# Objective-C

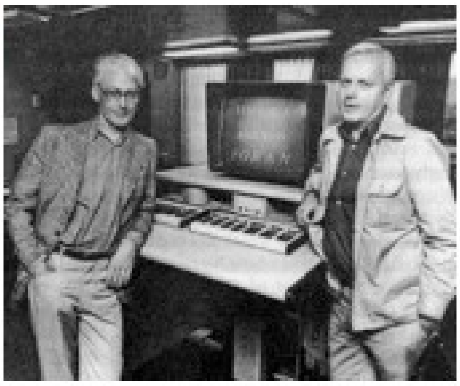
Kresten Krab Thorup
Trifork A/S
krab@trifork.com

Credits: Glenn Vanderburg, Relevance, Inc.

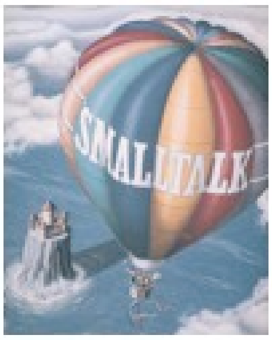
## Bits of History

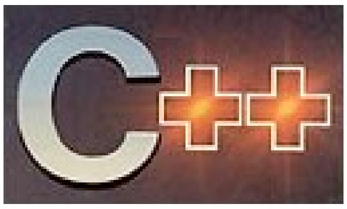
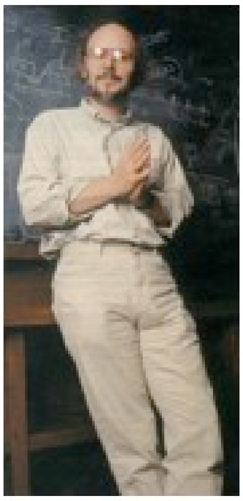
TRIFORK.

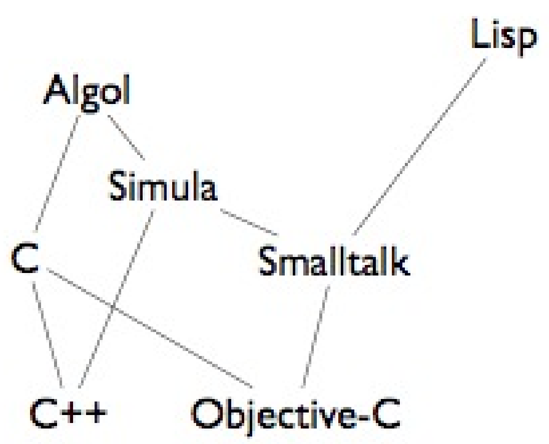
simula
SOFTWARE DEVELOPMENT

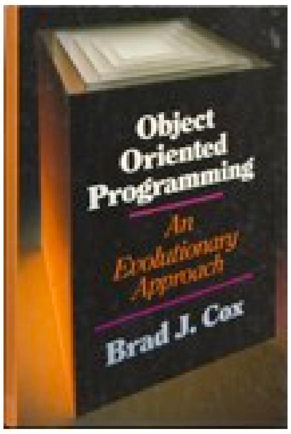


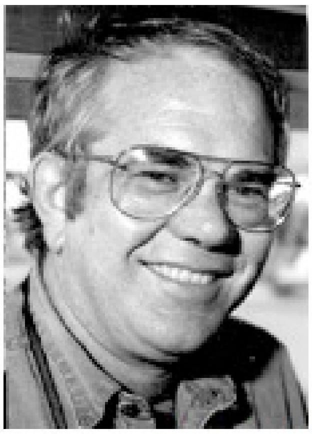
TRIFORK.


