Other Object-Oriented Languages

Charlotte Pierce

SWINBURNE UNIVERSITY OF TECHNOLOGY

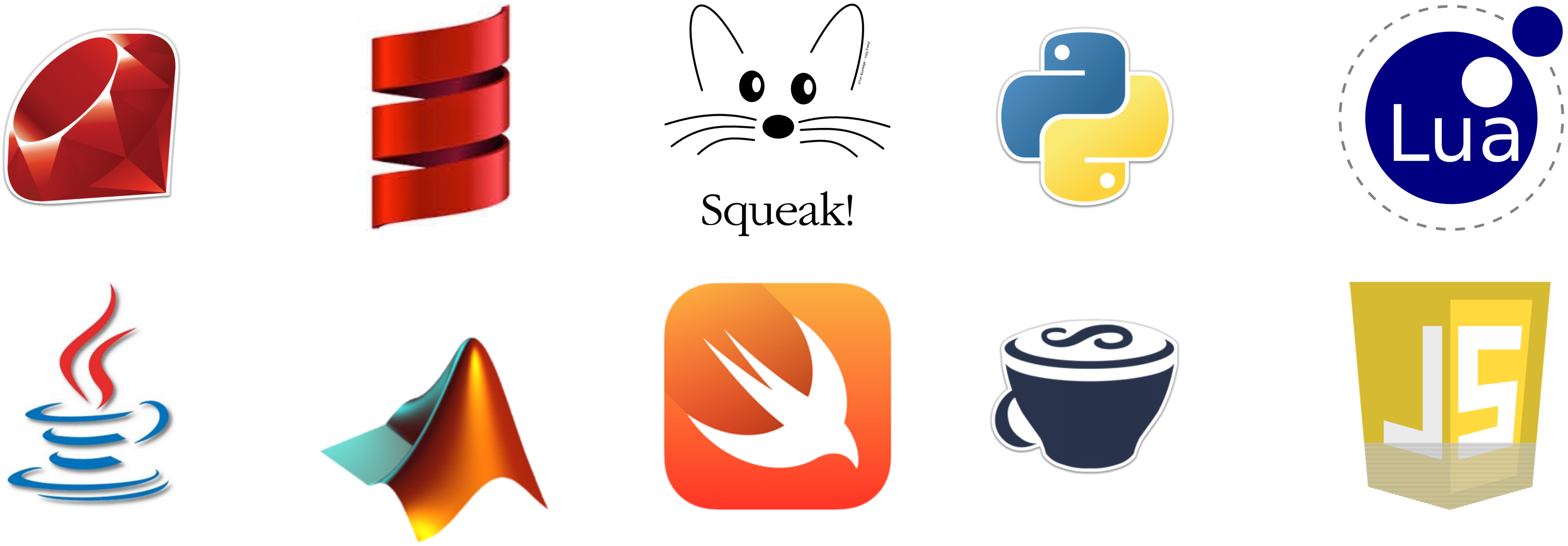
Industry uses a range of object oriented programming languages across all domains

To succeed you will need to understand how what you have learnt applies to other languages

Learning other languages
can appear daunting

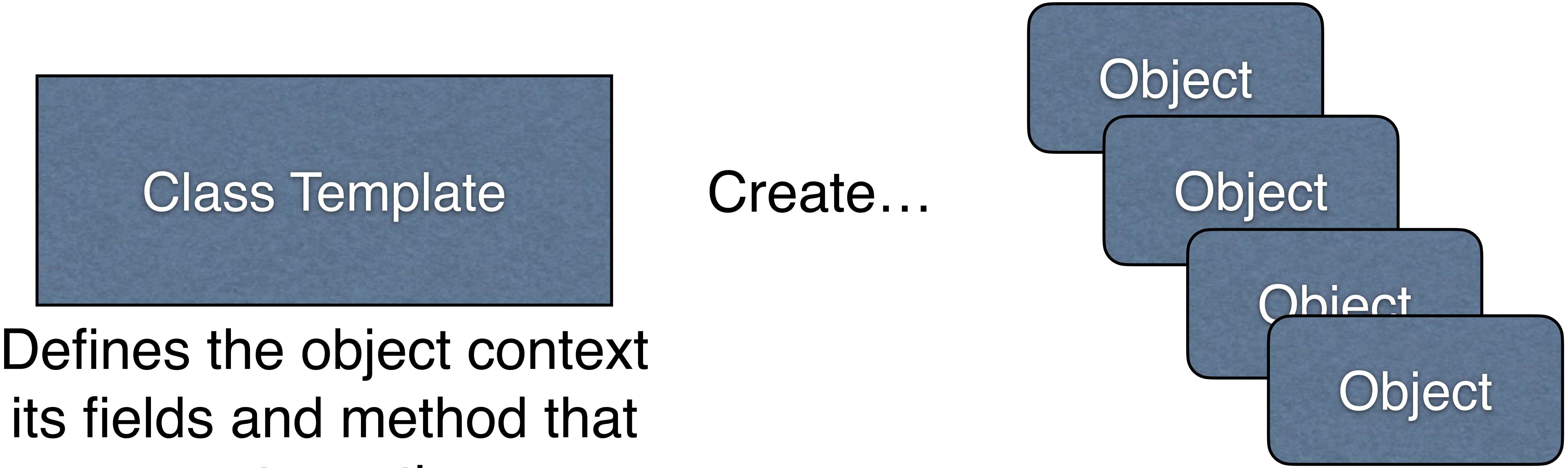
With the right start things should be relatively straight forward

