
The Feel of Objective-C
Kresten Krab Thorup, Ph.D.
CTO, Trifork
krab@trifork.com
Credits and Thanks to Glenn Vanderburg, Relevance LLC

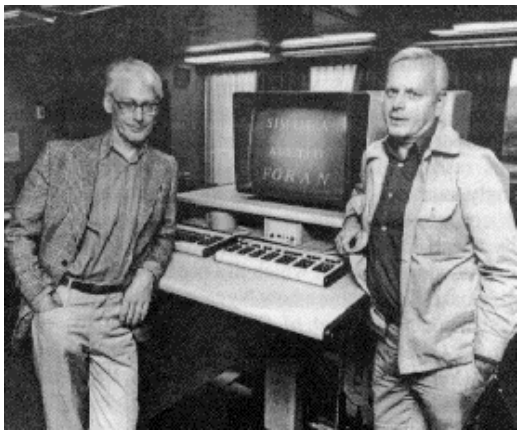
Objective-C

- Start with C
- Add the Smalltalk object model as a library
- Add a little syntax for
  - Class and method definition
  - Method calls
  - A few object literals

Bits of History

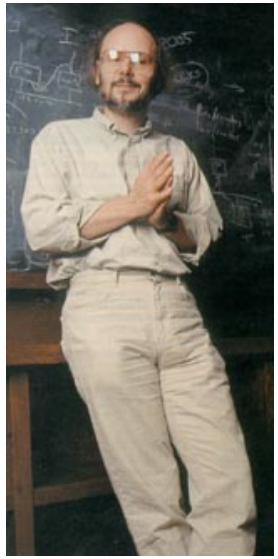
simula

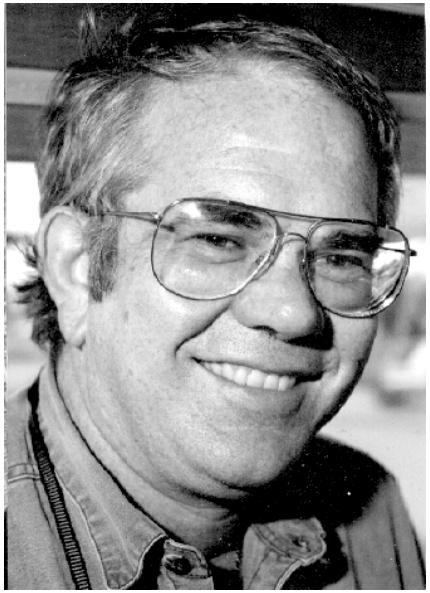
SOFTWARE DEVELOPMENT



TRIFORK.