SMALLTALK


C++


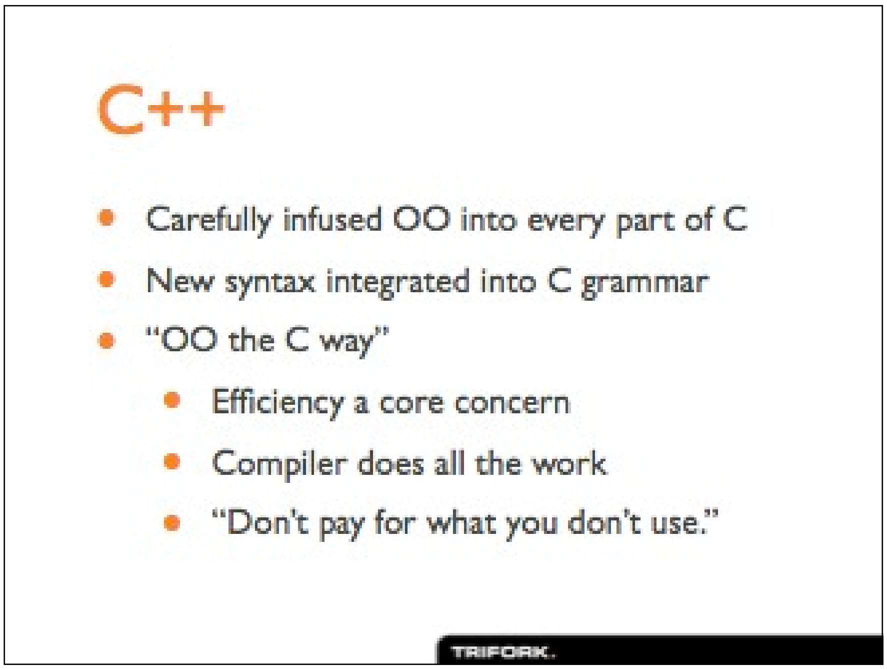

Lisp
Algol
Simula
C
Smalltalk
C++
Objective-C





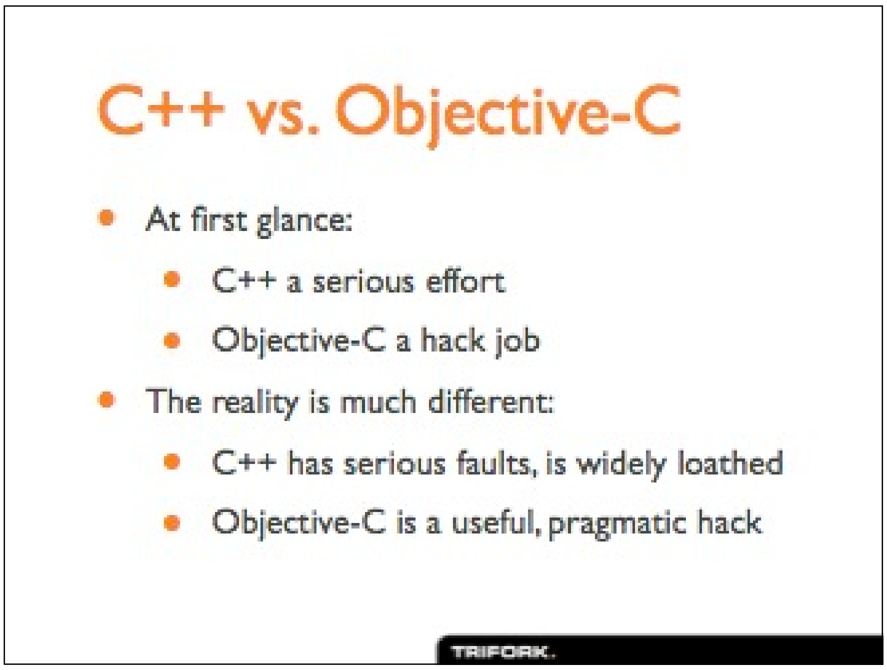
Brad Cox

## The Road Not Taken



## C++

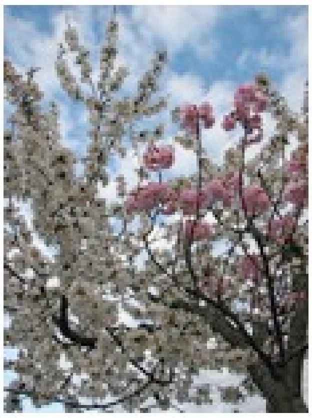
- Carefully infused OO into every part of C
- New syntax integrated into C grammar
- "OO the C way"
  - Efficiency a core concern
  - Compiler does all the work
  - "Don't pay for what you don't use."



## 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

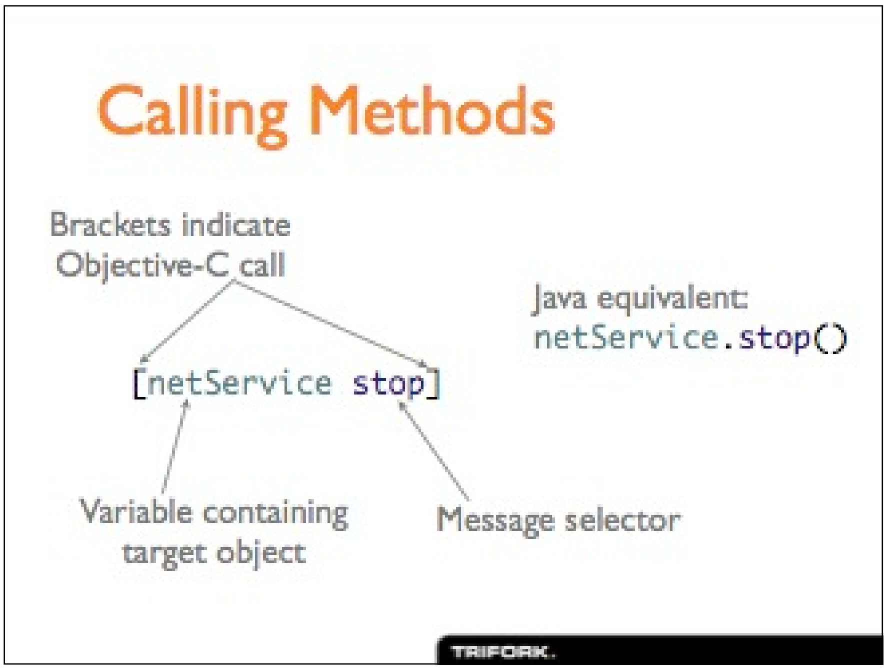


- A mashup of two languages
- Smalltalk grafted onto C
- The boundaries are obvious:
  - Non-C-like syntax in special "zones"
  - Flag characters to mark Objective-C zones
  - In C code, objects are opaque

## Objective-C: The Language

## 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



## Calling Methods

Brackets indicate Objective-C call

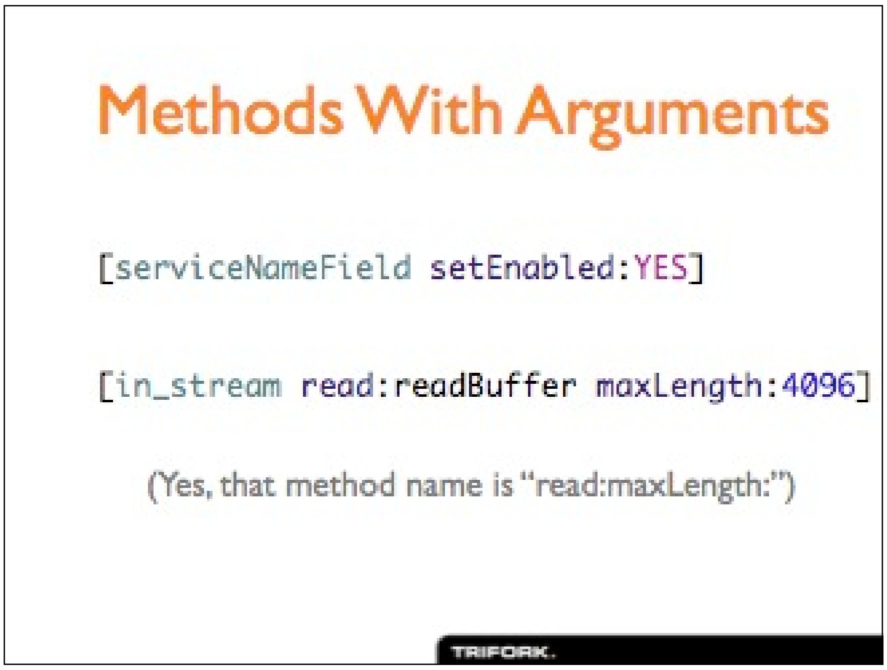
```
[netService stop]
```

Variable containing target object

Message selector

Java equivalent:

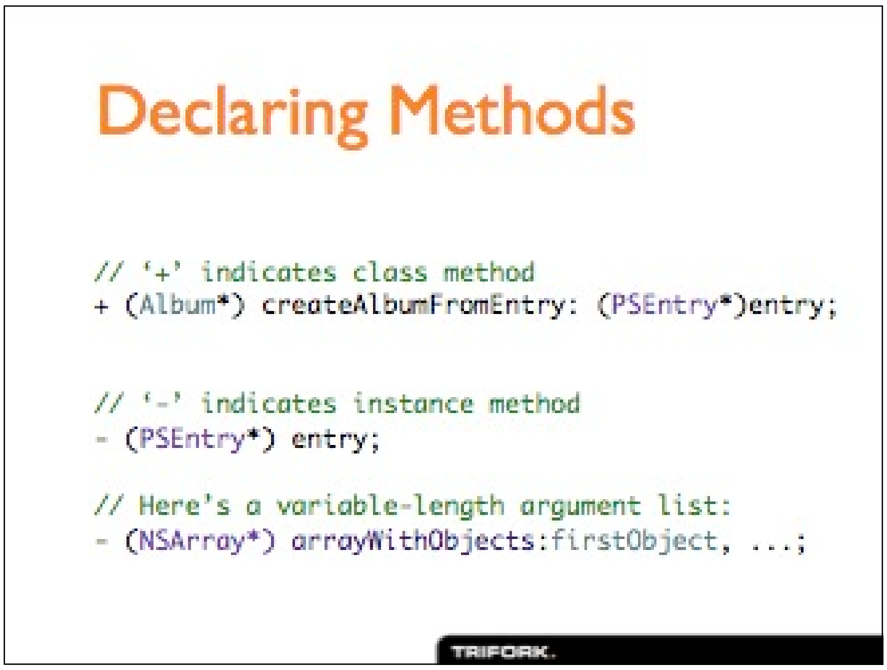
```
netService.stop()
```



## Methods With Arguments

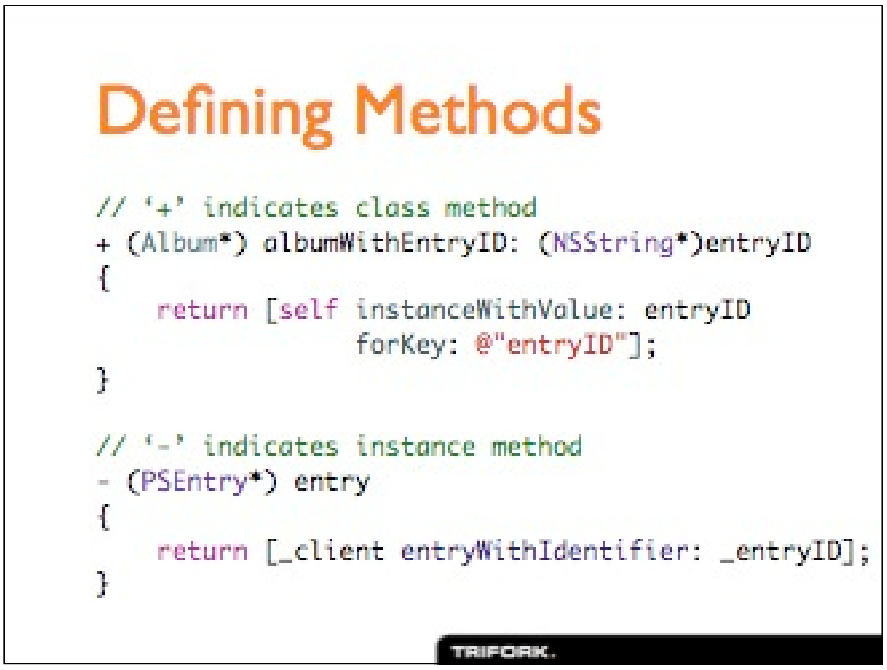
```
[serviceNameField setEnabled:YES]

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

(Yes, that method name is "read:maxLength:")



## Declaring Methods

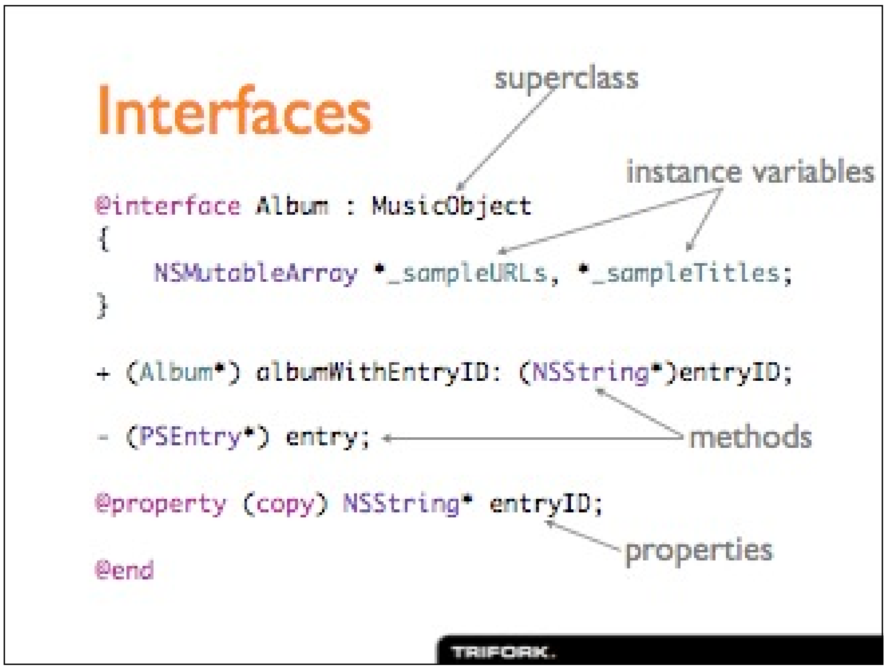
```
// '+' indicates class method
+ (Album*) createAlbumFromEntry: (PSEntry*)entry;

