

Objective C, una intro per javisti

Ugo Landini
ugol@computer.org

“Chi non conosce la
storia, è condannato a
ripeterla.”

G. Santayana

Forse non tutti sanno che...

- La programmazione ad oggetti così come la conosciamo nasce nei laboratori Xerox, Palo Alto Research Center (PARC)
- Alan Kay fu la figura chiave: inventò il mouse, le finestre, gli eventi e anche Smalltalk
- Siamo negli anni '70. Il resto del mondo è procedurale: scrive Cobol, si afferma Unix e si inizia ad usare il C.

Smalltalk

- Si narra che Steve Jobs e Steve Wozniak, in visita al PARC, recepirono molto bene mouse e finestre: nacque Lisa e poi il Mac, che cambiò la storia del personal computing
- recepirono meno bene Smalltalk (o forse non era pronto per far girare un sistema operativo)
- Smalltalk divenne però il papà a cui tutti si ispirarono

Objective C

- Inventato nel 1986 da Brad Cox e Tom Love, ispirandosi direttamente a Smalltalk
- Descritto nel libro “Object Oriented Programming, an evolutionary approach”, che è ancora una buona lettura
- NON è della famigerata scuola nordica (Stroustrup), e rispetta pienamente i principi OO

NeXT

- Steve Jobs lascia Apple nel 1986 e fonda Next computers
- Evidentemente il messaggio di Alan Kay era comunque arrivato, perchè Steve Jobs sceglie Objective C, un diretto erede di Smalltalk, come linguaggio per il SO delle sue workstation, NextStep

NeXT

- Le workstation NeXT non hanno fortuna commerciale, però sono apprezzatissime dai tecnici
 - sono le prime ad avere ethernet di serie
 - il primo Web Server (al CERN) è un NeXT
 - ID Software ci scrive Doom, Doom II e Quake

Il primo Web Server



Il primo Web Server



Da NeXT ad Apple

- Apple compra NeXT nel 1996, e Steve Jobs ritorna dunque nella sua azienda come CEO
- Objective C diventa mainstream perchè è il linguaggio di OSX
 - e dunque dell'iPhone...
- Objective C 2.0 è per ora legato a Leopard, in attesa di GNUStep.

Riassumendo...

- Objective C è un linguaggio ad oggetti antecedente Java
- Objective C è figlio diretto di Smalltalk
- Objective C è il linguaggio di OSX e dunque di iPhone
- Objective C 2.0 è l'ultima release

Objective C, analisi del linguaggio

Cose che saltano all'occhio di un javista

- Sintassi ad oggetti Smalltalk-oriented e non C++/Java oriented
- Non esistono i costruttori
- Garbage Collection, ma solo dalla versione 2.0
- Light footprint, no VM.
- Set di istruzioni minimale costruito sopra il C

Message Passing

- Si usano le [] per indicare che si sta scrivendo objective C e non semplice C
- La signature del metodo è particolare per chi viene da java...

| Java | Objective C |
|---------------------------------------|--|
| <code>oggetto.faiQuesto();</code> | <code>[oggetto faiQuesto];</code> |
| <code>oggetto.faiDueCose(a,b);</code> | <code>[oggetto fai:a con:b];</code> |
| <code>string.toUpperCase();</code> | <code>[string upperCaseString];</code> |

Message Passing

- `[string writeToFile:@”path” atomically:YES];`
 - Si legge indubbiamente bene (a parte le quadre)
 - la signature è meno evidente, ed è
 - `writeToFile:atomically:`
- la dichiarazione del metodo è:
 - `-(BOOL)writeToFile:(NSString*)path atomically:(BOOL)useAuxiliaryFile`

Il problema del costruttore

- Immaginiamo in Java di non avere l'aiuto della JVM per i costruttori, ma solo dei semplici metodi
- Un costruttore potrebbe essere un metodo statico pubblico di una factory, e la factory potrebbe essere la classe stessa .class
 - `String s = String.new();`
 - `public static String new() {...}`

Il problema del costruttore

- Per avere una inizializzazione parametrica:
 - `public static String new(int x) {...}`
- Ma a questo il factory method deve:
 - o violare l'incapsulamento dell'oggetto per accedere ai suoi attributi
 - o utilizzare dei mutator (inefficiente)

Il problema del costruttore

- Per cui in Objective C il processo di costruzione è separato in due fasi diverse.
 - + alloc (class method) che alloca la memoria
 - - init (instance method) si chiede all'oggetto di inicializzarsi
- Non c'è nessun supporto speciale, sono due normali metodi: di fatto non esistono i costruttori

Una semplice classe

```
// Simple.h
```

```
#import <Cocoa/Cocoa.h>
```

```
@interface Simple : NSObject {  
    NSUInteger x;  
}
```

```
@property(readwrite) NSUInteger x;  
- (void)print;  
@end
```

Una semplice classe

```
// Simple.m
```

```
#import "Simple.h"
```

```
@implementation Simple
```

```
@synthesize x;
```

```
- (void)print
```

```
{
```

```
    NSLog(@"Hello world!");
```

```
}
```

```
@end
```

