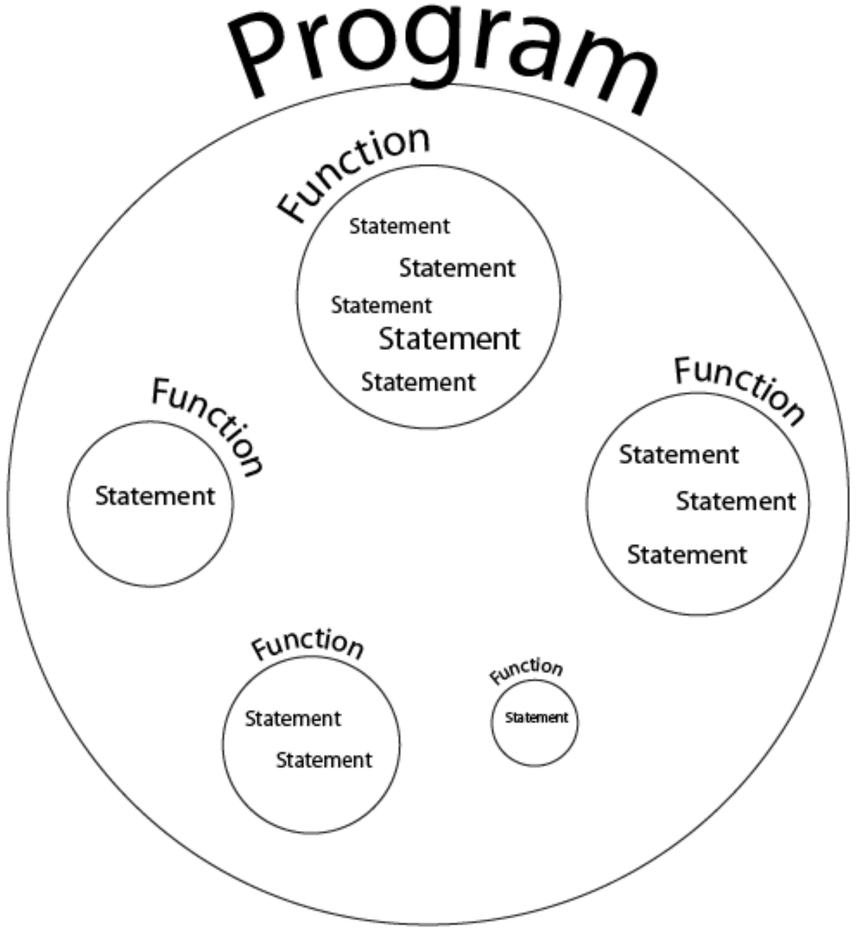
An Introduction to Object-Oriented Programming