// '-' 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];
}
```

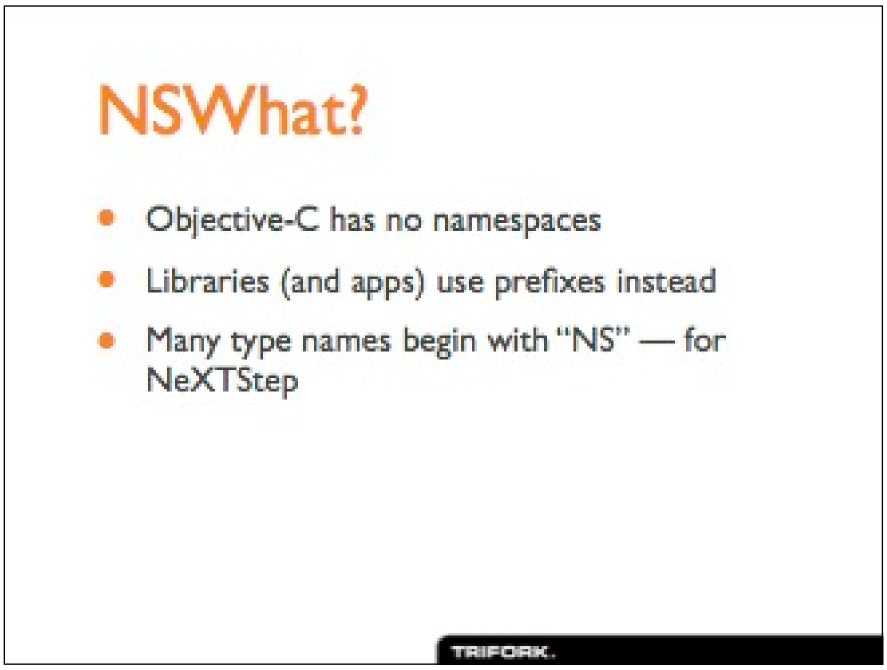
TRIFORK.



## Interfaces

```
@interface Album : MusicObject
{
    NSMutableArray *_sampleURLs, *_sampleTitles;
}

+ (Album*) albumWithEntryID: (NSString*)entryID;

- (PSEntry*) entry;

@property (copy) NSString* entryID;

@end
```

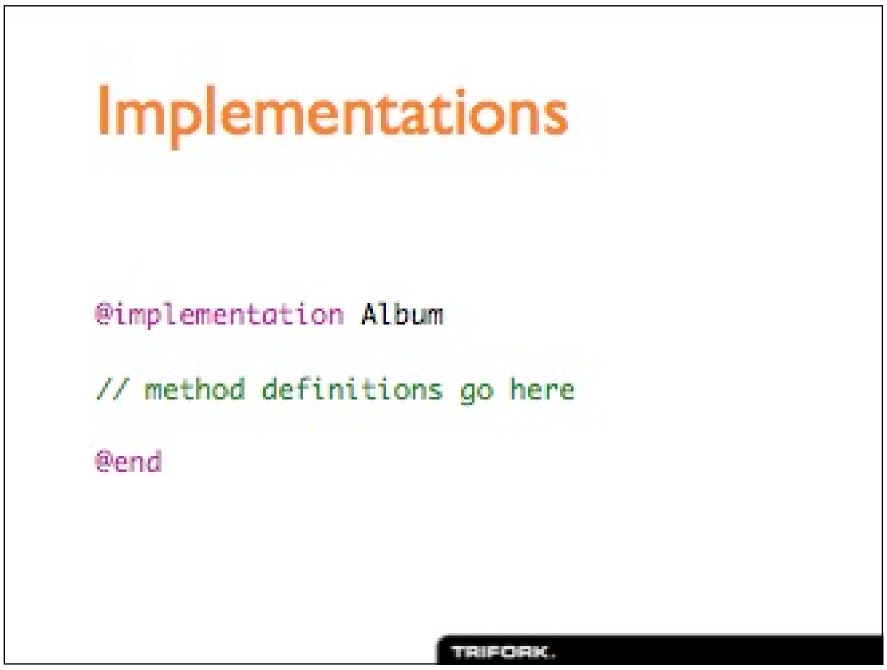
TRIFORK.



## NSWhat?

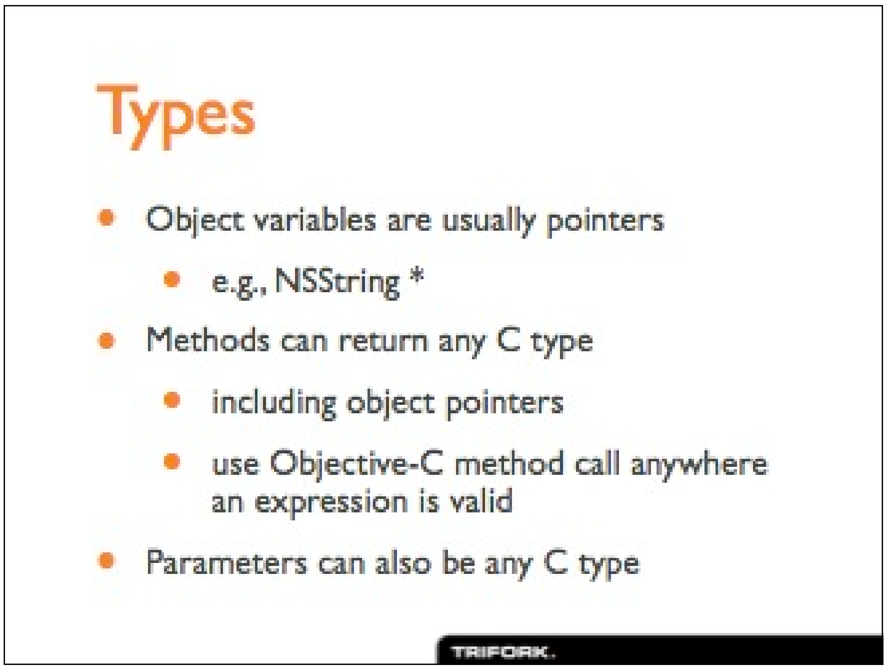
- Objective-C has no namespaces
- Libraries (and apps) use prefixes instead
- Many type names begin with "NS" — for NeXTStep

TRIFORK.



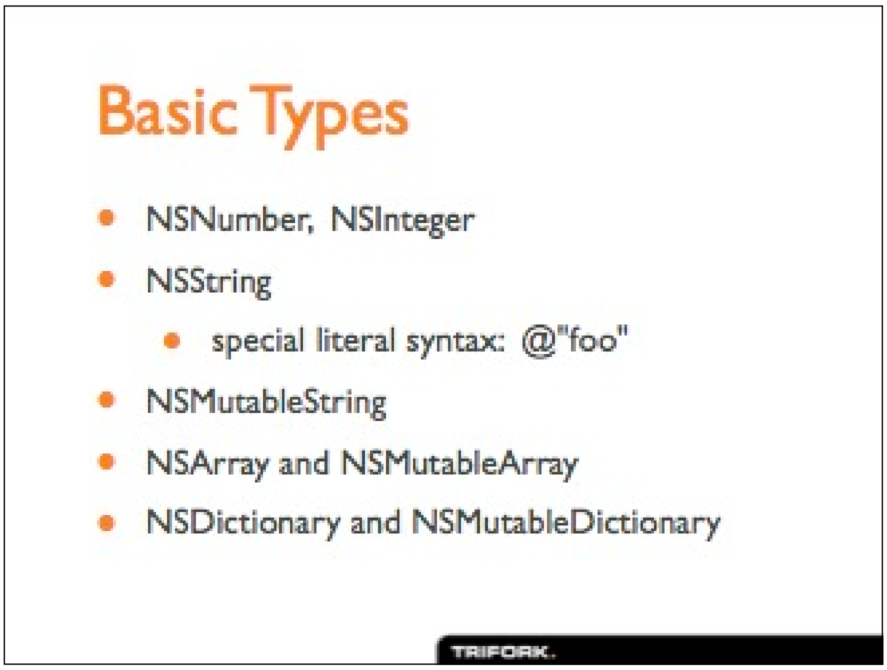
## Implementations