Una semplice classe

```
// Da qualche altra parte...
```

```
#import "Simple.h"
```

```
...
```

```
- (void)testSimple
```

```
{
```

```
    simple = [[Simple alloc] init];
```

```
    [simple setX: 42];
```

```
    STAssertEquals([simple x], 42, @"Universe is broken?");
```

```
}
```

Differenze visibili con Java e C

- Header file e implementation file
- : per indicare extends
- @ indica le parole chiave di Objective C
 - @"string" per differenziare le stringhe objc da quelle C
- - e + indicano metodi di istanza e di classe
- #import al posto di include (è un include intelligente)

Objective C vs C

- Objective C è stato pensato per essere implementato con un preprocessore
 - deve convivere con codice C
 - per questo ci sono decisioni “strane”, come le `[]` per i metodi o gli `@` per le nuove stringhe
- A parte qualche piccola stranezza, ci si fa presto l'abitudine.

Garbage Collection

- Objective C 2.0 supporta la GC
 - Come in java, non c'è molto di cui preoccuparsi in caso si scelga di usare la GC
- Il “vecchio” codice Objective C utilizza un altro meccanismo, un qualcosa di intermedio fra fra free/malloc ed un vero GC
- E' opportuno conoscere il meccanismo perchè ci sarà ancora per molto tempo tanto codice objc non GC (iPhone, per ora)

Memory Management

- Implementato con reference counting, ogni oggetto ha un contatore di quanti lo stanno usando: quando è a zero si può liberare
- Ma il contatore lo deve gestire lo sviluppatore
 - tramite i metodi alloc, retain, release e autorelease

Memory Management

```
// main.m
```

```
...
```

```
MyClass* myObject;
```

```
myObject = [[MyClass alloc] init];
```

```
...
```

```
[myObject release];
```

Memory Management

| Message | Description |
|---------------|--|
| + alloc | crea l'oggetto con refcount = 1 |
| - release | refcount = refcount - 1 |
| - autorelease | refcount = refcount - 1 (alla fine del metodo) |
| - retain | refcount = refcount + 1 |

- Il vantaggio di usare direttamente il Memory Management sono le prestazioni e la predicibilità: non ci saranno pause di GC
- Vedremo se iPhone 2.0 avrà o meno GC
 - il 9 giugno!

Categories

- Permettono di estendere una classe
 - anche senza averne il sorgente
 - qualsiasi classe, anche NSObject
 - alternativa all'ereditarietà
- Quante volte avreste voluto aggiungere un metodo alla classe String? In Objective C si può.

Categories

```
// Simple.h
```

```
#import <Cocoa/Cocoa.h>
```

```
@interface NSString(Simple)
```

```
- (void)print;
```

```
@end
```

- Si aggiunge il metodo print alla classe NSString
- non c'è modo di distinguere print dagli altri metodi "nativi"

Protocols

```
@protocol MyProtocol
```

```
- (void)add;
```

```
- (void)sub;
```

```
@end
```

```
...
```

```
@interface MyClass : MySuperClass <MyProtocol>
```

```
@end
```

- L'equivalente delle interfacce Java
- Permettono metodi opzionali (da Objective C 2.0)

Forwarding

- In Objective C è molto comune fare forwarding, ossia utilizzare ampiamente degli strategy (delegare)
- Un oggetto riceve un messaggio, non ha un metodo corrispondente, e delega ad un altro oggetto il compito
 - SEL è il data type che contiene il nome di un metodo
 - @selector è una pseudofunction per ottenere un SEL

Forwarding

```
- (void) forwardInvocation:
(NSInvocation*)invocation;
{
    SEL s = [invocation selector];
    if ([helper respondsToSelector:s])
    {
        [invocation invokeWithTarget:helper];
        NSLog(@"last exec -> thru domain");
    }
    else
        [self doesNotRecognizeSelector:s];
}
```

ID Objects

- E' il tipo più generico in Objective C
 - ad ID si può assegnare qualsiasi altro oggetto
- Un po' come Object (NSObject in Cocoa), ma senza controllo di tipo
 - un ID può essere QUALSIASI cosa, e ci si può invocare sopra QUALSIASI metodo
 - ovviamente l'oggetto potrà risponderci che non conosce il messaggio!

Uso di ID

- [TODO]

Altre differenze...

- Non ci sono namespace
 - per cui si usa anteporre due caratteri ai nomi delle classi
 - NS per esempio sta per NextStep, ed è anteposto a tutti gli oggetti Cocoa
- NIL risponde ai metodi, dunque non c'è mai una NullPointerException!
- Un esempio di pattern NullObject

Altre differenze...

- Le eccezioni ci sono, ma vanno abilitate esplicitamente
 - Non sono comunemente usate in objective C
- [TODO]

Libri e riferimenti vari

- Object Oriented Programming, an evolutionary approach, Brad Cox
- Mac OSX Internals, Amith Singh
 - <http://osxbook.com/>
- Cocoa Programming for Mac OSX 3rd edition, A. Hillegas
- Advanced OSX Programming, A.Hillegas
- Guida ObjC 2.0
 - <http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC>