Da Java a Objective-C: porting e dispositivi portatili

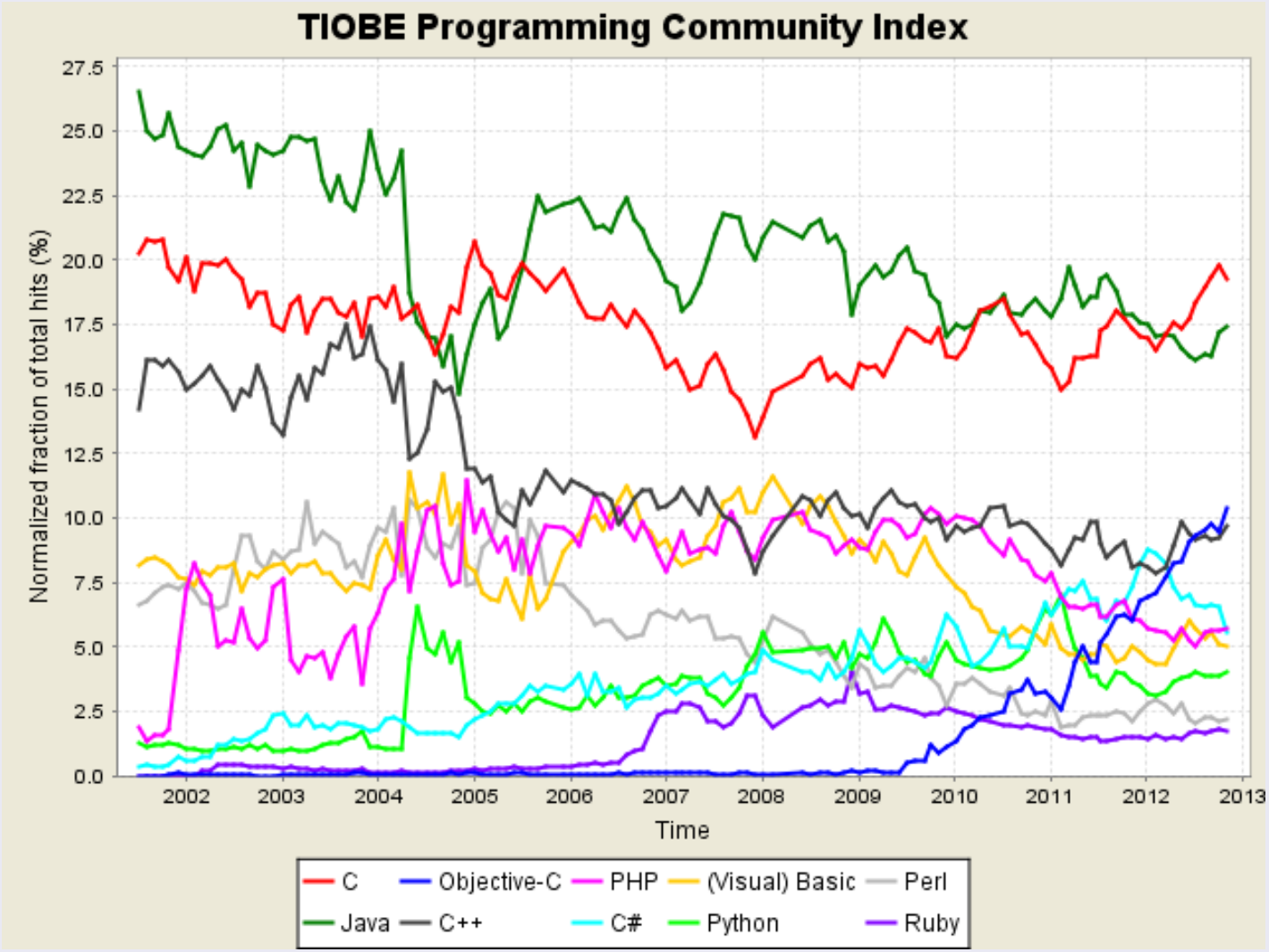
Nompleggio Pietro Antonio

II Facoltà di Ingegneria - Università di Bologna

pietro.nompleggio@studio.unibo.it

Dicembre 20, 2012

Perchè Java e Objective-C?



Motivazioni & Contesto

Due Linguaggi:

I due linguaggi principali per sviluppare per dispositivi mobili sono Java (linea verde) e Objective-C (linea blu) che ha avuto una forte crescita a seguito dell'uscita dell'iPhone.