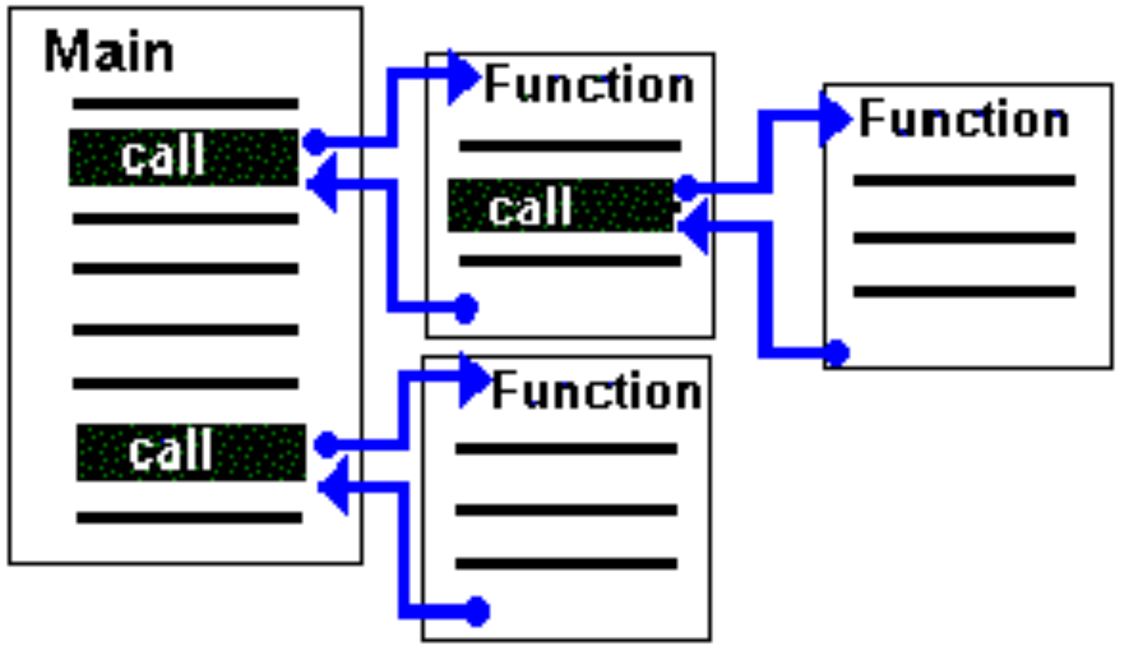
CC Lab 2016 openFrameworks Week 3



Procedural Programming

A program is a bunch of procedures.

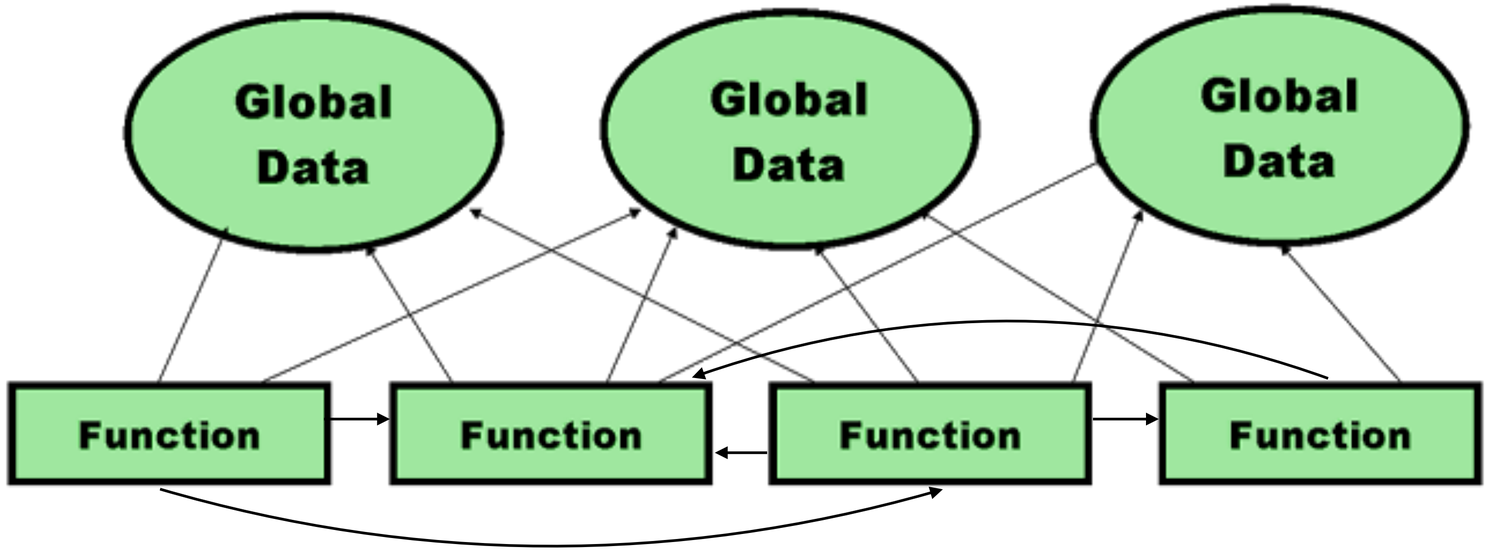
Image source: http://openframeworks.cc/ofBook/chapters/cplusplus_basics.html



Procedural Programming

A procedure can be composed of sub-procedures.