```
@implementation Album

// method definitions go here

@end
```



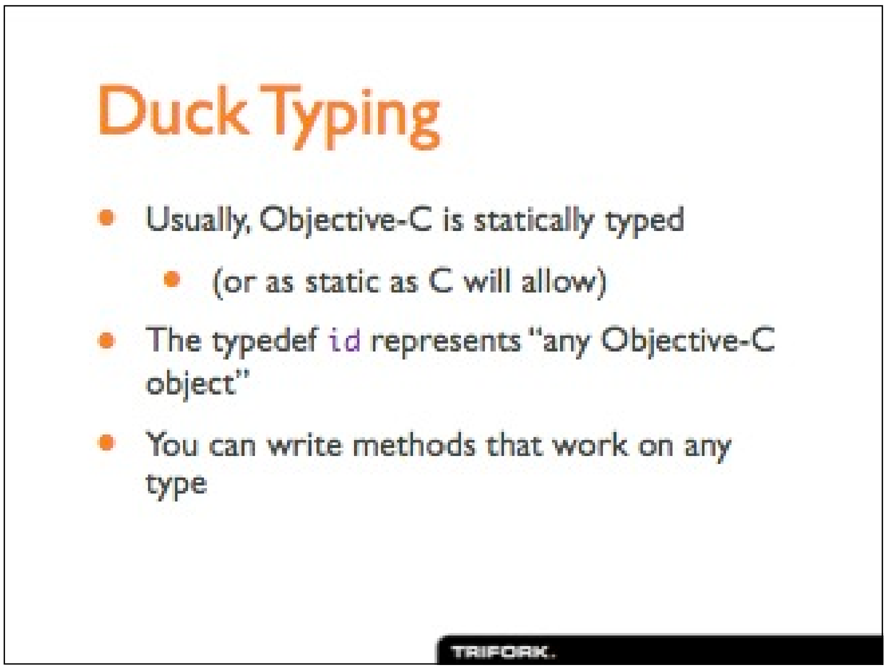
## Types

- 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



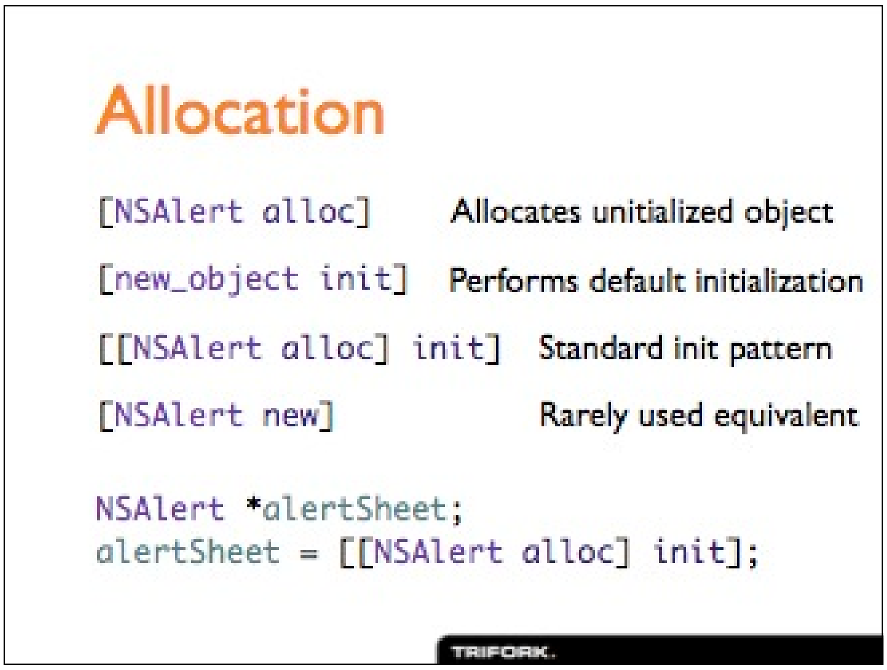
## Basic Types

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



## Duck Typing

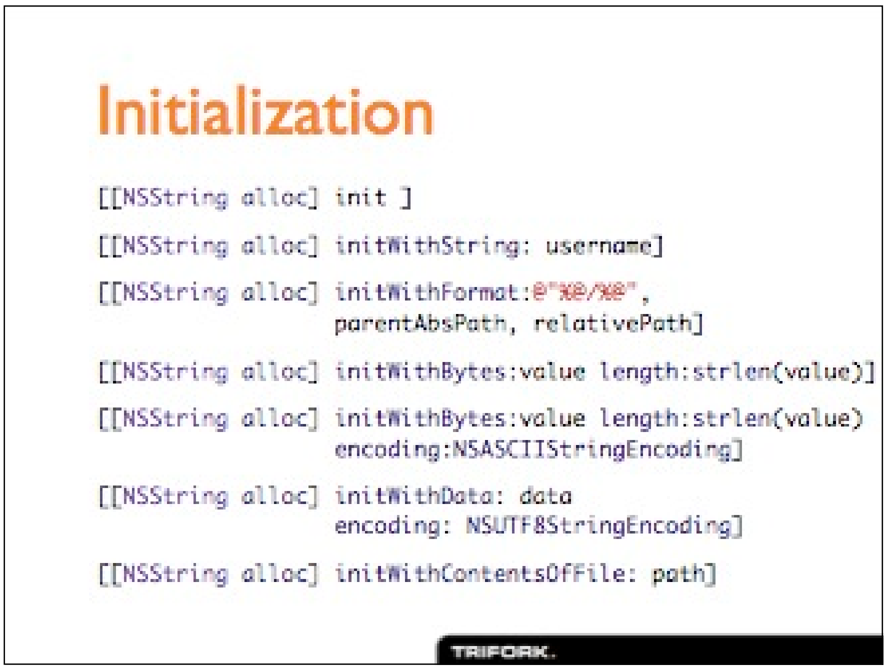
- Usually, Objective-C is statically typed
    - (or as static as C will allow)
- The typedef `id` represents "any Objective-C object"
- You can write methods that work on any type



## Allocation

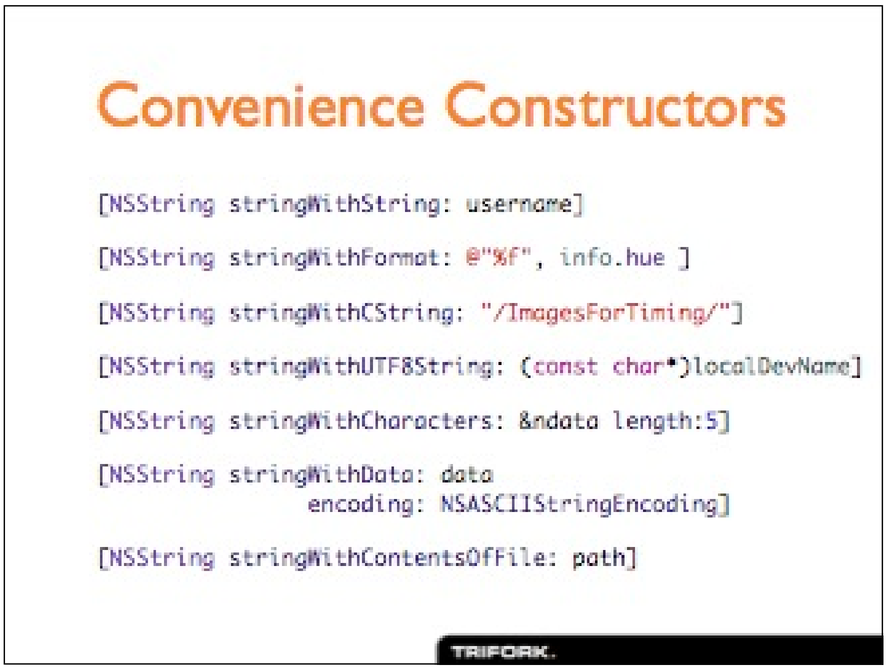
| | |
|---|---|
| `[NSAlert alloc]` | Allocates unitialized object |
| `[new_object init]` | Performs default initialization |