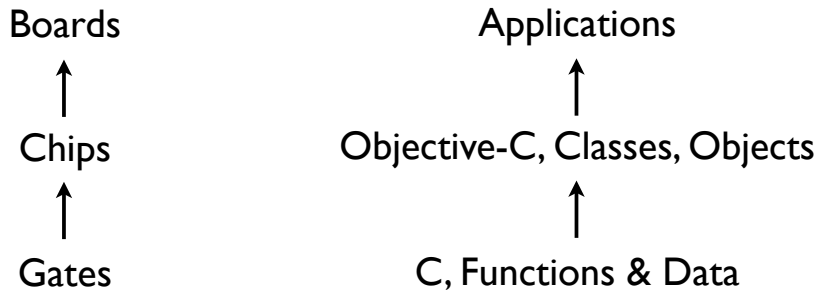

C++



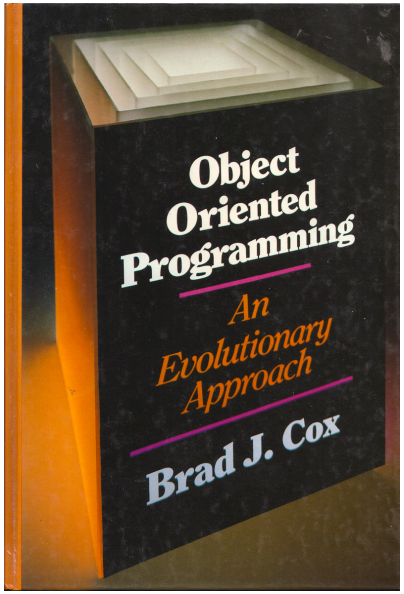


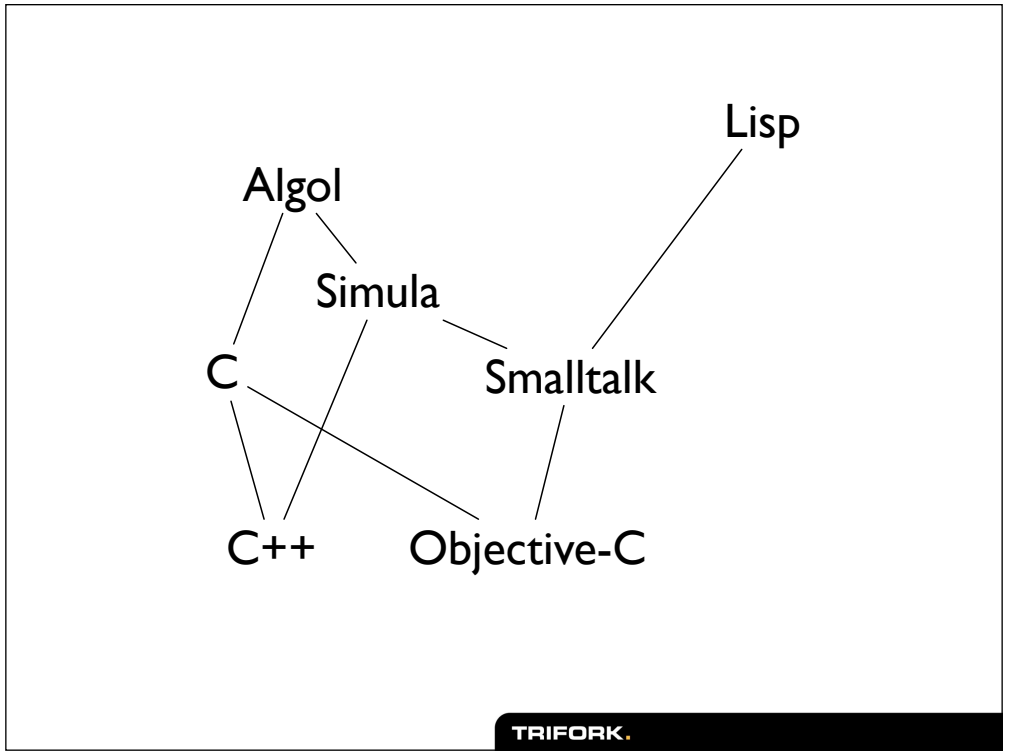
TRIFORK.

Brad Cox's Thinking



Objective-C


Object
Oriented
Programming
An
Evolutionary
Approach
Brad J. Cox


Lisp
Algol
Simula
C
Smalltalk
C++
Objective-C
TRIFORK.

TRIFORK.

10 minutes

C++

- Carefully infused OO into every part of C
- New syntax integrated into C grammar
- "OO the C way"
  - Efficiency a core concern
  - Compiler does all the work
  - "Don't pay for what you don't use."