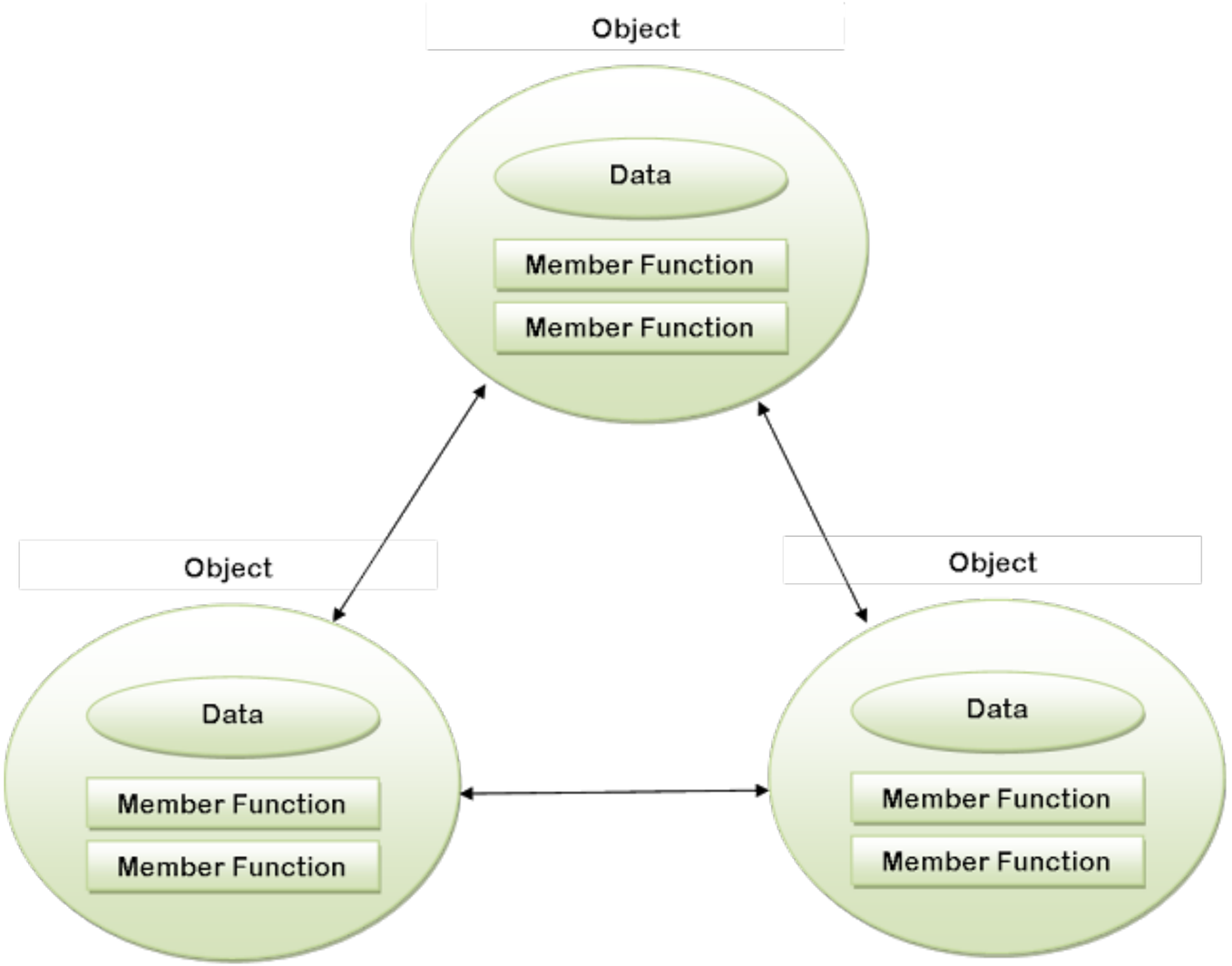
Image source: http://coronet.iicm.edu/sa/scripts/lesson01.htm