Porting

Purtroppo non è possibile eseguire una applicazione scritta in Java in dispositivi mobili Apple, perchè non è presente la Java Virtual Machine, c'è quindi bisogno di un porting. Il porting è un adattamento o una modifica di un software per consentirne l'uso in un ambiente di esecuzione diverso da quello originale.

Perchè farlo?

Gli sviluppatori che hanno realizzato un'app in Java potrebbero svilupparla anche in Objective-C per poter ampliare il loro mercato, ma questo vorrebbe dire eseguire il porting, e non sempre è possibile.

Motivazioni & Contesto

Due Linguaggi:

I due linguaggi principali per sviluppare per dispositivi mobili sono Java (linea verde) e Objective-C (linea blu) che ha avuto una forte crescita a seguito dell'uscita dell'iPhone.

Porting

Purtroppo non è possibile eseguire una applicazione scritta in Java in dispositivi mobili Apple, perchè non è presente la Java Virtual Machine, c'è quindi bisogno di un porting. Il porting è un adattamento o una modifica di un software per consentirne l'uso in un ambiente di esecuzione diverso da quello originale.

Perchè farlo?

Gli sviluppatori che hanno realizzato un'app in Java potrebbero svilupparla anche in Objective-C per poter ampliare il loro mercato, ma questo vorrebbe dire eseguire il porting, e non sempre è possibile.

Motivazioni & Contesto

Due Linguaggi:

I due linguaggi principali per sviluppare per dispositivi mobili sono Java (linea verde) e Objective-C (linea blu) che ha avuto una forte crescita a seguito dell'uscita dell'iPhone.

Porting

Purtroppo non è possibile eseguire una applicazione scritta in Java in dispositivi mobili Apple, perchè non è presente la Java Virtual Machine, c'è quindi bisogno di un porting. Il porting è un adattamento o una modifica di un software per consentirne l'uso in un ambiente di esecuzione diverso da quello originale.

Perchè farlo?

Gli sviluppatori che hanno realizzato un'app in Java potrebbero svilupparla anche in Objective-C per poter ampliare il loro mercato, ma questo vorrebbe dire eseguire il porting, e non sempre è possibile.

Java e Obj-C: Confronto (1/4)

Classi e File

La gestione delle classi nei file è una delle principali differenze tra questi due linguaggi:

- Java: usa un unico file con estensione **.java** per dichiarare ed implementare una classe.
- Objective-C: usa due file per poter dichiarare ed implementare una classe, l'header file **.h** dove viene dichiarata la classe, i campi e i metodi; e il source file **.m** dove viene implementata la classe.

Creazione Oggetti

Gli oggetti in Java e Objective-C vengono creati in maniera diversa:

- In Java per creare un oggetto serve la parola chiave **new** es: `Counter c = new Counter();`
- In Objective-C per creare un oggetto si usano più parole chiave, si usa il prefisso * prima del nome della variabile per indicare che è un oggetto, e poi **alloc**, **init**, es: `Counter *c = [[Counter alloc] init];`

Java e Obj-C: Confronto (1/4)

Classi e File

La gestione delle classi nei file è una delle principali differenze tra questi due linguaggi:

- Java: usa un unico file con estensione **.java** per dichiarare ed implementare una classe.
- Objective-C: usa due file per poter dichiarare ed implementare una classe, l'header file **.h** dove viene dichiarata la classe, i campi e i metodi; e il source file **.m** dove viene implementata la classe.

Creazione Oggetti

Gli oggetti in Java e Objective-C vengono creati in maniera diversa:

- In Java per creare un oggetto serve la parola chiave **new** es: `Counter c = new Counter();`
- In Objective-C per creare un oggetto si usano più parole chiave, si usa il prefisso * prima del nome della variabile per indicare che è un oggetto, e poi **alloc**, **init**, es: `Counter *c = [[Counter alloc] init];`

Java e Obj-C: Confronto (2/4)

Package e Framework

- In Java abbiamo i **package** che sono delle collezioni di classi, che costituiscono delle vere e proprie librerie, il package principale che racchiude le principali classi è il **java.lang**;
- In Objective-C le librerie si chiamano **framework** e la libreria principale equivalente al java.lang è il **Foundation Framework.**

Ereditarietà

L'ereditaerità è uno dei concetti fondamentali nel paradigma di programmazione ad oggetti, è una relazione tra due classi, e in entrambi i linguaggi abbiamo una **ereditarietà singola**.