Objective-C

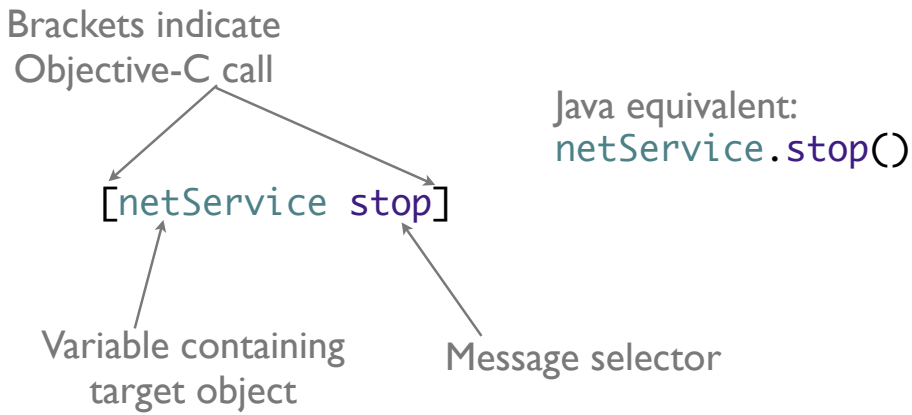
- A mashup of two languages
- Smalltalk grafted onto C
- The boundaries are obvious:
  - Non-C-like syntax in special “zones”
  - Flag characters to mark Objective-C zones
  - In C code, objects are opaque

C++ vs. Objective-C

- At first glance:
  - C++ a serious effort
  - Objective-C a hack job
- The reality is much different:
  - C++ has serious faults, is widely loathed
  - Objective-C is a useful, pragmatic hack

Objective-C:
The Language

Calling Methods



Java equivalent:

```
netService.stop()
```

Methods With Arguments

```
[serviceNameField setEnabled:YES]

[in_stream read:readBuffer maxLength:4096]
```

(Yes, that method name is “read:maxLength:”)

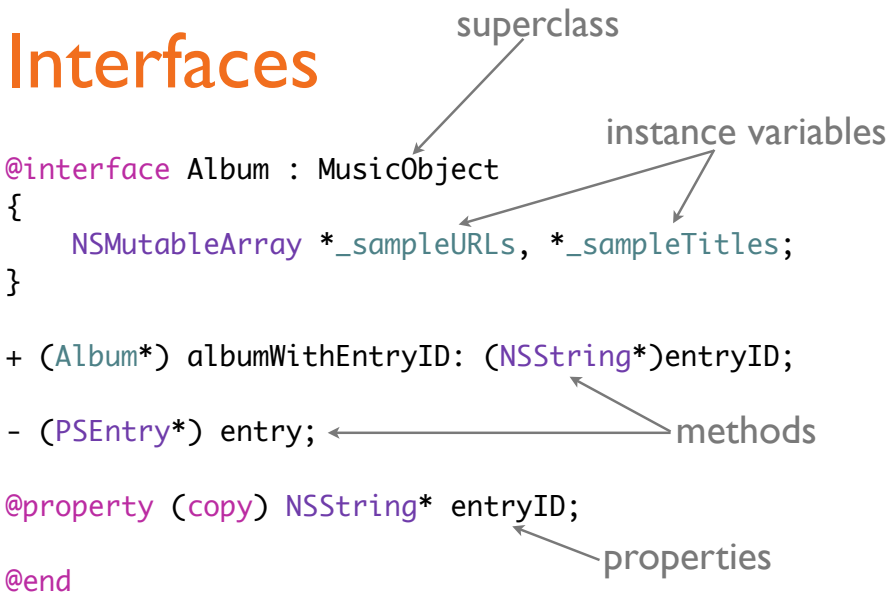
Declaring Methods

```
// ‘+’ indicates class method
+ (Album*) createAlbumFromEntry: (PSEntry*)entry;


// ‘-’ indicates instance method
- (PSEntry*) entry;

// Here’s a variable-length argument list:
- (NSArray*) arrayWithObjects:firstObject, ...;
```

Defining Methods

```
// ‘+’ indicates class method
+ (Album*) albumWithEntryID: (NSString*)entryID
{
    return [self instanceWithValue: entryID
                 forKey: @"entryID"];
}

// ‘-’ indicates instance method
- (PSEntry*) entry
{
    return [_client entryWithIdentifier: _entryID];
}
```

Interfaces


superclass
instance variables
@interface Album : MusicObject
{
NSMutableArray *_sampleURLs, *_sampleTitles;
}
+ (Album*) albumWithEntryID: (NSString*)entryID;
- (PSEntry*) entry;
methods
@property (copy) NSString* entryID;
properties
@end

NSWhat?