Procedural Programming

A dangerous flattened world

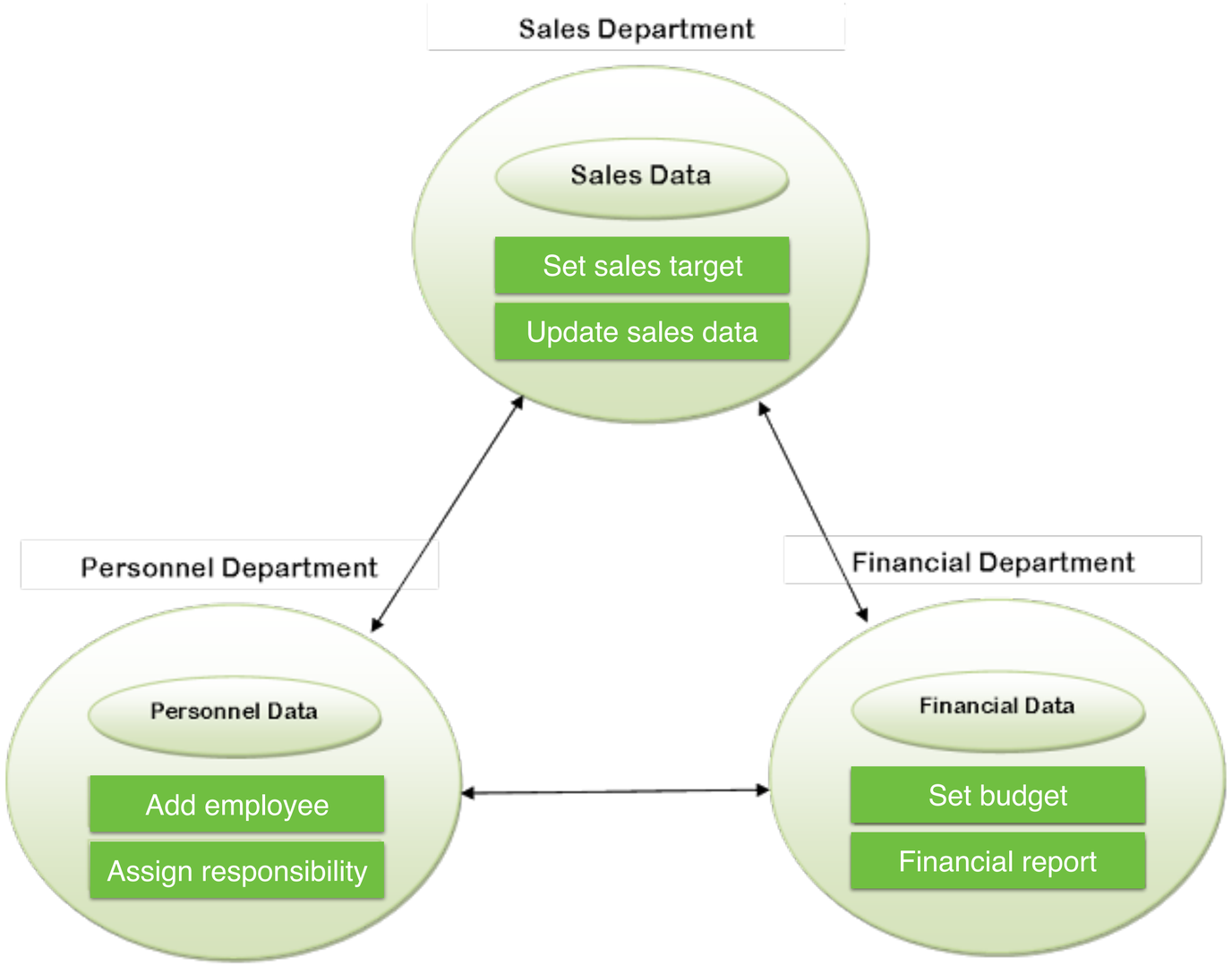
Image source: http://embeddedcareers-world.blogspot.com/2013/04/object-oriented-programming-with-c-part.html



Object-Oriented Programming (OOP)