- In Java tutte le classi derivano in maniera diretta o indiretta da una classe comune: **Object**, è poi lo sviluppatore che decide se estendere un'altra classe;
- In Objective-C tutte le classi derivano in maniera diretta o indiretta da una classe comune **NSObject** e anche qui lo sviluppatore può scegliere se ereditare da un'altra classe.

Java e Obj-C: Confronto (2/4)

Package e Framework

- In Java abbiamo i **package** che sono delle collezioni di classi, che costituiscono delle vere e proprie librerie, il package principale che racchiude le principali classi è il **java.lang**;
- In Objective-C le librerie si chiamano **framework** e la libreria principale equivalente al java.lang è il **Foundation Framework.**

Ereditarietà

L'ereditaerità è uno dei concetti fondamentali nel paradigma di programmazione ad oggetti, è una relazione tra due classi, e in entrambi i linguaggi abbiamo una **ereditarietà singola**.

- In Java tutte le classi derivano in maniera diretta o indiretta da una classe comune: **Object**, è poi lo sviluppatore che decide se estendere un'altra classe;
- In Objective-C tutte le classi derivano in maniera diretta o indiretta da una classe comune **NSObject** e anche qui lo sviluppatore può scegliere se ereditare da un'altra classe.

Java e Obj-C: Confronto (3/4)

Poliformismo

Il Poliformismo è un altro concetto fondamentale del paradigma ad oggetti, il suo significato è relativo proprio alla sua definizione, ossia avere più forme, più aspetti. **Entrambi** i linguaggi offrono pieno supporto al poliformismo, ovvero è possibile usare un riferimento di tipo superclasse per riferire un oggetto di tipo sottoclasse.

```
- BCounter bC = new Counter(); \\Java
- BCounter *bC = [[Counter alloc] init]; \\Objective-C
```

Overloading e Overriding

Grazie al poliformismo possiamo introdurre due concetti come l'**overloading** e l'**overriding**.

- In Java sono presenti **entrambi** i concetti;
- In Objective-C è presente **solo** l'overriding, ma **non** l'overloading.

Java e Obj-C: Confronto (3/4)

Poliformismo

Il Poliformismo è un altro concetto fondamentale del paradigma ad oggetti, il suo significato è relativo proprio alla sua definizione, ossia avere più forme, più aspetti. **Entrambi** i linguaggi offrono pieno supporto al poliformismo, ovvero è possibile usare un riferimento di tipo superclasse per riferire un oggetto di tipo sottoclasse.

```
- BCounter bC = new Counter(); \\Java
- BCounter *bC = [[Counter alloc] init]; \\Objective-C
```

Overloading e Overriding

Grazie al poliformismo possiamo introdurre due concetti come l'**overloading** e l'**overriding**.

- In Java sono presenti **entrambi** i concetti;
- In Objective-C è presente **solo** l'overriding, ma **non** l'overloading.

Java e Obj-C: Confronto (4/4)

Metodi

I metodi costituiscono il comportamento degli oggetti e quindi della classe:

- In Java possiamo definire un metodo in questo modo:
  `public <ParamRitorno> <NomeMetodo> (<ParamList>);`
  Per richiamare un metodo si usa la notazione puntata ”.”
- In Objective-C invece la sintassi è leggermente diversa:
  `- (ParamRitorno) <NomeMetodo> : (<ParamList>);`
  Per richiamare un metodo si usa la notazione con le parentesi **[ ]**

Array

- Le librerie standard di Java forniscono la classe **ArrayList** come versione “dinamica” di array, in cui il numero di elementi può variare a runtime;
- In Objective-C invece l’array che più assomiglia a un ArrayList di Java è un **NSMutableArray**.

Java e Obj-C: Confronto (4/4)

Metodi

I metodi costituiscono il comportamento degli oggetti e quindi della classe:

- In Java possiamo definire un metodo in questo modo:
  `public <ParamRitorno> <NomeMetodo> (<ParamList>);`
  Per richiamare un metodo si usa la notazione puntata **"."**
- In Objective-C invece la sintassi è leggermente diversa:
  `- (ParamRitorno) <NomeMetodo> : (<ParamList>);`
  Per richiamare un metodo si usa la notazione con le parentesi **[ ]**

Array

- Le librerie standard di Java forniscono la classe **ArrayList** come versione "dinamica" di array, in cui il numero di elementi può variare a runtime;
- In Objective-C invece l'array che più assomiglia a un ArrayList di Java è un **NSMutableArray**.