Approach new languages armed with the principles of OOP



See that each language supports abstraction, encapsulation and inheritance

Create classes with inheritance to build objects in many languages



Class Template

Create…

Object

Object

Object

Object

Defines the object context its fields and method that operate on these

eg: C++ / C# / Java / Swift / ObjC

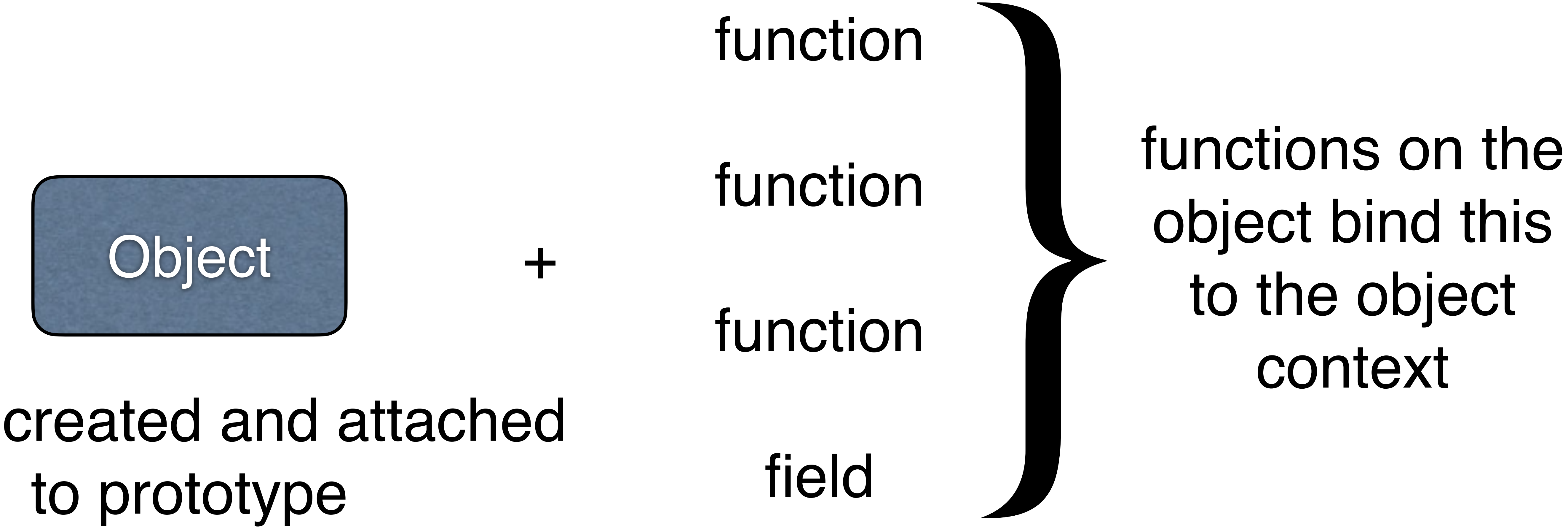
class declaration examples: C++

```
#include <iostream>

using namespace std;

class SomeClass
{
   private:

    // Data Members
    string _someString;

   public:

    SomeClass(string str)
    {
        _someString = str;
    }

    // Member Functions
    void printSomething()
    {
        cout << "This is a String: " << _someString << endl;
    }

};
```

class declaration examples: Swift

```
import Foundation

class SomeClass
{

        // class variables
        private var _someString : String

        // constructor
    init(str : String)
    {
        _someString = str
    }

        // Member functions
    func printSomething()
    {
        print(_someString)
    }

}

var sc = SomeClass(str: "hello world")
sc.printSomething()
```