Modeling the real world with data encapsulation

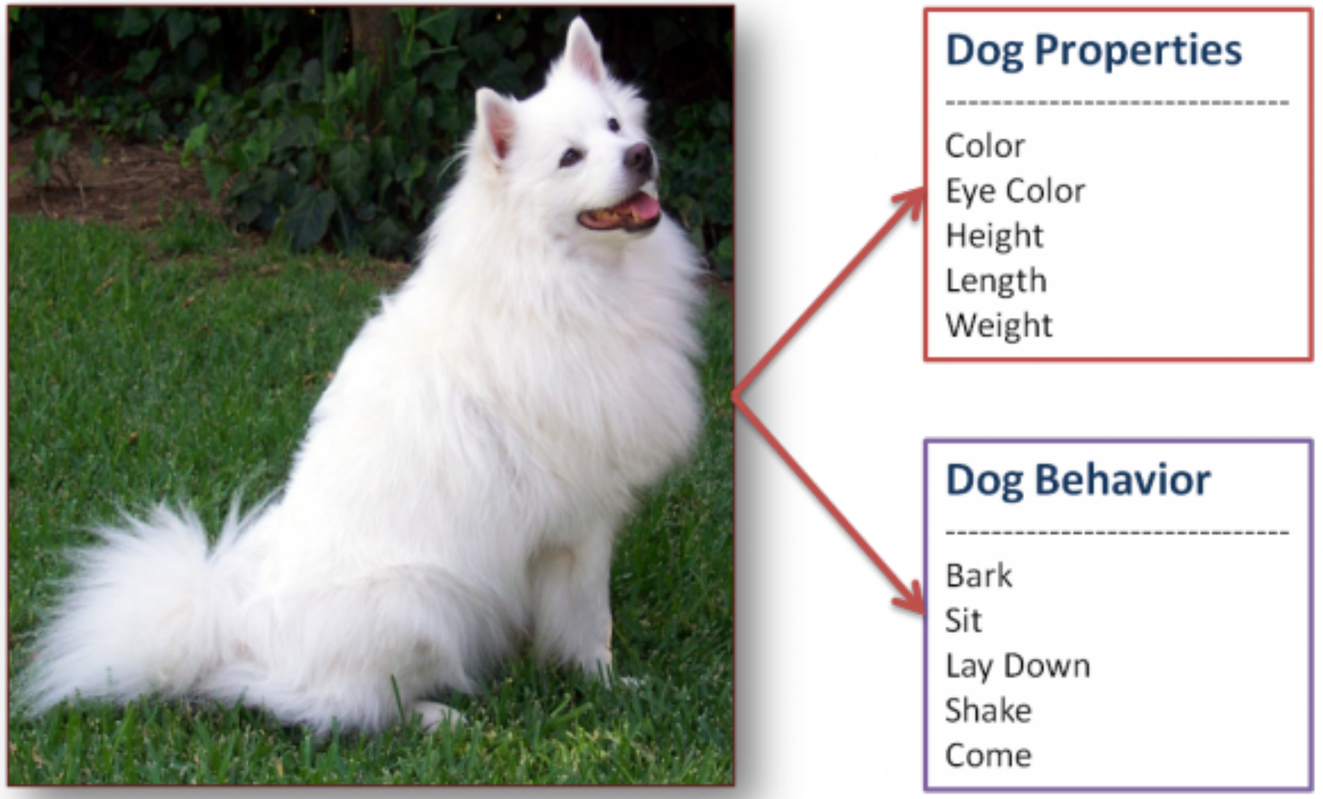
Image source: http://embeddedcareers-world.blogspot.com/2013/04/object-oriented-programming-with-c-part.html



Object-Oriented Programming (OOP)

