shell up to the edges with pie weights or dried beans (about 2 pounds) and place it in the oven. Bake for 10 minutes, remove the foil and pie weights and bake for another 10 minutes or until the crust is dried out and beginning to color.
For the filling, in a large mixing bowl, beat the cream cheese with a hand mixer. Add the pumpkin and beat until combined. Add the sugar and salt, and beat until combined. Add the eggs mixed with the yolks, half-and-half, and melted butter, and beat until combined. Finally, add the vanilla, cinnamon, and ginger, if using, and beat until incorporated.
Pour the filling into the warm prepared pie crust and bake for 50 minutes, or until the center is set. Place the pie on a wire rack and cool to room temperature. Cut into slices and top each piece with a generous amount of whipped cream.

pumpkin pie

FILE STRUCTURE

Ingredients

1 (8-ounce) package cream cheese, softened
2 cups canned pumpkin, mashed
1 cup sugar
1/4 teaspoon salt
1 egg plus 2 egg yolks, slightly beaten
1 cup half-and-half
1/4 cup (1/2 stick) melted butter
1 teaspoon vanilla extract
1/2 teaspoon ground cinnamon
1/4 teaspoon ground ginger, optional
1 piece pre-made pie dough
Whipped cream, for topping

testApp.h

Directions

Preheat the oven to 350 degrees F.

Place 1 piece of pre-made pie dough down into a (9-inch) pie pan and press down along the bottom and all sides. Pinch and crimp the edges together to make a pretty pattern. Put the pie shell back into the freezer for 1 hour to firm up. Fit a piece of aluminum foil to cover the inside of the shell completely. Fill the shell up to the edges with pie weights or dried beans (about 2 pounds) and place it in the oven. Bake for 10 minutes, remove the foil and pie weights and bake for another 10 minutes or until the crust is dried out and beginning to color.

For the filling, in a large mixing bowl, beat the cream cheese with a hand mixer. Add the pumpkin and beat until combined. Add the sugar and salt, and beat until combined. Add the eggs mixed with the yolks, half-and-half, and melted butter, and beat until combined. Finally, add the vanilla, cinnamon, and ginger, if using, and beat until incorporated.

Pour the filling into the warm prepared pie crust and bake for 50 minutes, or until the center is set. Place the pie on a wire rack and cool to room temperature. Cut into slices and top each piece with a generous amount of whipped cream.