| `[[NSAlert alloc] init]` | Standard init pattern |
| `[NSAlert new]` | Rarely used equivalent |

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



## Initialization

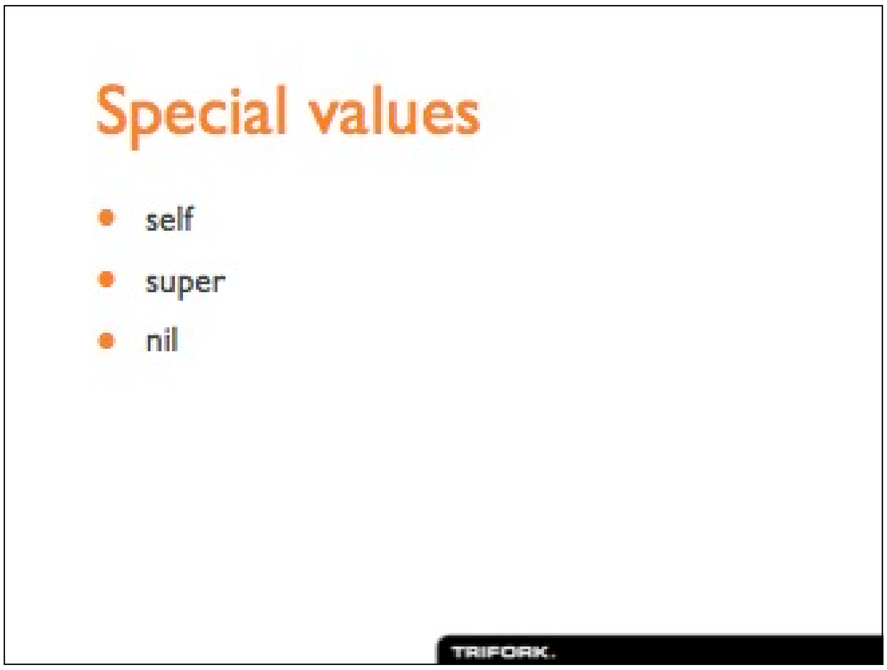
```
[[NSString alloc] init ]
[[NSString alloc] initWithString: username]
[[NSString alloc] initWithFormat:@"%@/%@",
                  parentAbsPath, relativePath]
[[NSString alloc] initWithBytes:value length:strlen(value)]
[[NSString alloc] initWithBytes:value length:strlen(value)
                  encoding:NSASCIIStringEncoding]
[[NSString alloc] initWithData: data
                  encoding: NSUTF8StringEncoding]
[[NSString alloc] initWithContentsOfFile: path]
```



## Convenience Constructors