The corporate model

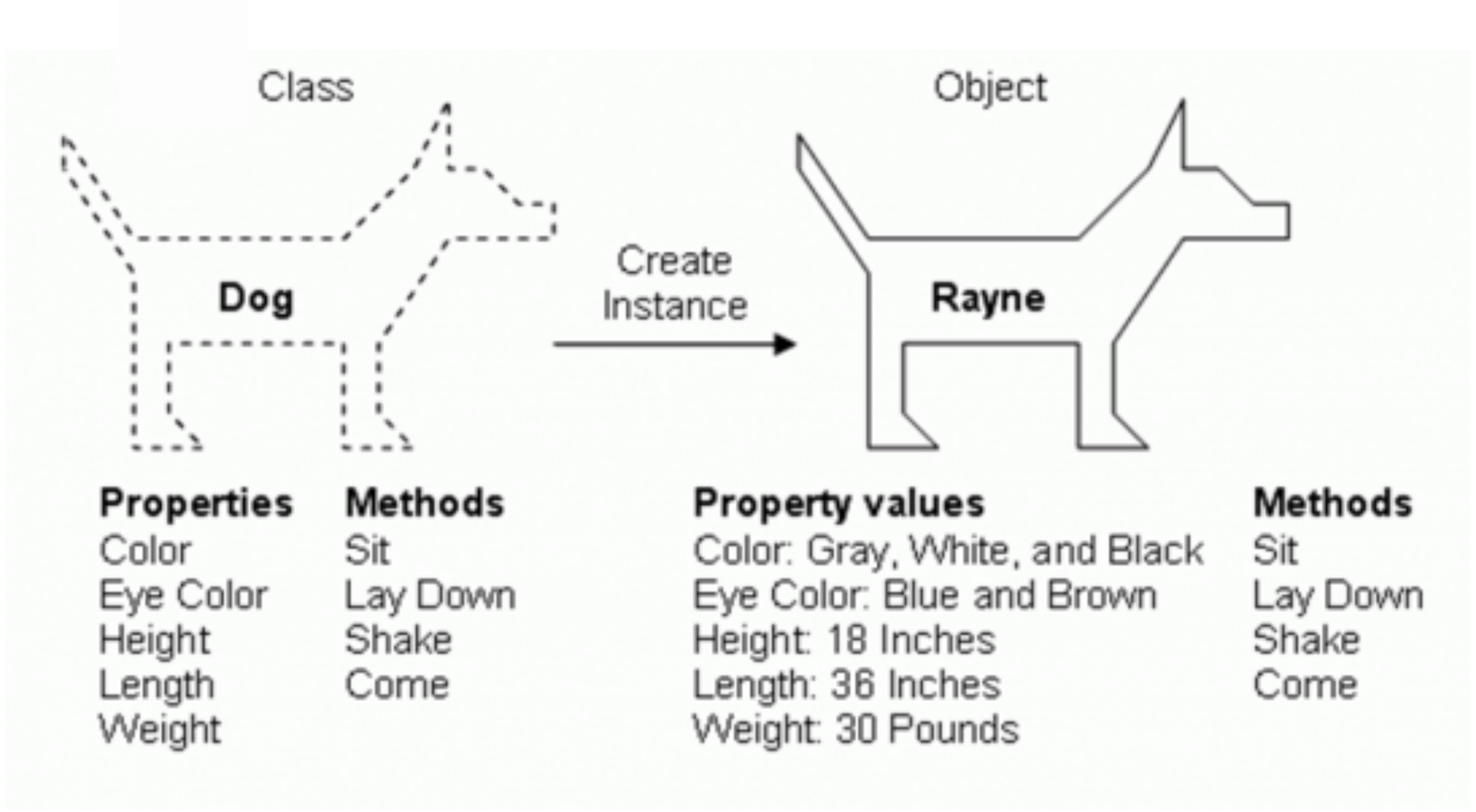
Image source: http://embeddedcareers-world.blogspot.com/2013/04/object-oriented-programming-with-c-part.html



Object-Oriented Design

Analyze the problem

Image source: http://lexisbirds.blogspot.com/p/c.html



Class

Mould of objects

Image source: http://lexisbirds.blogspot.com/p/c.html

A function that returns a value or an object:

```
int calculateArea(float width, float height){
    float = width * height;
    return area;
}
```

A function that does not return anything:

```
void setup()
{
  // do something
}
```

What does a function look like in C++?

```
void ofApp::setup(){
  // do something
}
void ofApp::update(){
  // do something
}
void ofApp::draw(){
  // do something
}
```

An oF app will at least have these functions.

Familiar, right?

```
class Dog {
// public functions
public:
    void setup(){
       //do something
    };
    void update(){
       //do something
    };
    void draw(){
       //do something
    };
    float mNumber;
};
```

Defining a class

The `class` keyword

```
class Dog {
// public functions
public:
    Dog(); // constructor
    void setup();
    void update();
    void draw();
    float mNumber;
};
```

Separate interface and implementation into .h and .cpp files.

Dog.h

```
#include "example.h"
Dog::Dog(){
   //initialization
}
void Dog::setup(){
   //do something
};
void Dog::update(){
   //do something
};
void Dog::draw(){
   //do something
};
```

Separate interface and implementation into .h and .cpp files.

Dog.cpp

Use a constructor to assign initial values to your objects.

```
Dog::Dog() {
   cout << "this class is being created" << endl;
   mNumber = 0;
}
```

You declare an object by calling the constructor.

```
Dog rayne();
```

A constructor can also accept arguments.

```
Dog::Dog(float important_number) {
   cout << "this class is being created" << endl;
   mNumber = important_number;
}
```

Then you use it by passing the argument.

Dog bella(42);

How to use a class in your main app?