testApp.cpp

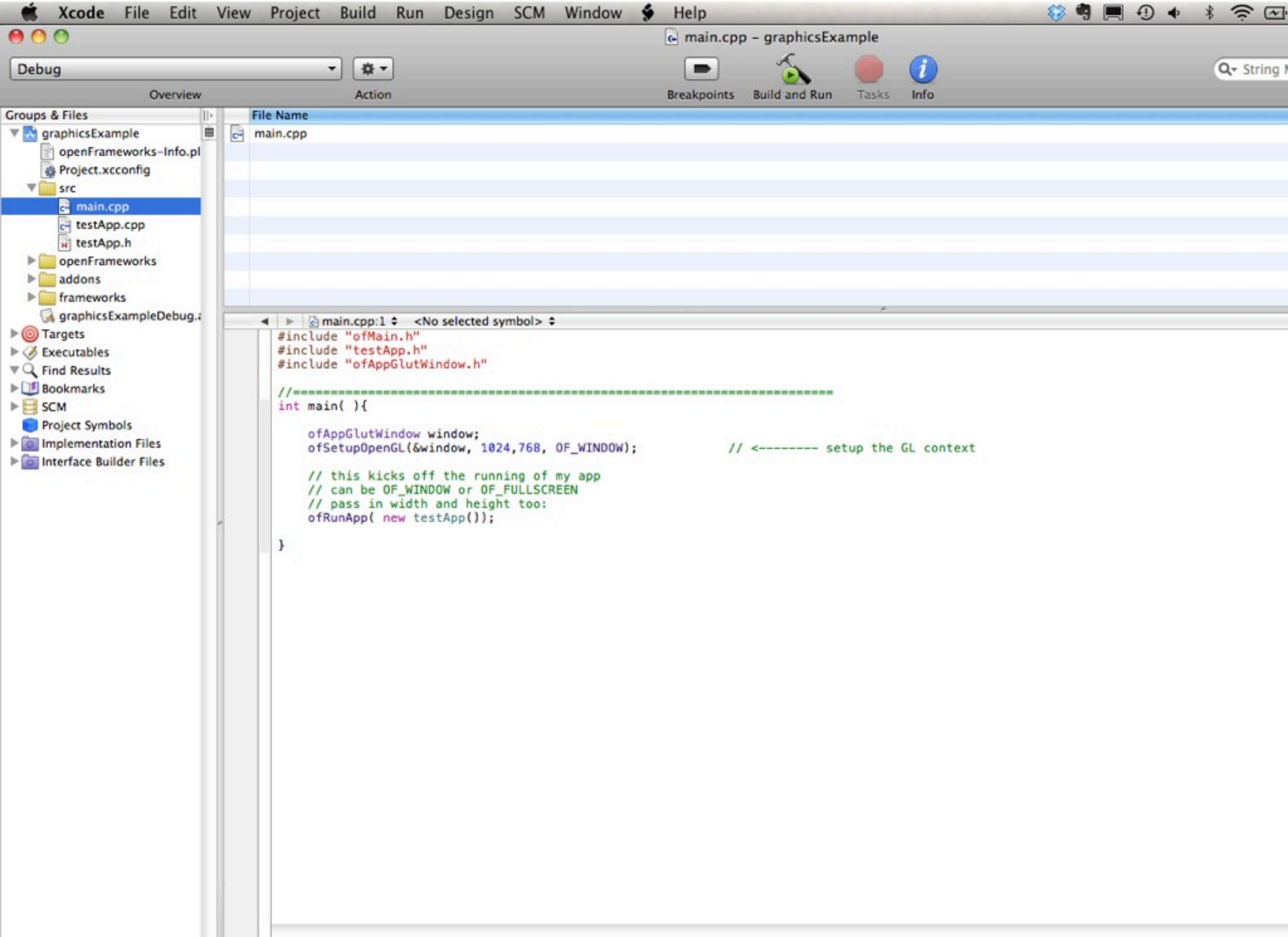
main.cpp

FILE STRUCTURE

main.cpp

Where the program starts

- Sets your screen size
- Starts off an infinite loop which runs your program



```
#include "ofMain.h"
#include "testApp.h"
#include "ofAppGlutWindow.h"

//========================================================================
int main( ){

    ofAppGlutWindow window;
    ofSetupOpenGL(&window, 1024,768, OF_WINDOW);            // <-------- setup the GL context

    // this kicks off the running of my app
    // can be OF_WINDOW or OF_FULLSCREEN
    // pass in width and height too:
    ofRunApp( new testApp());

}
```

testApp.h

FILE STRUCTURE

main.cpp

Where the program starts

- Sets your screen size
- Starts off an infinite loop which runs your program

testApp.h

This is your header file, or ingredients list.

- DECLARE all global variables and functions declared here
- Similar to declaring all global variables at the top of your processing sketch