```
[NSString stringWithString: username]

[NSString stringWithFormat: @"%f", info.hue ]

[NSString stringWithCString: "/ImagesForTiming/"]

[NSString stringWithUTF8String: (const char*)localDevName]

[NSString stringWithCharacters: &ndata length:5]

[NSString stringWithData: data
                encoding: NSASCIIStringEncoding]

[NSString stringWithContentsOfFile: path]
```

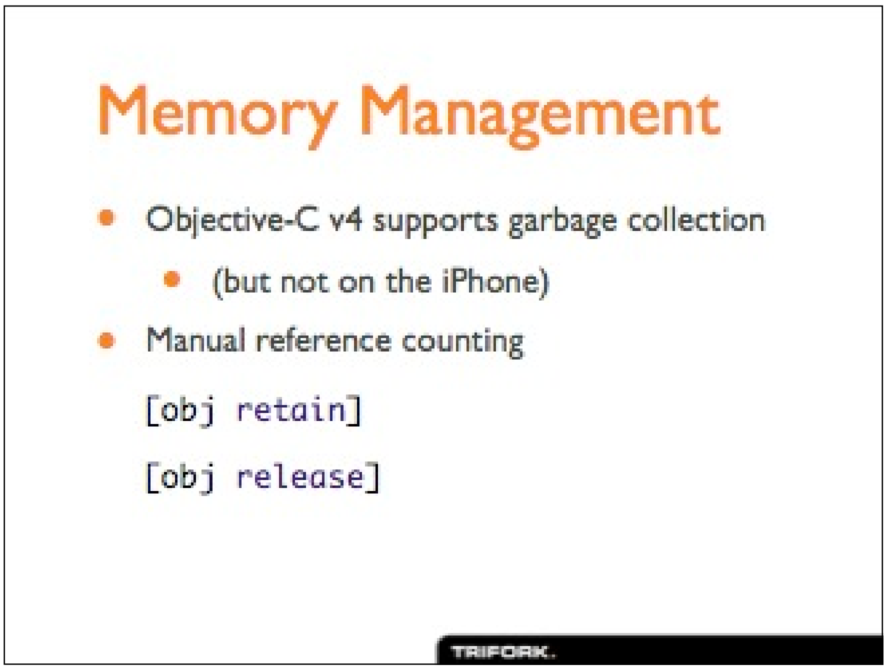
TRIFORK.



## Special values

- self
- super
- nil

TRIFORK.

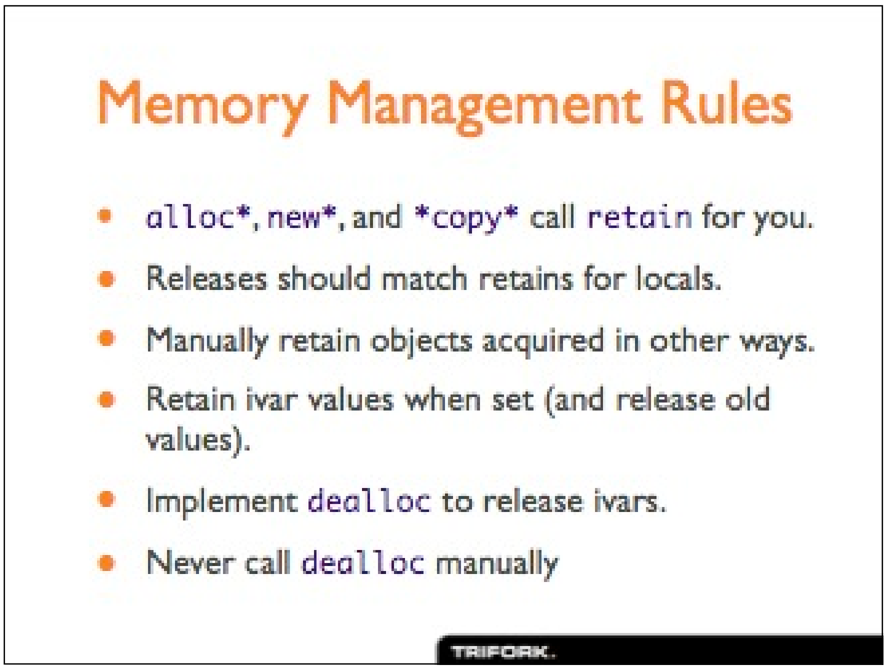


## Memory Management

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

```
[obj retain]