Use prototypes and extend objects in Prototyping languages



Object

+

function

function

function

field

functions on the object bind this to the object context

Object created and attached to prototype

eg: JavaScript

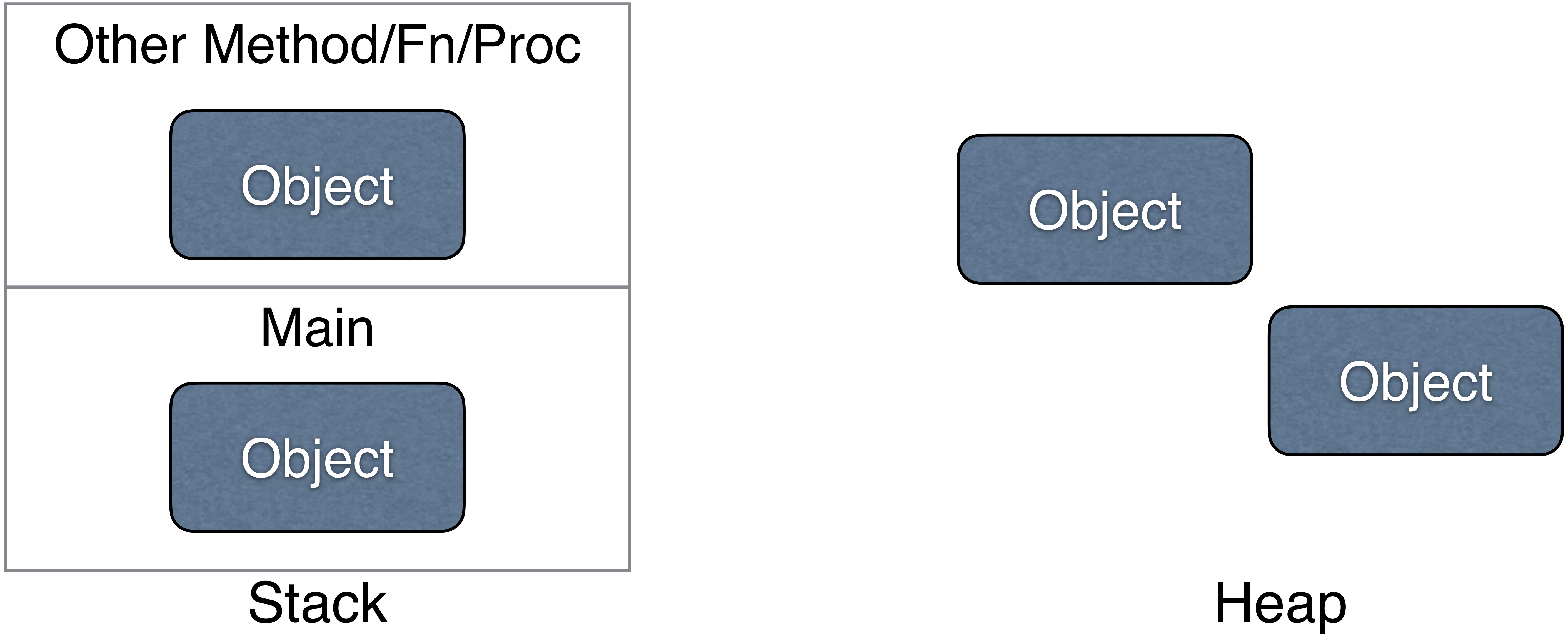
Example prototype: JavaScript

```
function Plant () {
    this.country = "Mexico";
    this.isOrganic = true;
}

// Add the showNameAndColor method to the Plant
prototype
// property
Plant.prototype.showNameAndColor =  function ()
{
    console.log("I am a " + this.name + " and
my color is " +
    this.color);
}

// Add the amIOrganic method to the Plant
prototype property
Plant.prototype.amIOrganic = function () {
    if (this.isOrganic)
        console.log("I am organic, Baby!");
}
```

```
function Fruit (fruitName, fruitColor) {
    this.name = fruitName;
    this.color = fruitColor;
}

// Set the Fruit's prototype to Plant's
constructor,
// thus inheriting all of Plant.prototype
methods and
// properties.
Fruit.prototype = new Plant ();

// Creates a new object, aBanana, with the
Fruit
// constructor

var aBanana = new Fruit ("Banana",
“Yellow");
```

Example code from http://javascriptissexy.com/javascript-prototype-in-plain-detailed-language/