- Objective-C has no namespaces
- Libraries (and apps) use prefixes instead
- Many type names begin with “NS” — for NeXTStep

Implementations

```
// Album.m

@implementation Album

// method definitions go here

@end
```

Types

- Object variables are usually pointers
  - e.g., NSString *
- Methods can return any C type
  - including object pointers
  - use Objective-C method call anywhere an expression is valid
- Parameters can also be any C type

Basic Types

- NSNumber, NSInteger
- NSString
  - special literal syntax: @"foo"
- NSMutableString
- NSArray and NSMutableArray
- NSDictionary and NSMutableDictionary

Allocation

`[NSAlert alloc]`	Allocates unitialized object
`[new_object init]`	Performs default initialization
`[[NSAlert alloc] init]`	Standard init pattern
`[NSAlert new]`	Rarely used equivalent

```
NSAlert *alertSheet;
alertSheet = [[NSAlert alloc] init];
```

Initialization

```
[[NSString alloc] init ]

[[NSString alloc] initWithString: username]

[[NSString alloc] initWithFormat:@"%@/%@",
                  parentAbsPath, relativePath]

[[NSString alloc] initWithBytes:value length:strlen(value)]

[[NSString alloc] initWithBytes:value length:strlen(value)
                  encoding:NSASCIIStringEncoding]

[[NSString alloc] initWithData: data
                  encoding: NSUTF8StringEncoding]

[[NSString alloc] initWithContentsOfFile: path]
```

Special values

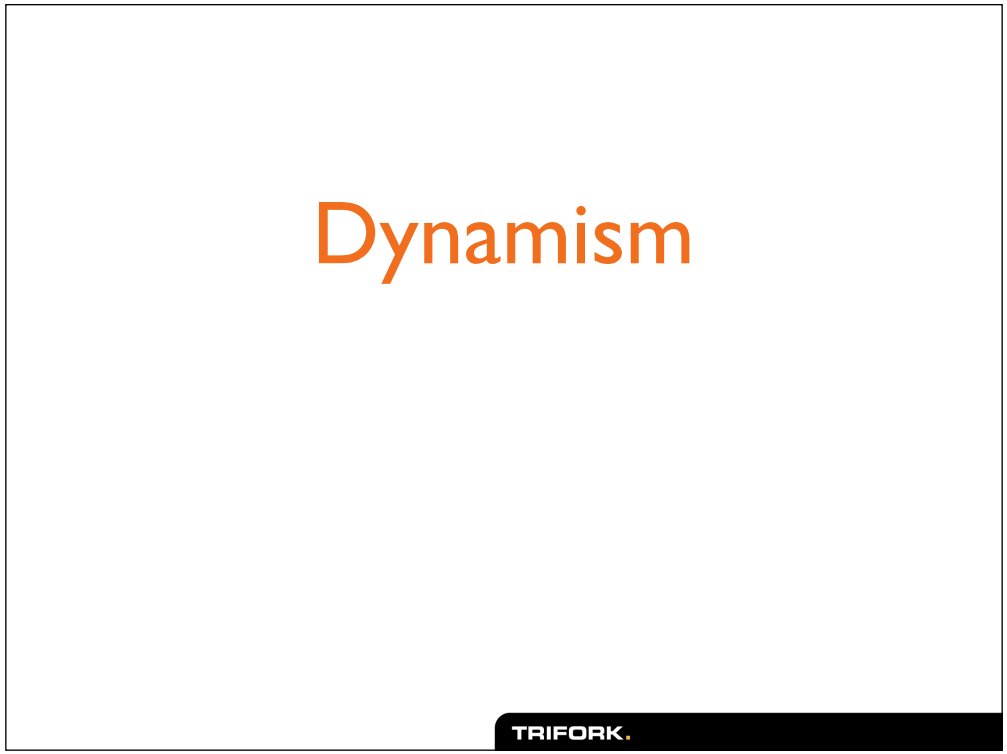
- self
- super
- nil

Memory Management

- Objective-C v4 supports garbage collection
  - (but not on the iPhone, yet...)
- Manual reference counting

```
[obj retain]
```

```
[obj release]
```



