Solidan pokušaj unapređenja C-a

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Dalma Beara, Denis Aličić, Mateja Marjanović, Anja Miletić
beara.dalma@gmail.com, a.denis96@gmail.com,
mateja.marjanovic96@gmail.com, anya.miletic@gmail.com

30. april 2019

Sažetak

U ovom radu je opisan nastanak i razvoj programskog jezika Objective-C, kao i njegova uloga u razvoju drugih jezika koji imaju široku primenu (Java, Swift) kao i bitnih koncepata koji su počevši od Objective-C ušli u upotrebu (interfejsi). Predstavljene su osnovne karakteristike jezika i najbitniji koncepti koje jezik, na kom je zasnovan operativni sistem MAC OS, implementira kao i najpoznatija razvojna okruženja (eng. *framework*).

Sadržaj

1 Uvod

Objective-C je objektno-orijentisan programski jezik namenjen kako za sistemsko programiranje, tako i za izradu korisničkih aplikacija. Nastao iz potrebe za većom produktivnosti programera. Kroz naredna poglavlja biće predstavljen uticaj jezika na svet kakav danas poznajemo kao i primeri u kojima je korišćen Objective-C za implementaciju čitavih operativnih sistema i korisničkih okruženja.

2 Nastanak i istorijski razvoj, mesto u razvojnom stablu, uticaji drugih programskih jezika

U ovoj sekciji biće naveden istorijski razvoj, šta je sve na njega uticalo, kao i na šta je on uticao.

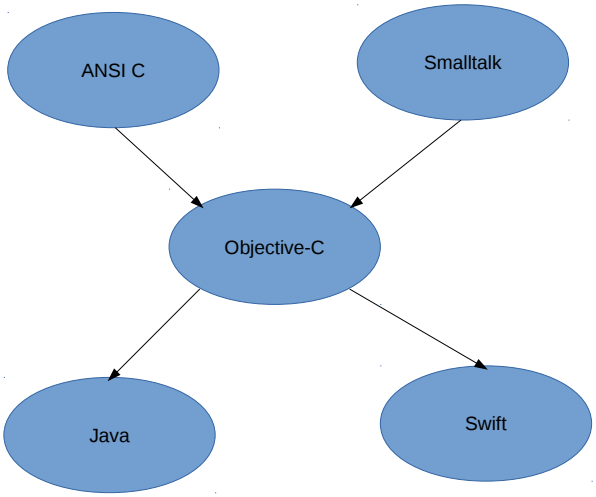
2.1 Nastanak i istorijski razvoj

Iako se pojam jezika Objective-C primarno vezuje za proizvode kompanije Epl (eng. *Apple*) – MAC OS X, iPhone itd, on je zapravo nastao mnogo pre njih. Njegove temelje postavili su Bred Koks i Tom Lav 1981. godine, težeći da nađu način za povećavanje produktivnosti programera [**?**]. Te godine pojavio se novi, revolucionarni jezik Smalltalk [**?**], koji je tada unapredio koncept objektno-orijentisanog programiranja. U osnovi tog jezika bilo je posmatranje programa kao skupova objekata koji su mogli da komuniciraju jedni sa drugima dinamički pozivajući metode. To je omogućilo da se stanje programa menja pod uticajem korisnika. Koks je u ovoj ideji video mogućnost da se vrtoglavo ubrza pisanje programa, pošto su se u njemu mogle praviti biblioteke objekata i posle koristiti u drugim programima bez izmene. Međutim, Smalltalk je bio veoma spor i zahtevao je da se svi programi pišu i pokreću u posebnom okruženju. Tada se javila potreba za spajanjem objektno-orijentisanih ideja Smalltalk-a i brzog jezika C 1. Koks je 1983. objavio naučni rad u kom je predstavio objektno-orijentisani prekompilator (eng. *Object-Oriented Precompiler*). Da bi ovu svoju ideju izbacili na tržište, njih dvojica su osnovali kompaniju Stepstoun (eng. *Stepstone*), izmenili kompilator za OOPC i preimenovali jezik u Objective-C. Iako je glavna ideja ovog projekta bila pisanje i prodavanje pomenutih biblioteka koje su mogle ponovo da se koriste, korisnici su imali dosta zamerki, te je jezik sve više poprimao karakteristike objektno-orijentisanih jezika kakve ih danas znamo, kao što su sakupljač otpada (eng. *garbage collector*) i interpreter, te je kompanija uskoro propala. Stiv Džobs je 1988. kupio licencu, a nekoliko godina kasnije i sva prava za Objective-C za potrebe svoje kompanije NeXT Computer koja se 1996. pripojila Apple-u. Sa tom tranzicijom, sam jezik je pretrpeo razne promene, a najvažnije su uvođenje dveju novih „komponenti": kategorija (eng. *categories*), danas poznatih kao ekstenzije (eng. *extensions*) i protokola (eng. *protocols*). Kategorije su služile da programerima omoguće da sami dodaju funkcije u objekte iz već postojećih biblioteka, a protokoli su omogućavali objektima da komuniciraju jedni s drugima. Oni su danas podržani u jeziku Java i poznati su kao interfejsi (eng. *interfaces*).