See that some languages let you place objects on the stack and heap



eg: C++ / Delphi

Stack and Heap: C++

```
int main() {
   // Declare an object
   SomeClass obj1("obj1");

   // Create a pointer to an object
   SomeClass *obj2 = new SomeClass("obj2");

   //  Call a method
   obj1.printSomething();

   obj2->printSomething();

   return 0;
}
```

Stack and Heap: C++

```
int main() {
   // Declare an object
   SomeClass obj1("obj1");

   // Create a pointer to an object
   SomeClass *obj2 = new SomeClass("obj2");

   //  Call a method
   obj1.printSomething();

   obj2->printSomething();

   return 0;
}
```

Created on the stack

Stack and Heap: C++

```
int main() {
   // Declare an object
   SomeClass obj1("obj1");

   // Create a pointer to an object
   SomeClass *obj2 = new SomeClass("obj2");

   //  Call a method
   obj1.printSomething();

   obj2->printSomething();

   return 0;
}
```

Created on the heap

See that objects behave as they should, based on polymorphic behaviour

Static typing ensures that objects should be capable at compile time

Compiler

Variable type determines
what it can do…

Dynamic typing provides flexibility with checks for object capabilities at run time

Object

Object determines what it can do…

No need to indicate its type

Some languages have a mix of static and dynamic typing features

Compiler

Object

Watch out how how different objects are destroyed

Manual memory management makes freeing memory the developer’s problem

Developer needs to know when to delete things.
Deconstructors are used to propagate deletion to relations

Example: C++
new = malloc
delete = free

Deconstructors in C++

```
class SomeClass
{
   private:

   // Data Members
   string _someString;
    SomeOtherClass *_someObject;

   public:

   SomeClass(string str)
   {
        _someString = str;
         _someObject = (SomeOtherClass *) new SomeOtherClass();
   }

   ~SomeClass()
   {
      delete &_someObject;
   }

};
```

Memory dealloc in C++

```
class SomeClass
{
   private:

    // Data Members
    string _someString;
     SomeOtherClass *_someObject;

   public:

    SomeClass(string str)
    {
         _someString = str;
          _someObject = (SomeOtherClass *) new SomeOtherClass();
    }

    ~SomeClass()
    {
      delete &_someObject;
    }

};
```

deconstructor

Objects created on the heap must be destroyed when finished with

Reference counting allows objects to free themselves

Objects know how many things refer to them.
Delete themselves when this is 0.

Example: Objective C, Swift

Garbage collection means the system handles object deletion for you

Graph of object references is maintained.

Garbage collection triggered automatically under specific conditions (e.g., when the consumed heap space crosses a threshold)

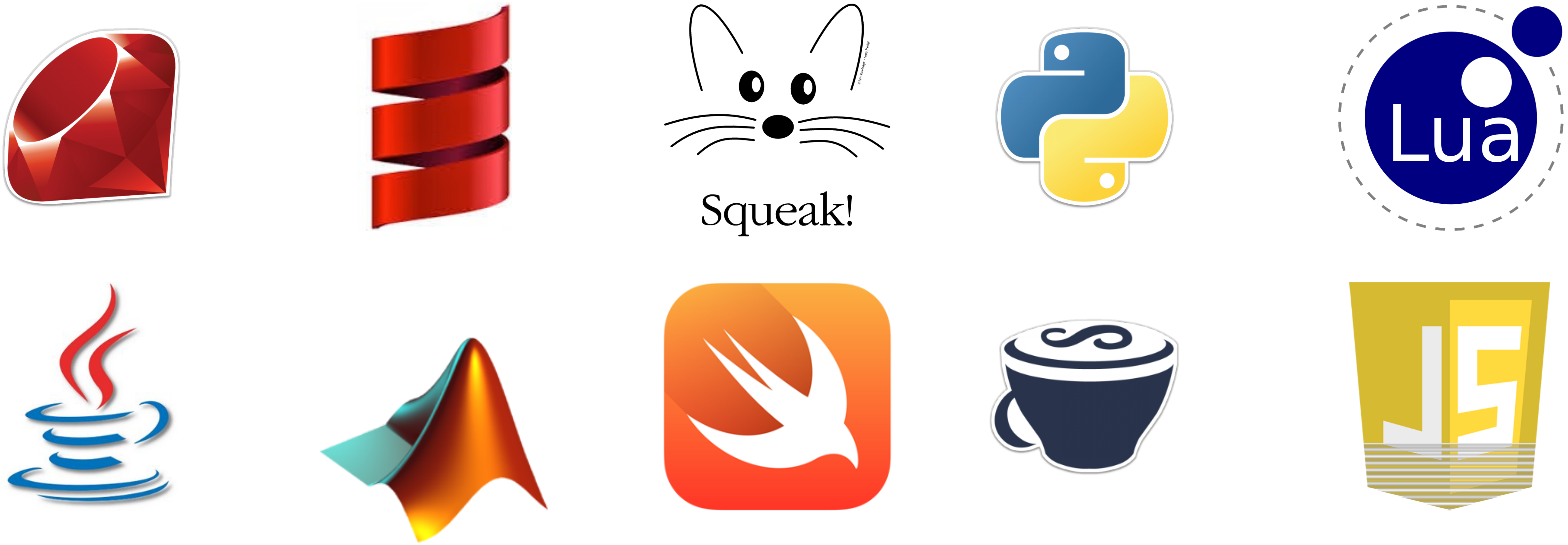
Sees objects as belonging to “generations”: from short-living to long-term