30 minutes

Bundles

- NeXT's Objective-C was early to adopt dynamic loading of code, and now unloading.
- In Objective-C, this is embodied in the concept of a Bundle, which is a loadable module containing code and data (resource file).
- In practice, it's a directory named .bundle, which holds the relevant artifacts.

Incremental Typing

- Usually, Objective-C is statically typed
  - (or as static as C will allow)
- The typedef `id` represents "any Objective-C object"
- You can write methods that work on any type

Incremental Typing

- Method parameters with no type default to the special type `id`
- Works great for starting a project with no, or little typing information
- Gradually add type information to your classes as they get more users, or to increase confidence in the

Protocols

- In Smalltalk terminology, a ‘protocol’ is a set of methods that may be implemented by many classes.
- In Objective-C, this was formalized to resemble what you may know as an ‘interface’ in Java.

Protocols

```
@protocol KeyValueAccess

- valueForKey:(NSString*)key;
- setValue:(id)val forKey:(NSString*)key;

@end
```


```
id <KeyValueAccess> obj = ...;
[obj setValue:@”Peter” forKey:@”name”];
```

Protocols

```
// intersection types
id <InputStream, OutputStream> stream = ...

// or even...
NSFooBar <KeyValueAccess> foobar = ...;

// In Java, such types can be used to
// declare Class parameter constraints...
```

Categories

- Categories are collections of methods that you add to some other class.
  - Similar to Ruby mixins
- They can be added to classes you don't have the source to — even things like NSString!
- Extend library classes to fit your application.

Categories

category name

```
@interface NSString (reverse)
- (NSString*)reverse;
@end
```

category name

```
@implementation NSString (reverse)
- (NSString*)reverse {
  ...
}
@end
```

Categories

- Extend classes & Alternative to subclassing
- Distribute class code 'aspect oriented' [DCI]
  - Say, for an interpreter, the base classes may be the abstract syntax tree
  - One category adds the type checker for all nodes...
  - Another category adds the evaluation

Reflection/Introspection

- Objective-C has rich support for reflection
  - Learn the type of an object
  - Learn about methods
  - Does this object support method foo?
  - Call methods dynamically

The Runtime System

- Remember, Objective-C is just a runtime library + some helpful syntax.
- You can access that library directly:
  - Dynamically create an instance of a class
  - Catch and handle calls to missing methods

Verbosity Fetish

```
[ row objectAtIndex: item ]

[ row insertObject: @"foo" atIndex: item ]

[ newKernel setDataModifiedFromOriginal: NO ]
```

Frameworks

- Most Objective-C libraries are called “frameworks”
- Don’t fit the usual definition of “framework”
- Essentially a library with extra use/packaging information for IDE

A Step Backward

- Manual memory management
- Buffer overflows, core dumps
- Dust off your old C/C++ debugging skills

Signatures!

$$h = \sqrt{x^2 + y^2}$$

```
- (float)calcLength:(float)x and:(float)y
{
   return sqrt(x*x + y*y);
}

// In Intel ABI, floats are passed in the
// FPU registers, integers on the stack.

int h = [obj calcLength:3 and:5]
```

Fuzzy Boundaries

- Many things implemented as ordinary C functions or macros.
- Many important types *not* defined as objects.
  - (usually for efficiency)
- Difficult to remember where the boundaries are.

The Feel of Objective-C

Kresten Krab Thorup, Ph.D.
CTO, Trifork
krab@trifork.com

TRIFORK.