Najpopularniji jezici na koje je uticao Objective-C su Swift i Java, a navode se još i Groovy, TOM, Nu i Objective-J.

Tabela 1: Uticaj jezika na Objective-C

jezik	preuzeto
C	imperativni koncepti
Smalltalk	objektno-orijentisani koncepti



Slika 1: Razvojno stablo

2.2 Uticaj drugih programskih jezika

Kao što je već navedeno, Objective-C je nastao kao kombinacija koncepata na kojima počiva Smalltalk utočenih u sintaksu jezika C. Isprva je čak funkcionisao tako što je kod prevođen na C i onda izvršavan. S druge strane, on je uticao na jezik Swift koji je razvio Epl. Swift [**?**] se često naziva i „Objective-C bez C-a" 1.

3 Osnovna namena programskog jezika, svrha i mogućnosti

Krajem 80-ih godina prošlog veka, popularnost Objective-C jezika je rasla sa razvojem NeXT sistema. Najviše se koristi za razvoj softvera za Apple iOS operativni sistem. Uticaj Objective-C jezika je vidljiv u Java programskom jeziku. Objective-C definiše mali ali moćan skup ekstenzija ANSI C programskog jezika koje omogućavaju objektno-orijentisano programiranje. Cocoa 5.1, Apple-ov API za macOS je napisan uz pomoć Objective-C jezika, takodje veliki deo aplikacija je napisan koristeći taj jezik. Međutim, ono što treba napomenuti je da je, po istraživanju portala Stackoverflow, ovaj jezik rangiran kao jedan od najnepoželjnijih mežu

programerima koji ga koriste.

4 Osnovne osobine ovog programskog jezika, podržane paradigme i koncepti

Objective-C je pravi nadskup programskog jezika C koji pruža objektno-orijentisane sposobnosti i dinamičko izvršno okruženje (eng. *runtime*). Objective-C nasleđuje sintaksu, primitivne tipove i naredbe za tok izvršavanja od jezika C, a dodaje sintaksu za definisanje klasa i metoda, kao i mnoge biblioteke i ugrađene klasne tipove. Takodje omogućava dinamičko tipiziranje i povezivanje.

4.1 Osnovni koncepti

U predstojećim pasusima biće opisani neki zanimljivi koncepti programskog jezika Objective-C.

4.1.1 Klasa NSObject

Klasa NSObject predstavlja baznu klasu za većinu klasnih hijerarhija u ovom jeziku. Ona svojim naslednicama (na primer NSString, NSArray, NSNumber, NSDate...) omogućava da se ponašaju kao objekti i pruža im osnovni interfejs ka rantajm (*runtime*) sistemu.