#include
In order to use the class and its functions. We need to have the tell the app that we are using them. So the first thing we want to do is include the `example.h` file in your main app `.h` file - like the `ofApp.h`. Just include your class doing `#include "Dog.h"`

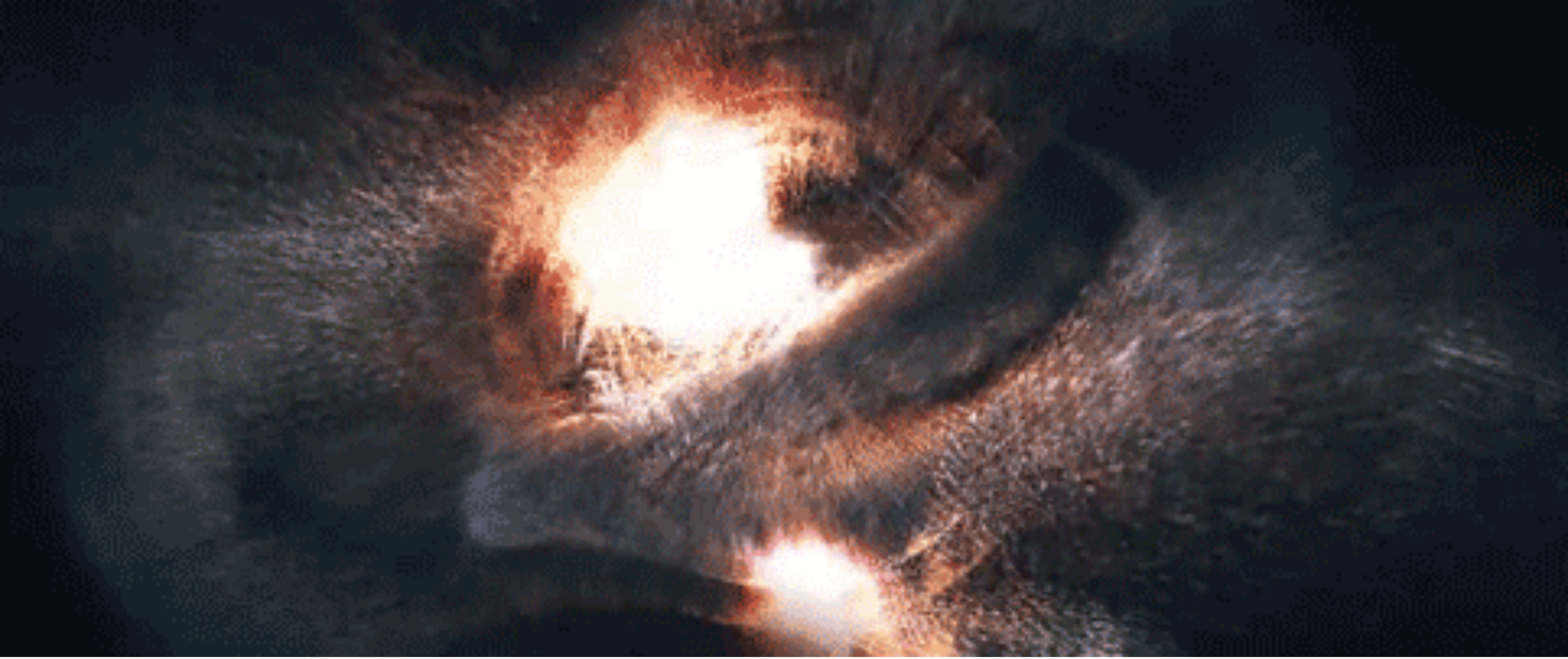
`example.mNumber` **or** `example.setup()`
Access a variable or a function is easy. Use `.` after class name to access the variable, so as the function. In order to distinguish between variable a function. We need `()` to be added after a function, it is prepared for taking arguments even though we are using none.

What is `#pragma once` ???

You probably noticed that `#pragma once` is literally being used in every single header files.

> In the C and C++ programming languages, #**pragma once** is a non-standard but widely supported preprocessor directive designed to cause the current source file to be included only **once** in a single compilation. - Wikipedia

If you don't have it in your program. The program will probably still work. However, if the header file is include somewhere like a precompiled header file, the header file might get sourced by both the precompiled header and the it self, causing an error.



Example: A particle system!

(Not this one, of course…)

Homework

- Improve your particle system. You can either
  - make it interactive (remember how to do mouse interactions? You can implement keyboard interactions, or even include physical buttons/ sensors using Firmata!), or
  - give the particles a richer look. Try to draw different colors and shapes. Try to make the look change over time. You can even let your app (literally) rain cats and dogs!
- Push your code to Github, publish a video demo to our slack channel.