Da Java a Objective-C

Porting

Dopo aver esaminato le principali differenze e somiglianze tra questi due linguaggi, si può pensare di eseguire un porting da Java a Objective-C, per farlo ci sono due alternative:

- Porting manuale, comporta la riscrittura completa del codice, un lavoro molto lungo ma sicuramente il più sicuro e completo;
- Porting automatico, tramite strumenti di conversione che permettono la traduzione da un codice ad un'altro, non sempre funzionanti come dovrebbero;

j2Objc

Dopo molte ricerche e prove eseguite per cercare un tool che permettesse questo passaggio in automatico, l'unico che si propone di tradurre codice Java in codice Objective-C è un tool attualmente in sviluppo da Google **j2objc**, questo tool permette la traduzione di codice senza interfaccia grafica.

Da Java a Objective-C

Porting

Dopo aver esaminato le principali differenze e somiglianze tra questi due linguaggi, si può pensare di eseguire un porting da Java a Objective-C, per farlo ci sono due alternative:

- Porting manuale, comporta la riscrittura completa del codice, un lavoro molto lungo ma sicuramente il più sicuro e completo;
- Porting automatico, tramite strumenti di conversione che permettono la traduzione da un codice ad un'altro, non sempre funzionanti come dovrebbero;

j2Objc

Dopo molte ricerche e prove eseguite per cercare un tool che permettesse questo passaggio in automatico, l'unico che si propone di tradurre codice Java in codice Objective-C è un tool attualmente in sviluppo da Google **j2objc**, questo tool permette la traduzione di codice senza interfaccia grafica.

Sperimentazioni con j2Objc (1/3)

Insistendo sulle somiglianze e sulle differenze tra Java a Objective-C evidenziate nella tesi, ho realizzato alcuni esempi in Java e successivamente in Objective-C per verificare la correttezza del tool sviluppato da Google, sia dal punto di vista del codice generato che dei possibili "mapping" tra i due linguaggi. Ecco un esempio significativo della traduzione di un metodo Java:

```
public boolean addElementInArray(Integer i) {
boolean result = false;
for (Integer a:myArray) {
if (a.intValue() == i.intValue()) {
result = true;
break; }
}
if (!result) {
myArray.add(i);
return true;
 }
return false;
 }
```

Sperimentazioni con j2Objc (2/3)

Risultato della traduzione con il tool:

```
- (BOOL)addElementInArrayWithJavaLangInteger:(
    JavaLangInteger *)i {
BOOL result = NO; {
id<JavaLangIterable > myArray_;
array__ = (id<JavaLangIterable >)
if (!array__) {
@throw [[[JavaLangNullPointerException alloc] init]
autorelease];
...
if (!result) {
[((JavaUtilArrayList *) NIL_CHK(myArray_)) addWithId:i
    ];
return YES;
}
return NO;
}
```

Sperimentazioni con j2Objc (3/3)

Considerazioni sulla traduzione

Il risultato non è quello che ci saremmo aspettati:

- Vengono create classi che non appartengono al Foundation Framework di Objective-C, es: **JavaLangInteger**,**JavaLangIterable**;
- Non vengono tradotte come dovrebbero neanche le classi base come per esempio la traduzione di un ArrayList di Java che come ho spiegato equivale a un NSMutableArray in Objective-C, invece il tool crea anche qui una nuova classe chiamata **JavaUtilArrayList**;
- L'unica cosa che può essere corretta è la struttura del metodo come parametri in ingresso e variabili di ritorno, però tutto il resto è sbagliato.

Conclusioni

Il tool sviluppato da Google ha diversi problemi:

- Non è un traduttore, ma un emulatore, perchè crea nuove classi che non esistono in Objective-C ma che si appoggiano ad una loro libreria che emula il codice Java;
- Se voglio esportare il codice tradotto, in questo pseudo-Objective-C non posso senza esportare anche le loro librerie di emulazione, senza le quali non funziona niente;
- Se voglio effettuare delle modifiche al codice non posso farlo, perchè non esiste una documentazione su come usare queste nuove classi create come la JavaUtilArrayList;

Al momento l'unica soluzione possibile per effettuare un porting da Java a Objective-C è la traduzione manuale, che garantisce la correttezza e la compatibilità. Ho dimostrato all'interno della tesi tramite la realizzazione di alcuni esempi che passare manualmente tra i due linguaggi è possibile, e per ora rimane la miglior soluzione.

Demo video


Counter Demo
0
Inc
Dec
Reset
Set
Gestore
2:00 AM
Menu
Counter
0
+
-
Reset

Grazie!