Na slici ispod možemo videti jednu tipičnu deklaraciju korisnički definisane klase.

4.1.2 Klasteri klasa

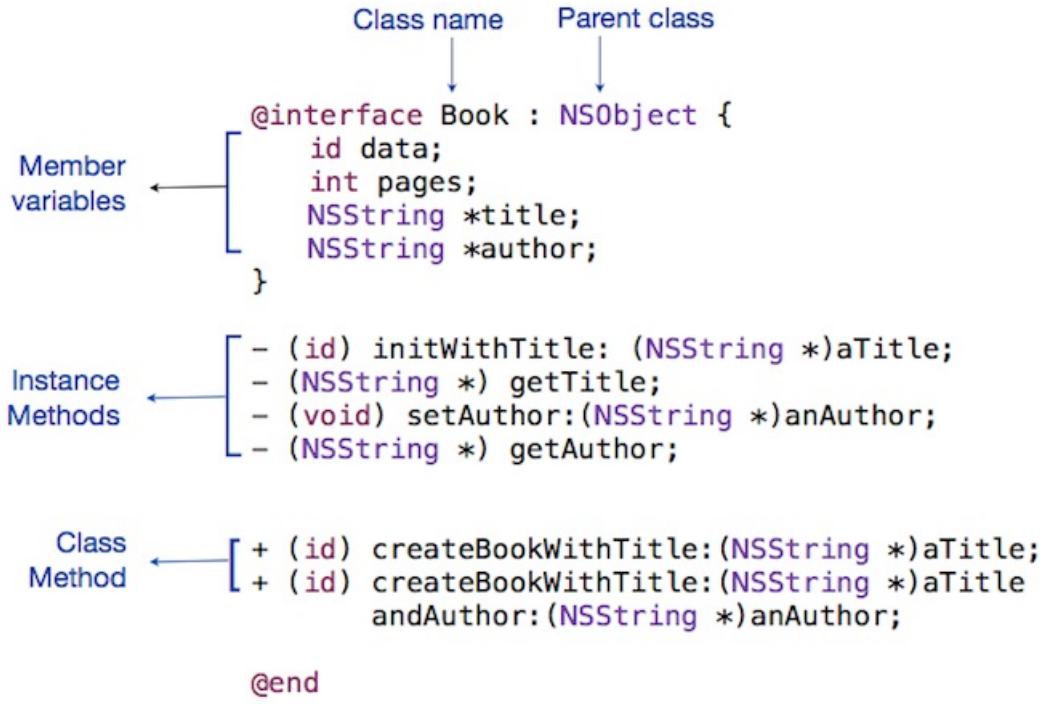
Klasteri klasa (eng. *class clusters*) su programski šablon koji se često koristi u pisanju programa na Objective-C jeziku. Klasteri klasa grupišu određeni broj privatnih podklasa unutar javne apstraktne nadklase. Ovakvo grupisanje pojednostavljuje javno vidljivu arhitekturu objektno-orijentisanih biblioteka i radnih okvira, bez smanjenja njihove funkcionalnosti. Apstraktna nadklasa mora da deklariše metode za kreiranje instanci svojih privatnih podklasa.

4.1.3 Introspekcija

Introspekcija je moćno svojstvo objektno-orijentisanih jezika i okruženja. Ono predstavlja mogućnost objekta da predstavi detalje o sopstvenoj implementaciji u toku izvršavanja programa. Ti detalji mogu biti njegovo mesto u drvetu nasleđivanja, da li implementira određeni protokol i da li odgovara na neku poruku. Primer:

```
1000 while ( id anObject = [objectEnumerator nextObject] ) {
         if ( [self class] == [anObject superclass] ) {
1002         // do something appropriate...
         }
1004 }
```