testApp.h

FILE STRUCTURE

main.cpp

Where the program starts

- Sets your screen size
- Starts off an infinite loop which runs your program

testApp.h

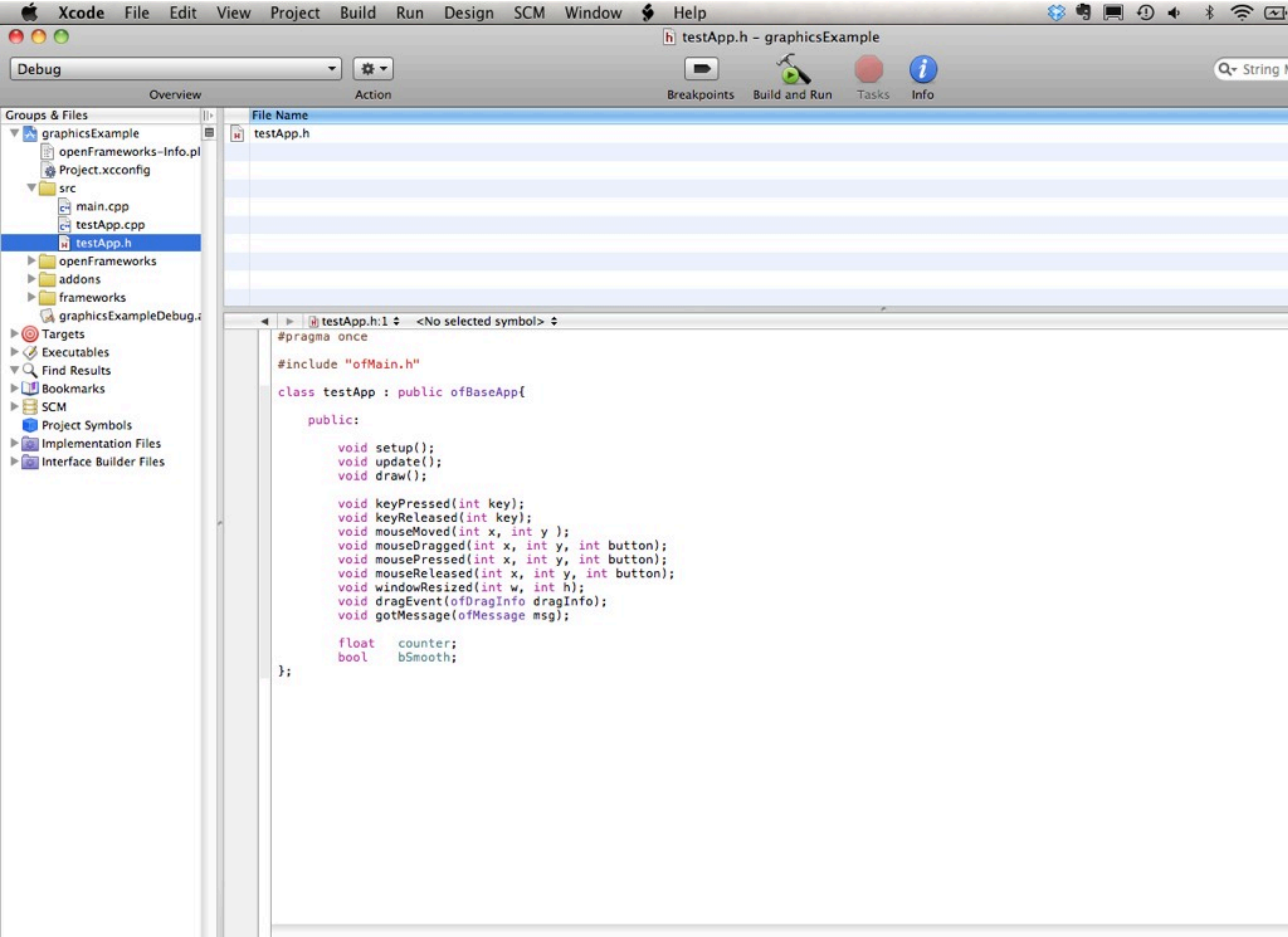
This is your header file, or ingredients list.

- DECLARE all global variables and functions declared here
- Similar to declaring all global variables at the top of your processing sketch

Other things that go in here:

//For later reference

- Preprocessor statements there to prevent multiple header definitions
- Include statements to other classes
- Class extension statements
- Variables local to the class
- Prototypes / declarations of any functions to be contained in the class
- Security settings of these functions and variables (e.g. public, private, protected, etc).



```
#pragma once

#include "ofMain.h"

class testApp : public ofBaseApp{

    public:

        void setup();
        void update();
        void draw();

        void keyPressed(int key);
        void keyReleased(int key);
        void mouseMoved(int x, int y );
        void mouseDragged(int x, int y, int button);
        void mousePressed(int x, int y, int button);
        void mouseReleased(int x, int y, int button);
        void windowResized(int w, int h);
        void dragEvent(ofDragInfo dragInfo);
        void gotMessage(ofMessage msg);

        float   counter;
        bool    bSmooth;
};
```