[obj release]
```

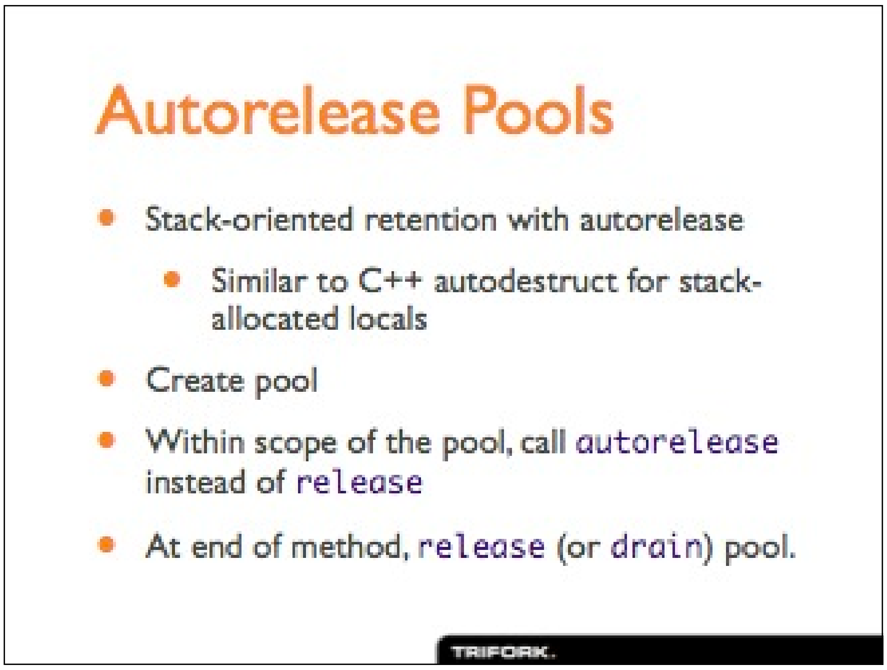
TRIFORK.



## Memory Management Rules

- `alloc*`, `new*`, and `*copy*` call `retain` for you.
- Releases should match retains for locals.
- Manually retain objects acquired in other ways.
- Retain ivar values when set (and release old values).
- Implement `dealloc` to release ivars.
- Never call `dealloc` manually

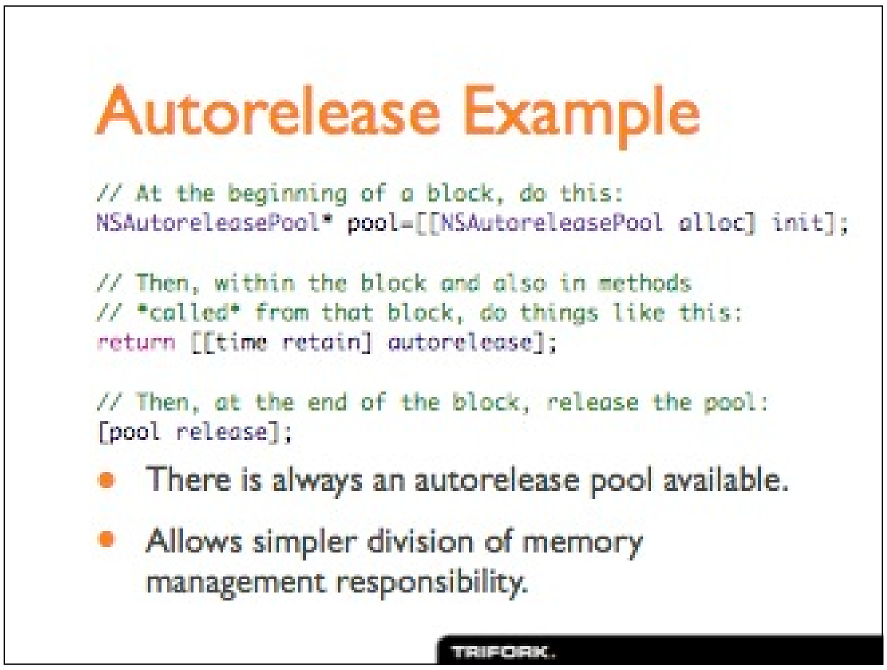
TRIFORK.



## Autorelease Pools

- Stack-oriented retention with autorelease
  - Similar to C++ autodestruct for stack-allocated locals
- Create pool
- Within scope of the pool, call `autorelease` instead of `release`
- At end of method, `release` (or `drain`) pool.

TRIFORK.

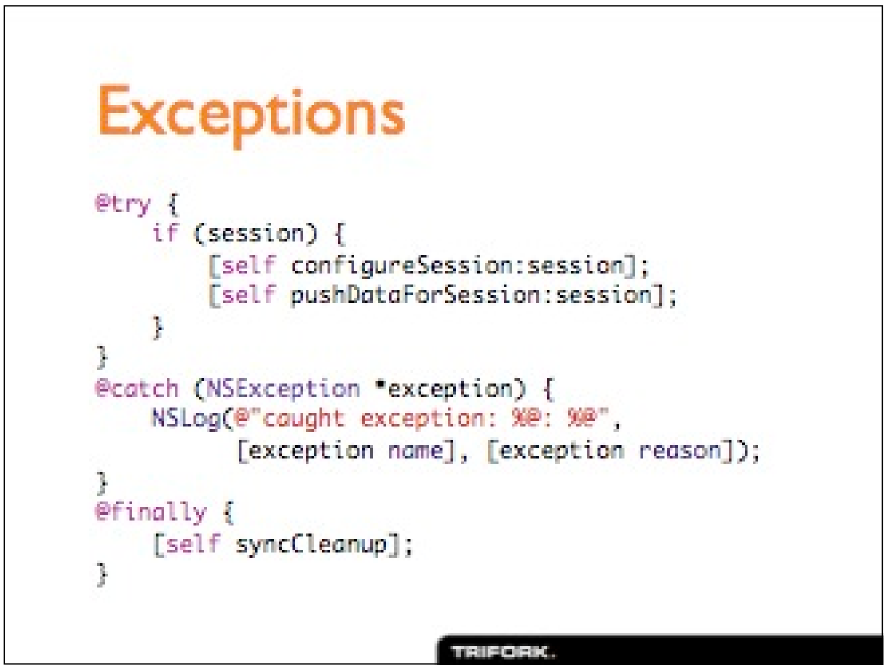


## Autorelease Example

```
// At the beginning of a block, do this:
NSAutoreleasePool* pool=[[NSAutoreleasePool alloc] init];

// Then, within the block and also in methods
// *called* from that block, do things like this:
return [[time retain] autorelease];

// Then, at the end of the block, release the pool:
[pool release];
```

- There is always an autorelease pool available.
- Allows simpler division of memory management responsibility.

TRIFORK.



## Exceptions

```
@try {
    if (session) {
        [self configureSession:session];
        [self pushDataForSession:session];
    }
}
@catch (NSException *exception) {
    NSLog(@"caught exception: %@: %@",
          [exception name], [exception reason]);
}
@finally {
    [self syncCleanup];
}
```

Kresten Krab Thorup
Trifork A/S
krab@trifork.com

Credits: Glenn Vanderburg, Relevance, Inc.