Listing 1: introspekcija



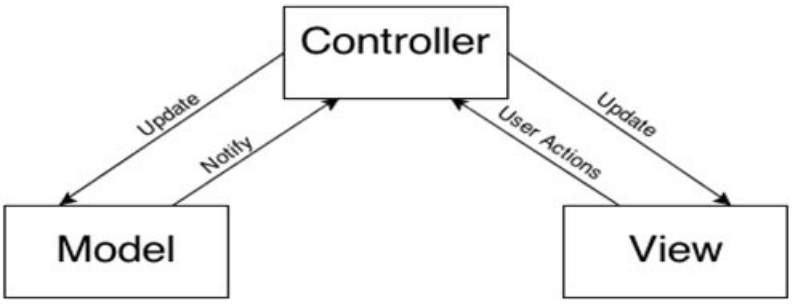
Slika 2: korisnički definisana klasa

4.1.4 Alokacija objekata

Prilikom alokacije objekata, dodeljuje se zahtevana memorija iz regiona virtualne memorije dostupne programu. Da bi se izračunalo koliko memorije treba alocirati, uzimaju se u obzir instancirane promenjlive tog objekta - uključujući njihove tipove i redosled - kao što je navedeno u klasi objekta.

4.1.5 MVC

MVC (eng. *Model-View-Controller*) [**?**] je programski šablon (eng. *design pattern*) visokog nivoa koji predstavlja globalnu arhitekturu aplikacije i razvrstava objekte prema ulozi koju imaju u toj aplikaciji. Objektno-orijentisano programiranje ima više koristi od ovog modela; povećava se modularnost korišćenih objekata, njihovi interfejsi su bolje definisani. Sami programi se bolje adaptiraju promeni zahteva. MVC smatra da postoje tri vrste objekata: model, pogled i kontroler objekti. MVC šablon definiše uloge koje ovi objekti imaju u aplikaciji i način na koji komuniciraju 3. Pri dizajniranju aplikacije, bitan korak je biranje - ili pravljenje - klasa za objekte koji pripadaju jednoj od ovih uloga.



Slika 3: MVC arhitektura

5 Najpoznatija okruženja (framework) za korišćenje ovog jezika i njihove karakteristike

5.1 Cocoa API

Cocoa [**?**] je objektno-orijentisano okruženje za Mac OS X operativni sistem. Uz korišćenje Cocoa Touch dodatka za prepoznavanje pokreta i realizaciju animacije moguća je izrada aplikacija i na iOS operativnom sistemu koji se koristi na mobilnim uređajima kao što su iPhone, iPod i iPad. Cocoa aplikacije se razvijaju pomoću Xcode i Interface Builder alata, i razvijaju se u programskom jeziku Objective-C. Pored nabrojanih alata Cocoa aplikaciju je moguće pisati u jednostavnom uređivaču teksta i napraviti izvršnu verziju programa pomoću GCC prevodioca iz komandne linije.

5.1.1 Istorijat razvoja Cocoa

Cocoa je izveden iz NeXTSTEP i OPENSTEP programerskih okruženja razvijenih u NeXT-u kasnih 1980-tih. Apple je kupio NeXT u decembru 1996. godine, i postepeno je počeo da radi na Rhapsody operativnom sistemu koji je trebao da bude direktni naslednik OPENSTEP-a. Trebao je da ima emulacionu bazu za Mac OS aplikacije pod imenom Blue Box. OPENSTEP baza biblioteka i binarna podrška su nazvani Yellow Box. Rhapsody je evoluirao u Mac OS X, a Yellow Box je postao Cocoa. Kao rezultat, Cocoa klase počinju sa skraćenicom „NS" (za NeXT-Sun kreaciju OPENSTEP-a): NSString, NSArray, i tako dalje.