testApp.cpp

FILE STRUCTURE

main.cpp

Where the program starts

- Sets your screen size
- Starts off an infinite loop which runs your program

testApp.h

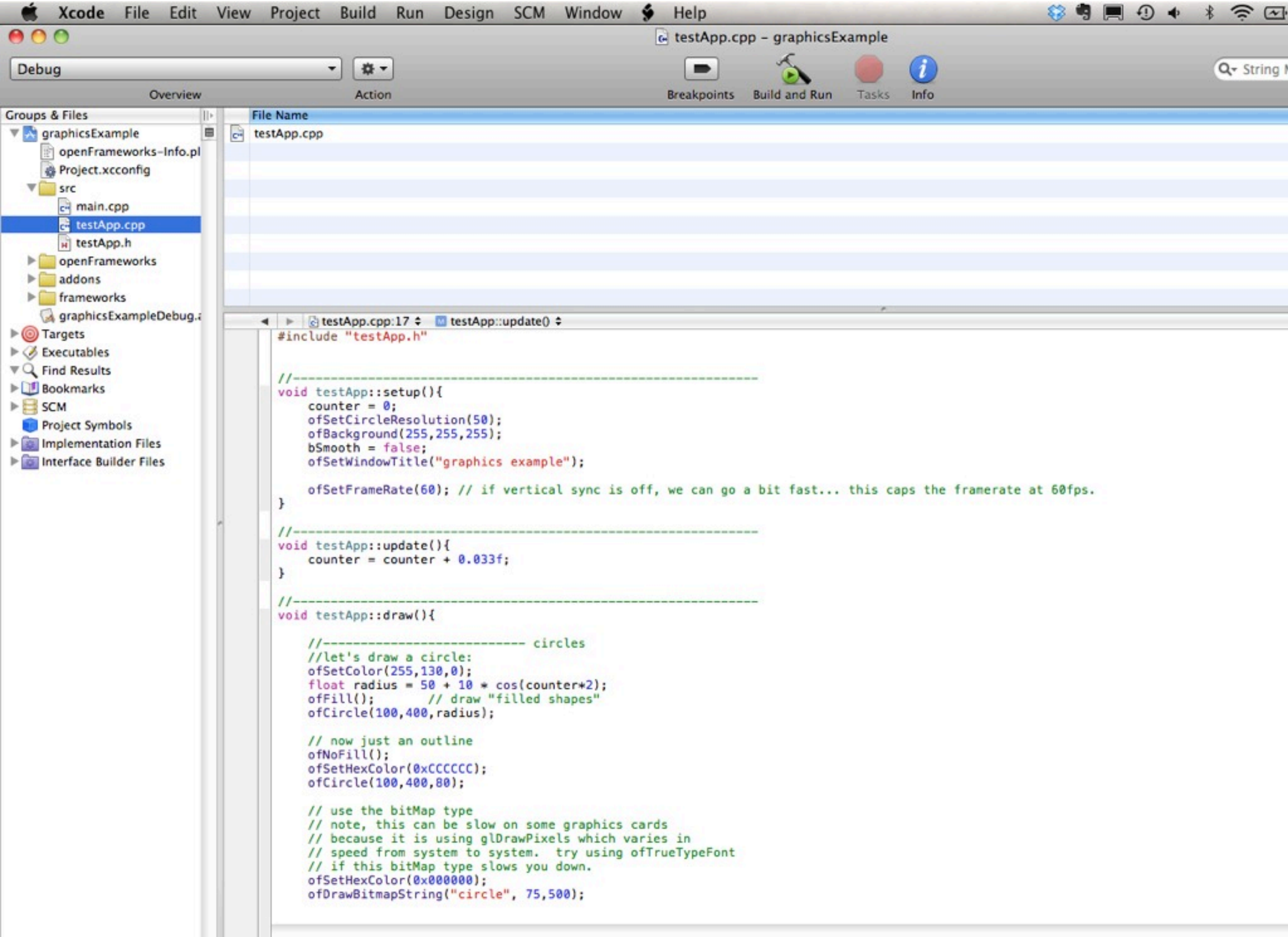
This is your header file, or ingredients list.

- DECLARE all global variables and functions declared here
- Similar to declaring all global variables at the top of your processing sketch

testApp.cpp

Where your functions live (the directions)

- Includes all the functions we are familiar with: setup (), draw(), etc
- An include statement that references the .h file
- All of the code to fill in the function prototypes.



```
#include "testApp.h"


//--------------------------------------------------------------
void testApp::setup(){
    counter = 0;
    ofSetCircleResolution(50);
    ofBackground(255,255,255);
    bSmooth = false;
    ofSetWindowTitle("graphics example");

    ofSetFrameRate(60); // if vertical sync is off, we can go a bit fast... this caps the framerate at 60fps.
}

//--------------------------------------------------------------
void testApp::update(){
    counter = counter + 0.033f;
}

//--------------------------------------------------------------
void testApp::draw(){

    //--------------------------- circles
    //let's draw a circle:
    ofSetColor(255,130,0);
    float radius = 50 + 10 * cos(counter*2);
    ofFill();       // draw "filled shapes"
    ofCircle(100,400,radius);

    // now just an outline
    ofNoFill();
    ofSetHexColor(0xCCCCCC);
    ofCircle(100,400,80);

    // use the bitMap type
    // note, this can be slow on some graphics cards
    // because it is using glDrawPixels which varies in
    // speed from system to system.  try using ofTrueTypeFont
    // if this bitMap type slows you down.
    ofSetHexColor(0x000000);
    ofDrawBitmapString("circle", 75,500);
```

create an app

ACTIONIZE

Let’s walk through the process of creating an app.