Example: Java / C#

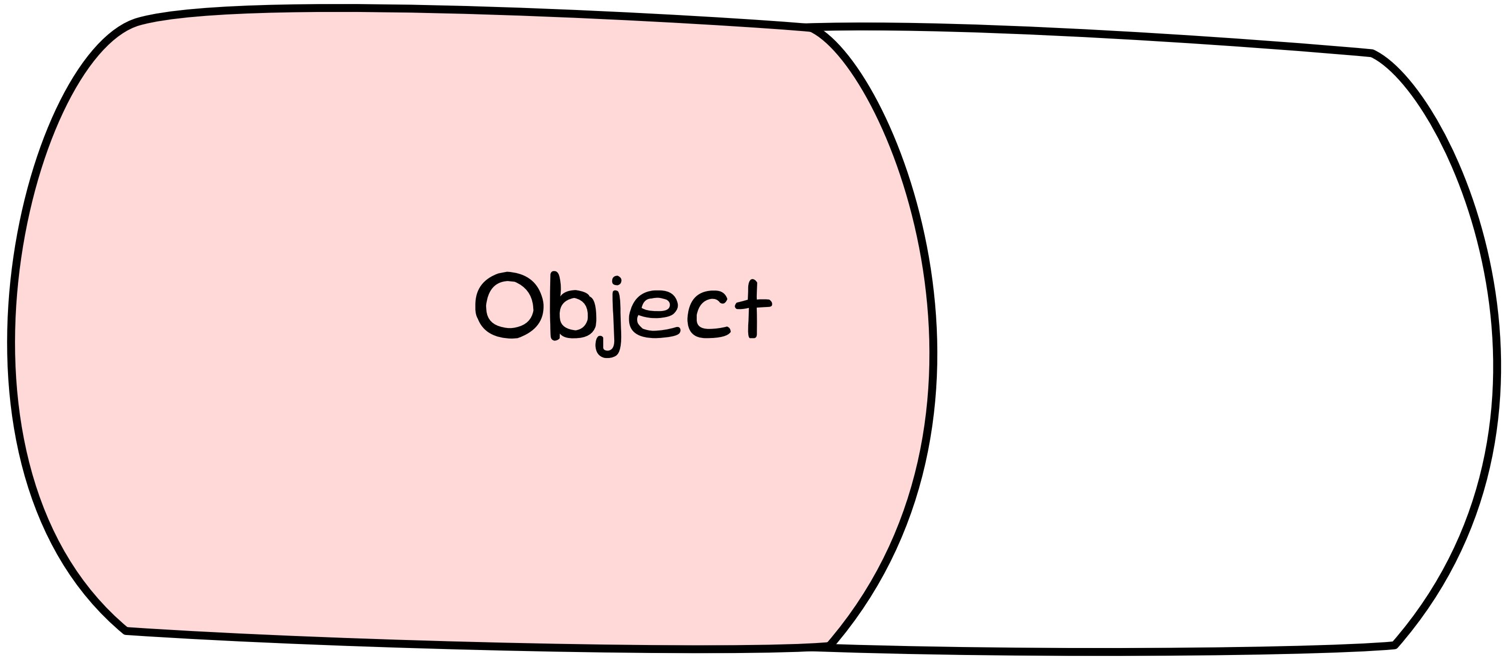
Will you be able to pickup other OO programming languages?

Software in industry is dominated by OO programming languages

Approach new languages armed with the principles of OOP



Remember, in each case it is about objects that know and can do things



OO Languages: coding based on abstraction, encapsulation, inheritance and polymorphism