Pre njegove sadašnje upotrebe „Cocoa" je bilo ime programa koji je omogućavao deci da kreiraju multimedijalne projekte. Prvobitno je bila poznata kao KidSim, a sada je licencirana drugoj firmi kao Stagecast Creator. Program je prestao da se razvija u jednoj od racionalizacija koje su sprovedene nakon što se Stiv Džobs vratio u Apple. Ime je ponovo upotrebljeno kako bi se izbeglo odlaganje koje bi nastalo registrovanjem novog žiga (eng. *trademark*), uz saglasnost Stagecast-a da reklamira stari Cocoa pod novim imenom.

5.2 GNUstep

GNUstep [?] je besplatno, objektno-orijentisano okruženje, namenjeno za različite arhitekture koje teži jednostavnosti i elegantnosti. U potpunosti je kompatibilno sa Cocoa 5.1 okruženjem. Namenjeno je kako razvijanju naprednih desktop aplikacija, tako i serverskih aplikacija. Za razliku od Cocoa moguće je pisati programe za različite operativne sistemime (Solaris, GNU/Linux, NetBSD, OpenBSD, FreeBSD, Windows). Osnovni jezik za razvoj je Objective-C, ali GNUstep nije ograničen samo na to (Java, Ruby, Guile, Scheme).

5.2.1 Razvoj aplikacija

Za razvoj scenskih komponenti aplikacije je namenjen Gorm [?], dok ProjectCenter [?] predstavlja integrisano razvojno okruženje (eng. *IDE*). Korisničko okruženje Etoile [?] koje je moguće koristiti na različitim operativnim sistemima je u potpunosti implementirano koristeći GNUstep.

6 Instalacija i uputstvo za pokretanje na Linux/Windows operativnim sistemima

U ovoj sekciji biće navedena instalacija i sve što je za nju potrebno na Linux i Windows operativnim sistemima, kao i uputstva za pokretanje programa.

6.1 Instalacija na Linux operativnim sistemima

6.1.1 Instalacija neophodnih paketa

Da bismo mogli da kompajliramo programe napisane u Objective-C, potrebno je prvo da instaliramo par paketa. Za kompajliranje Objective-C programa koristi se GCC kompajler[?], uz još par dodataka. Prvo što treba da uradimo je da instaliramo GCC podršku za Objective-C [?].

```
1000 $ sudo apt-get install gobjc
```

Listing 2: instalacija GCC podrške

Treba da instaliramo radno okruženje (eng. *framework*) na kom se moderni Objective-C zasniva i bez kog ne bi imalo puno smisla raditi, iako je moguće. To okruženje se zove GNUstep [?] i instalira se komandama:

```
1000 $ sudo apt-get install gnustep
     $ sudo apt-get install gnustep-devel
```

Listing 3: instalacija GNUstep-a

6.1.2 Kompajliranje

Svaki put kada pokrenemo novu sesiju, moramo da pokrenemo skript GNUstep.sh, da bismo mogli da kompajliramo Objective-C programe.

```
$ chmod +x /usr/share/GNUstep/Makefiles/GNUstep.sh
$ /usr/share/GNUstep/Makefiles/GNUstep.sh
```

Da ne bismo svaki put morali da izvršavamo taj skript, možemo da uradimo sledeće:

```
$ echo /usr/share/GNUstep/Makefiles/GNUstep.sh >> ~/.
    bashrc
```

Listing 4: izvršavanje

Na taj način, svaki put kad se pokrene nova sesija, biće izvršen GNUstep.sh i nećemo morati da brinemo.

Kao što smo već naveli, Objective-C koristi GNU-ov C kompajler, uz dodatnu podršku. Sem standardnih argumenata, GCC-u su potrebni i dodatni argumenti (eng. *flags*), kao što su -lobjc, -lgnustep-base, itd.

Dakle ako želimo da prevedemo hello.m program, moramo da izvršimo sledeće:

```
$ gcc hello.m -o hello `gnustep-config --objc-flags` -
    lgnustep-base -lobjc
```

Listing 5: prevođenje programa

Izvršavanje programa se radi uobičajeno:

```
$ ./hello
```

Listing 6: izvršavanje programa

6.2 Instalacija na Windows operativnim sistemima

6.2.1 Kompajliranje i izvršavanje

Kada smo instalirali MinGW i GNUstep, možemo da kompajliramo i izvršavamo Objective-C programe u shell okruženju, koje pokrećemo tako što otvorimo Shell program koji se nalazi u GNUstep. Inicijalno se nalazimo u C:
GNUstep
home
<username> direktorijumu. Program ćemo kompajlirati sledećom naredbom:

```
$ gcc `gnustep-config --objc-flags` -L /GNUstep/System/
    Library/Libraries hello.m -o hello -lgnustep-base -
    lobjc
```

Listing 7: kompilacija

A izvršavanje sa:

```
./hello.exe
```

Listing 8: izvršavanje programa

7 Primer jednostavnog koda i njegovo objašnjenje

Najpre navodimo primer dobgo poznatog programa „Zdravo, svete".

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

int main (int argc, const char * argv[])
{
        NSAutoreleasePool *pool = [[NSAutoreleasePool
    alloc] init];
        NSLog (@"Hello, World!");
        [pool drain];
        return 0;
}
```

Listing 9: Zdravo, svete

Uključena je sistemska biblioteka Foundation.h, u kojoj se nalaze tipovi NSAutoreleasePool i funkcija NSLog. NSLog služi za ispisivanje poruke (tj. objekta) skladištene u promenljivoj tipa NSAutoreleasePool*. Na kraju programa komandom drain se šalje otpuštajući (engl. release) signal kojim se oslobađa memorija iz promenljive.

Navedimo zatim primer zanimljivog koda [**?**] koji omogućava kategorije (eng. *categories*). Kategorije su koncept jezika Objective-C koji omogućava korisnu funkcionalnost dodavanja metode klasi samo za određenu upotrebu. Dakle, koristeći kategorije, možemo dodati neku metodu u klasu, ali samo za svoju aplikaciju, tako da klasa zapravo ostane ista. Svaka metoda uvedena kroz kategoriju biće dostupna svim instancama te klase (u okviru aplikacije u kojoj je dodata, naravno), kao i instancama svih klasa koje je nasleđuju. Pri izvršavanju takvih metoda nema nikakve razlike u odnosu metode iz klase napravljene na tradicionalan način.

```
@interface NSString(MyAdditions)
+(NSString *)getCopyRightString;
@end
@implementation NSString(MyAdditions)
+(NSString *)getCopyRightString {
    return @"Zdravo";
}
@end
```

Listing 10: kategorije

Sintaksa kojom se najavljuje da će biti uvedena kategorija je ista kao i kad se deklariše klasa (interface, end, između definicija), sa dodatkom imena kategorije u zagradama:

```
1000 @interface ImeKlase (ImeKategorije)
@end
```

Listing 11: deklarisanje kategorije

Tu se posle znaka + navodi (u zagradama) tip, a potom ime metode koja će biti dodata klasi, a zatim se ključnim rečima implementation i end najavljuje da između njih sledi definicija nečega, u ovom slučaju gorenavedene kategorije. Tu se znakom + najavljuje dodavanje metode, a posle njenog tipa i imena u vitičastim zagradama sledi kod. Ovde smo definisali jednu jednostavnu metodu koja vraća nisku „Zdravo".

8 Zaključak

U ovom radu su opisane najvažnije osobine i koncepti jezika. Objective-C predstavlja osnovu operativnih sistema koji su u širokoj upotrebi i kao takav će i u budućnosti zauzimati zapaženo mesto u svetu informatike. Trajanje (skoro 40 godina) svedoči o kvalitetu jezika. Rad prestavlja podstrek za dalje istraživanje mogućnosti i funkcionalnosti jezika i njegovih okruženja.